Making Innovation Thrive™

Dialogic® Software Video Transcoder Developer's Reference Manual

Release 3.0

September 2010 64-1138-01 - Rev A

www.dialogic.com

Copyright and Legal Notice

Table Of Contents

Revision History

Revision	Release date	Notes
1.0	September 2005	DEH, Video Transcoder 2.0, Beta
1.1	November 2005	DEH, Video Transcoder 2.0
1.2	April 2007	LBG, Video Transcoder 2.1, Beta
1.3	July 2007	LBG, Video Transcoder 2.1
1.4	September 2010	Dialogic Rebranding
Last modified: September 2010		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

1. Introduction

The *Software Video Transcoder Developer's Reference Manual* contains the following topics:

- Software Video Transcoder (SVT) interface that is provided by the transcoder resource controller (TRC) module
- Programming model used to develop Software Video Transcoder applications
- Transcoder control features
- TRC functions
- Error and asynchronous event indications
- Video transcoder management interface module (VTMNG)
- Video transcoder platform used to provide the managed transcoder resources

This document is intended for video application developers. It assumes that you are familiar with wire line and wireless video concepts and the C programming language.

2. Software Video Transcoder overview

Overview of the Software Video Transcoder

The Software Video Transcoder is a software component that provides simplex or full-duplex transcoding between H.263 and MPEG-4 video streams. The transcoder converts bit streams from one format to another as supported by the following standards:

- ITU-T Recommendation H.263 baseline level 10, as defined in ITU-T Recommendation H.263 and 3GPP specifications TS.26.111, TS.26.911, TS.26.140.
- ISO/IEC 14496-2:2004 (MPEG-4 Video) Simple profile level 0, and 3GPP specifications TS.26.111, TS.26.911, TS.26.140.

The following table summarizes the transcoder capabilities:

Capability	**Description**
Video encoding formats	MPEG-4 simple profile level 0, 1, 2, and 3. H.263 baseline level 10, 20, and 30.
Video frame resolutions	(QCIF) Quarter Common Interchange Format (176 x 144). (CIF) Common Interchange Format (352 x 288).
Video encoding bit rate	25 to 384 kbit/s
Frame rates	4 to 30 fps
Video inputs	Input comes through RTP packetized streams.
IP network connections	RTP/RTCP/UDP/IP
RTP payload ID	MPEG-4: User configurable. Default is 100. H.263: 34 for RFC 2190 packetization. H.263: Dynamic (96 - 127 for RFC 2429 packetization).

Transcoder architecture

The transcoder includes the following main components:

Component	Description
Transcoder resource controller module (TRC)	Software that enables video applications to allocate and control video process channels on video transcoder platforms. Also provides the ability to overlay text and graphic content on top of a video stream.
Video transcoder platform	Physical platform on which the media transcoding takes place.

Component	Description
Video Transcoder Management Interface (VTMNG)	Software that provides the ability to manage and monitor video channels.

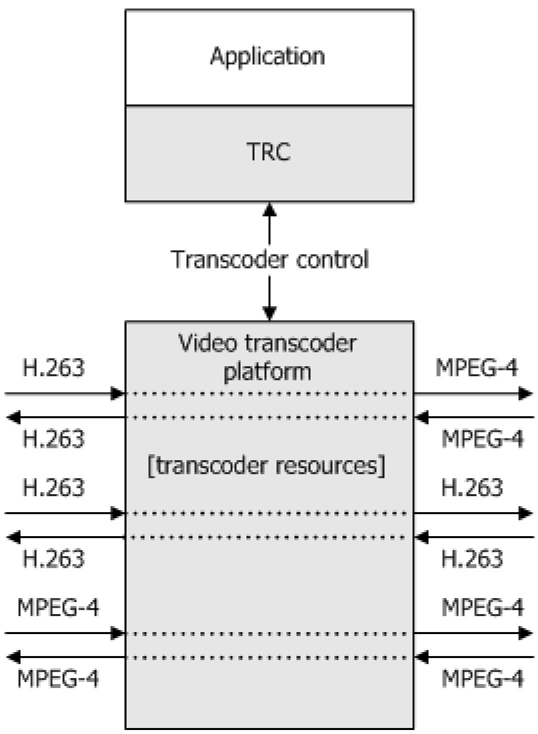
Transcoder resource controller module (TRC)

The transcoder resource controller module (TRC) is the software that enables video applications to allocate and control video process channels on video transcoder platforms.

The TRC supports media adaptation to and from different mobile video and IP video codecs. Media adaptation provides the framework for dynamically altering video content, such as the spatial and temporal resolution and encoding format of the multimedia content.

The TRC includes an overlay module that provides the ability to overlay text and graphic content on top of a video stream. The overlay can be scrolled vertically or horizontally and can be set to loop so that the overlay restarts when it reaches the end of the content. For example, you can create a banner message that can continuously scroll at the bottom of the image.

The following illustration shows the types of video transcoding that can be performed using the TRC to control transcoder resources:



A TCP/IP connection is maintained between the TRC on the host side and its counterpart on the video transcoder platform. The host TRC automatically reestablishes any connection that is lost and reports loss-of-connection and reconnection events to the controlling application.

Video transcoder platform

The video transcoder platform is the physical platform on which the media transcoding takes place. One TRC can control up to five video transcoder platforms. Once the video transcoder platform software is installed on a video transcoder platform, the video transcoder platform becomes a server to the TRC, on which media transcoding and overlay processing takes place.

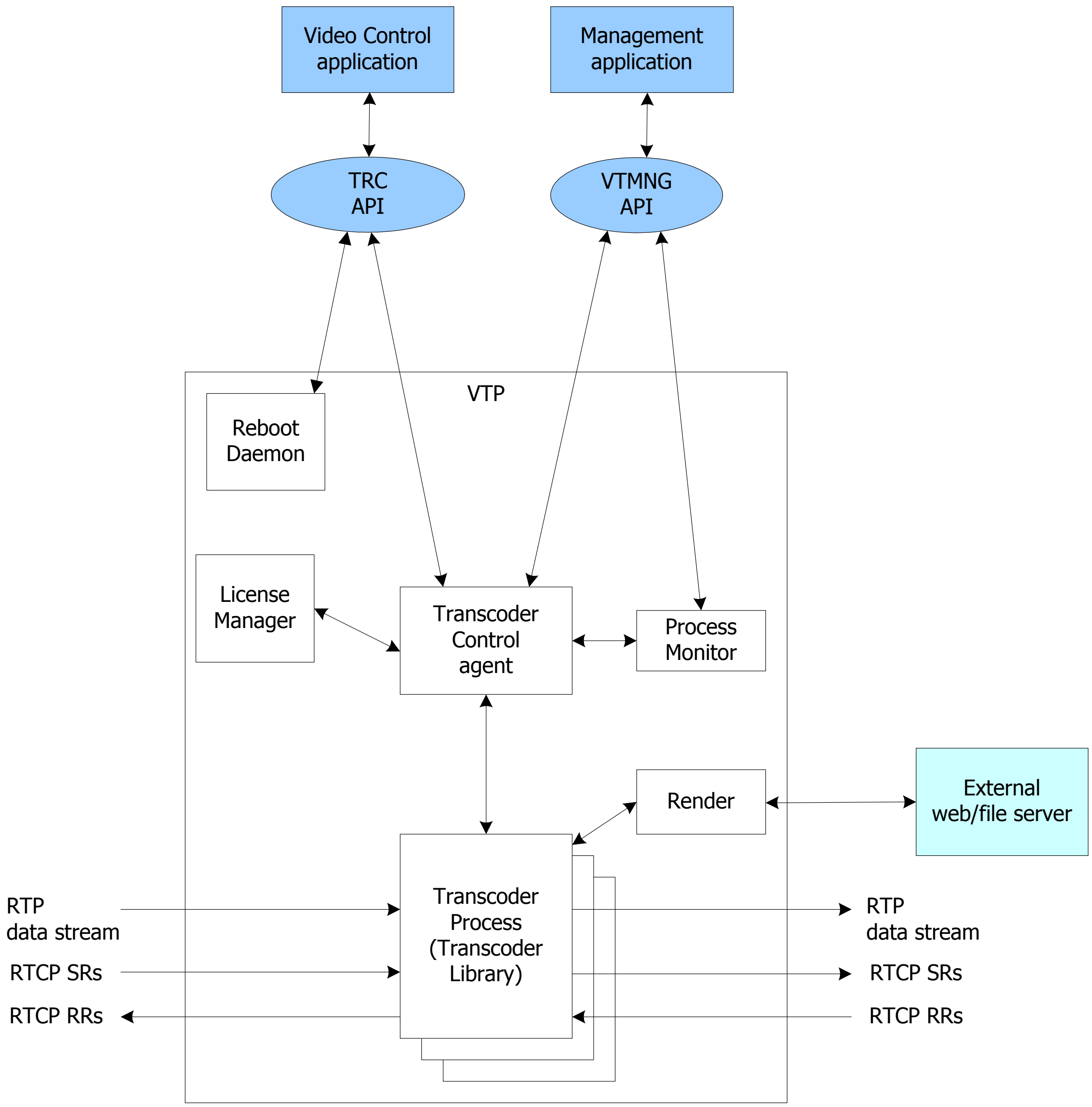
Each video transcoder platform provides a set of transcoding resources that the TRC manages. The total number of resources that a given video transcoder platform provides is based on the processing power of the platform and the number of available licenses. A simplex channel uses one license, a full-duplex channel uses two.

The TRC performs video transcoder resource allocation. Applications can request the overall video transcoder platform status and the status of each transcoder resource at any time. If necessary, an application can also use the TRC to reset a video transcoder platform.

The VTMNG API provides an interface for managing a video transcoder platform. A management application can view configuration, status, and statistical information for a specific video transcoder platform from the video transcoder platform-level, the application-level, or from a per-channel level.

A video transcoder system can consist of one or more standalone video transcoder platforms. You can create a standalone video transcoder platform by installing the video transcoder platform software on a hardware platform.

The following illustration shows the software components associated with a standalone video transcoder platform:



The following table describes the standalone video transcoder platform components:

Component	Description
Reboot daemon (*xscontrol*)	A background application that services requests from the transcoder resource controller module to reset a video transcoder platform. The video transcoder platform system control process can reset a video transcoder platform, even when the transcoder agent is unresponsive.

Component	Description
Transcoder Control Agent	An application that acts as a server to all TRC API and VTMNG API instances. Each TRC API instance establishes a TCP/IP connection to the transcoder agent. The transcoder agent receives all control requests over this interface and acts as the central control point for all video transcoder platform-based transcoding. Each VTMNG API instance uses UDP messaging to communicate with the control agent for all management activities.
License Manager	A server application that manages all deployed licenses on a given video transcoder platform.
Transcoder Process (*trcp*)	Process that performs video transcoding and overlay processing for the set of channels assigned by the transcoder agent. A number of transcoder processes (usually 30) exist on each video transcoder platform.
Process Monitor (*vtmon*)	A process that is responsible for starting and monitoring all other video transcoder processes. The Process Monitor takes whatever action is necessary to recover a process if the monitored process is lost.
Render (*trcr*)	A process that performs text and image content rendering for use in overlays that are requested through the control interface.
Transcoder Library	A set of functions that perform all video frame decoding, encoding, and overlay processing.

Video Transcoder Management Interface (VTMNG)

Use the Video Transcoder Management Interface (VTMNG) to do the following:

- Manage the video transcoder resources that are controlled using the TRC.
- Allow an external manager to query configuration, status, and statistical information about transcoder resources.
- Modify configuration information.
- Receive asynchronous event indications from transcoder resources by registering through the VTMNG API.
- Trigger recovery actions such as restarting or rebooting a VTP

The VTMNG API allows all video transcoder resource management to occur externally from the resource control interface provided by the TRC module. Multiple management applications can issue VTMNG API requests simultaneously.

Each video transcoder can be configured to issue asynchronous events (or traps) to an external application that is using the VTMNG API with the optional trap receive port active. A single client-side management application is usually used as a collection point for all video transcoder platforms. Alternatively, different sets of video transcoder platforms can be configured to send their traps to different management applications.

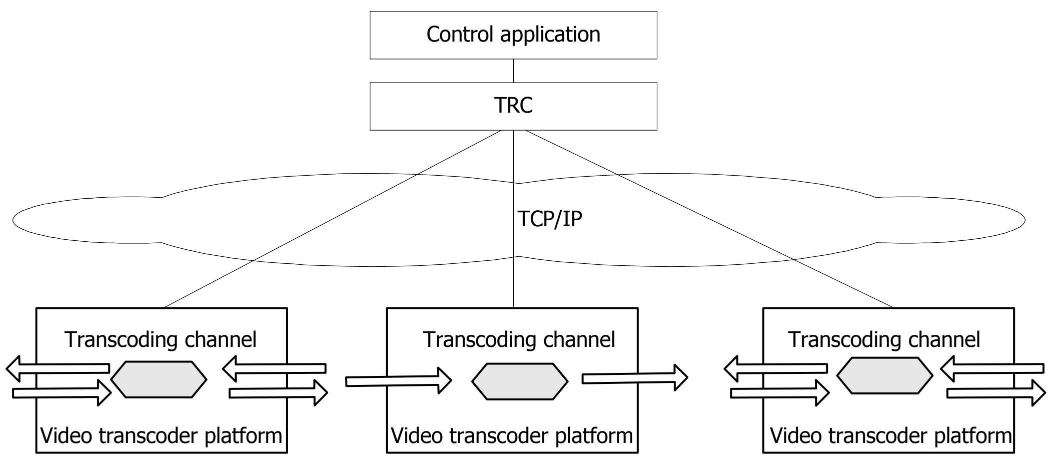
In addition to a complete management API, an operator console-based management utility is also provided (*vtmgr*). This utility provides full control over all video transcoder management capabilities; allowing for rapid deployment prior to development of any management-related applications.

For more information, refer to *Using the Video Transcoder Management Interface (VTMNG)* on page 28.

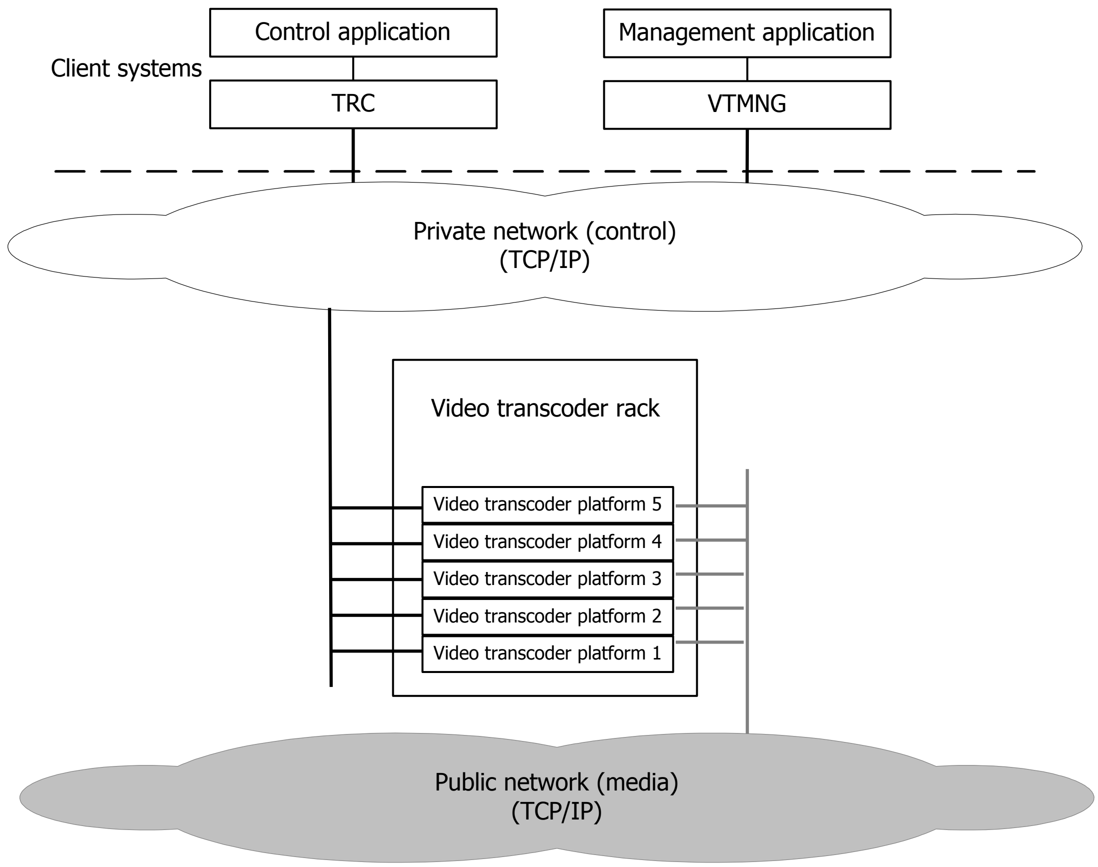
Application control of video transcoder platforms

A single application instance can control up to five video transcoder platforms. Each video transcoder platform can be connected to a private network and a public network. In this case, the TRC control and any management (VTMNG) takes place over the private network. Video media is transmitted and received over the public network.

The following illustration shows an application controlling three standalone video transcoder platforms:



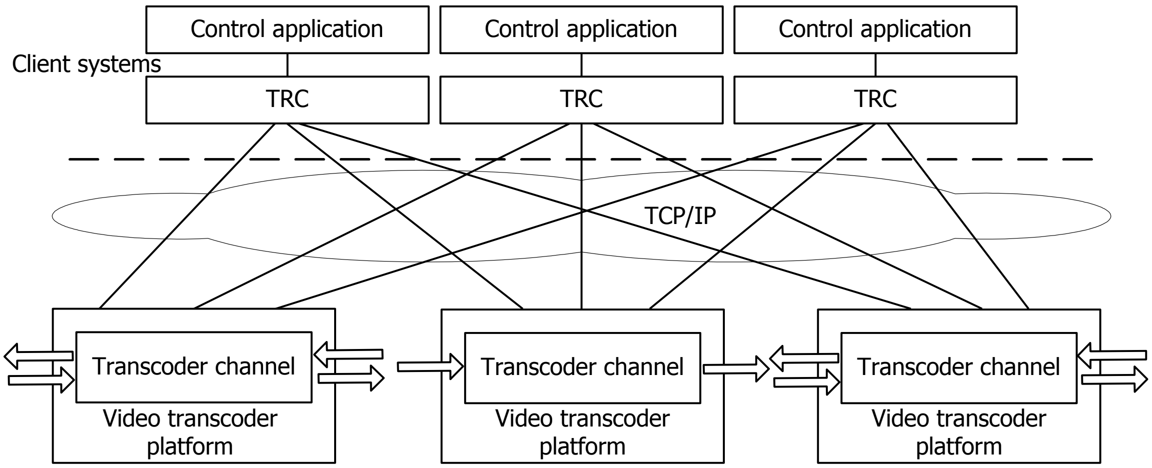
The following illustration shows an application controlling the resources provided by a set of video transcoder platforms. A separate management application is used to manage the platforms. Both applications connect to all video transcoder platforms through a private control network. Each video transcoder platform connects to the public network through a separate interface.



Multiple application support

The TRC supports multiple applications residing on the same client-side chassis and configurations in which applications are spread across multiple client-side chassis.

The following illustration shows the connection mapping between TRC instances and the video transcoder platforms being controlled by the TRC instances:



External communication protocols

The Software Video Transcoder supports the following communication protocols:

Component	Supported communication protocol
Media	RTP (transporting H.263 and MPEG4 video streams).
Media control	RTCP (with a control stream for each RTP media stream).
Management	The proprietary protocol used between the VTMNG API and the TRC Control Agent / Process Monitor.
Transcoder control	The proprietary protocol used between the TRC API and the TRC Control Agent / *xscontrol*.

Transcoder channel control

A video transcoder channel passes the received bit stream through a decoder and then passes the decoded data stream through an encoder to produce the output bit stream.

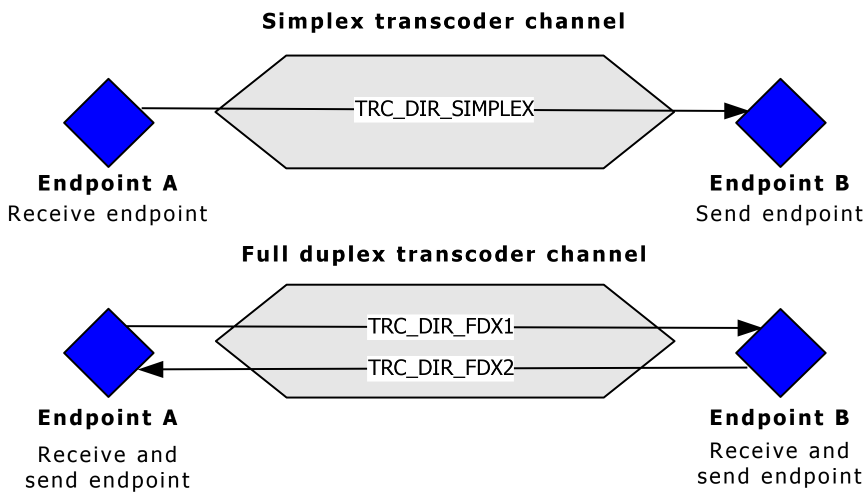
The following table provides a description of the types of transcoder channels:

Channel type	Number of licenses used	Description
Simplex	One	Performs decode/encode manipulation for data flowing in a single direction (TRC_DIR_SIMPLEX). In a simplex channel, the transcoder receives bit streams through endpoint A (the receive endpoint) and transmits them through endpoint B (the send endpoint).

Channel type	Number of licenses used	Description
Full-duplex	Two	Performs decode/encode manipulation for data flowing in two directions: The TRC_DIR_FDX1 direction performs decode/encode manipulation for data flowing from endpoint A to endpoint B. The TRC_DIR_FDX2 direction performs decode/encode manipulation for data flowing from endpoint B to endpoint A. In a full-duplex channel, each of the two endpoints act as both a receive endpoint and a send endpoint.

For more information, refer to *trcCreateVideoChannel* on page 116.

The following illustration describes the endpoints and transcoding directions of the RTP data flow for the simplex and full-duplex transcoder channels:



When you start a transcoding channel, the following type of information is sent to the TRC:

- General endpoint information, including the encoding type, profile, level, data rate, frame rate, and packetization mode.
- Channel input information, including optional RTCP configuration.
- Channel output information, including the IP address, port number, RTP payload ID, and type of service for the outbound packets.
- Decoder and encoder configuration information.
- Overlay channel options.
- Optional RTCP activation
- Optional set of image and text overlays defined.

For more information, refer to *trcStartVideoChannel* on page 145 and *Starting the transcoding process* on page 59.

Text and graphic overlays

In addition to video transcoding, the Software Video Transcoder provides text and graphic overlay functionality that can be used to create interactive menus or display corporate logos, among other things.

An overlay is an independent object that is associated with a specific transcoding channel direction. Overlays can only be created on existing transcoding channels. Each overlay can be started, stopped, and destroyed independently from other overlays.

The creation of a new overlay requires two different sets of information:

- The overlay type, a size and position (tTrcOvlConfig).
- A definition of the content to be displayed in the overlay (tTrcOvlContent).

Each transcoding channel direction creates up to 32 overlays of the following basic types of overlays:

Overlay type	Created from...
Text	Text strings provided by the controlling application and rendered into bitmap representations by the video transcoder platform. The video transcoder platform uses font files that are stored in the local file system or are accessible through NFS.
Graphic	Image files (jpeg, png or gif) that can be locally stored on the video transcoder platform or fetched from a web server (using http).

An overlay is of one of these types, not both. However, a graphic overlay can be used to display an image that already contains text, for example, a pre-rendered menu.

Each overlay can take up to the full size of the video frame and can overlap other overlays. Layers can be used to control display precedence of overlapping overlays. Transparency can be used to create complex overlay groups composed of text and graphics.

Text overlays

The creation of a text overlay is based on a text overlay content definition that provides the text string to be rendered and a font definition. The video transcoder platform provides cached and dynamic font rendering.

Cached font rendering uses the font definitions found in the *trcr.cfg* configuration file to provide fast rendering. These cached fonts require less processing at runtime but require fonts and sizes to be identified at video transcoder platform startup time. The character set supported by cached fonts is limited to ASCII characters.

Dynamic font rendering allows rendering in any size using a font file that the application specifies at runtime. While more flexible, dynamic font rendering requires more processing at runtime.

Graphic overlays

Graphic overlays are created from standard image files (jpeg, png or gif) obtained either locally or from a web server. The images can be resized dynamically to fit the overlay area. Cached images can also be defined in the *trcr.cfg* configuration file to minimize processing and shared memory footprint for commonly used images (logos, icons, and so on).

Customizing overlays

Options such as foreground colors, background colors, level of transparency, borders, and scrolling can be used to dynamically customize the appearance of overlays.

Rendering architecture overview

Text and image rendering is handled by a central rendering process. The rendering process is responsible for generating the cached fonts, rendering text, fetching and resizing the image files, and providing the resulting bitmap images in the appropriate format for the Software Video Transcoder processes that perform the actual overlaying.

The rendering process makes the resulting images available to the Software Video Transcoder processes by storing them in a shared memory area. The size of this area can be configured through the *trcr.cfg* file.

Each piece of content requested by a Software Video Transcoder process uses its own space within the shared memory. Only cached images defined at startup in *trcr.cfg* are shared across all Transcoder processes. Using cached images greatly reduces the shared memory usage since the same image is reused by all Software Video Transcoder processes that access it.

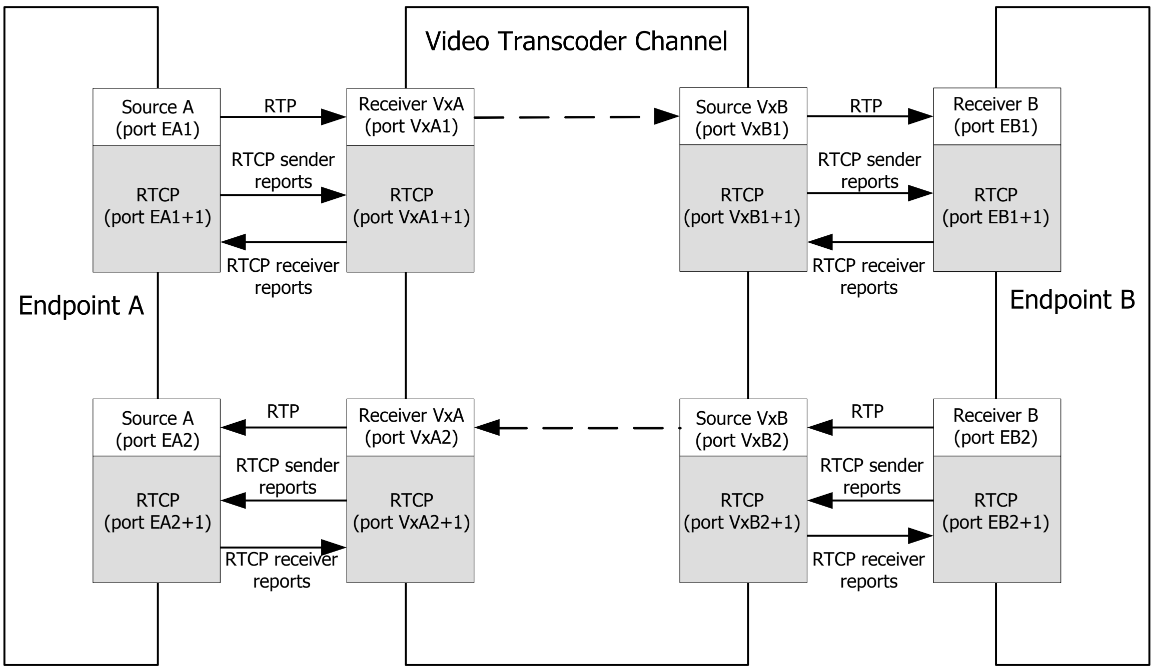
RTP control protocol support

The video transcoder provides support for the RTP control protocol (RTCP). By supporting an RTCP data stream for each RTP stream, the video transcoder provides remote endpoints with additional control information.

The video endpoints receive the RTCP to monitor the quality of service for an RTP session. RTCP allows an endpoint to synchronize the video RTP stream with its associated audio RTP stream even though each stream takes a different path to the endpoint. For the video transcoder, video data takes a path in which the stream flows through the transcoder while the associated audio bypasses the transcoder. By supporting RTCP, a video endpoint can obtain stream synchronization.

RTCP also conveys information about the participants of an RTP session. The video transcoder platform provides RTCP support by acting as a translator (as described by RFC 3550).

The following illustration shows the full set of RTP and RTCP endpoints that are in use for a full-duplex transcoding channel. RTCP sender reports are sent from endpoints that are transmitting RTP data. RTCP receiver reports are sent from endpoints that are receiving RTP data.



Port numbering

When assigning RTP and RTCP port numbers, be sure to assign each RTP port an even number. Assign the associated RTCP port to the RTP port number + 1. RTCP ports always use odd numbers.

Translator implementation

Any point that connects two different RTP data flows must either act as a translator or a mixer:

Function	Description
Translator	An intermediate system that forwards RTP packets with their synchronization source identifier intact. The video transcoder platform is considered a translator because it does not support any connection types that involve receiving multiple input RTP streams and the primary reason for supporting RTCP in the transcoder is to allow the ultimate endpoint to synchronize the video with the audio associated with the originating endpoint.
Mixer	Receives RTP data from multiple sources mixing the streams into a single output stream. Acts as its own synchronization source. The video transcoder platform is never used as a mixer.

Decoupling of each connection leg

The video transcoder platform acts as an RTCP translator, making it not visible as an RTCP endpoint. The video transcoder platform does not have its own synchronization source (SSRC) because it never acts as a synchronization source.

A translator must not simply forward RTCP packets from one leg to another. For example, the video transcoder platform performs transcoding on the video stream, resulting in a potentially different outbound packet count than the received (inbound) packet count. A translator must make transformations in the sender and receiver report information sent in RTCP messages.

The video transcoder will use the receipt of a sender or receiver report over one leg as the trigger to issue the same type of report on the associated leg. The information in the outbound report will not match the inbound report but will instead reflect the packet counts of the associated leg.

Video stream synchronization

Because the video transcoder platform acts as a translator, it can pass along synchronization information in the RTCP messages it transmits. This information allows an endpoint to perform inter-media synchronization of the video stream and the audio stream. The video stream passes through the video transcoder platform translator and the audio stream is delivered without passing through the video transcoder platform. See *RFC 3550* for a detailed description of the format of this synchronization information.

Endpoints that resolve the synchronization information, as well as endpoints that generate the RTP flows to be synchronized, should have their times synchronized to the network time protocol (NTP). Each video transcoder platform must also be synchronized to the NTP in order to relay accurate timing information.

Receiving sender and receiver reports

Each type of RTCP termination point maintained by the video transcoder platform is either acting as an RTP receiver monitor or as an RTP transmitter monitor. When acting as an RTP receiver monitor, the video transcoder platform expects to receive sender reports from the remote endpoint and generate receiver reports back to that endpoint. When acting as an RTP transmitter monitor, the video transcoder platform expects to receive receiver reports from the remote endpoint and will generate sender reports to that endpoint.

Report generation

Any video transcoder platform receiver port will send periodic receiver reports. Any video transcoder platform transmitter port will send periodic sender reports.

The session bandwidth allocated for the sending of RTCP reports is based on the session bandwidth of the given endpoint configuration with RTCP information restricted to no more than 5% of the available bandwidth. The transcoder assumes that 25% of this RTCP bandwidth is dedicated to sender reports and the remainder is dedicated to receiver reports.

It is important that all participants using RTCP use the same value for the session bandwidth so that the same RTCP interval is correctly calculated.

The video transcoder platform will not make any bandwidth restriction calculations to determine report generation intervals since the transcoder will simply issue an RTCP report whenever it receives the given type of report from the other side of the connection.

RFC 3550 specifies that an endpoint that is not actively sending data should issue a receiver report instead of issuing a sender report. This restriction does not apply to the transcoder since the decision to send RTCP is made by the remote endpoint.

BYEs

When an RTCP BYE packet is received on any video transcoder platform RTCP port, a BYE is issued out of the corresponding endpoint and the control application is notified that the BYE has occurred. When an application stops a channel, it is considered a hard channel termination. In this case, BYEs are not sent and they are not expected as receives.

Reception diagnostics

RTCP statistics can be used to diagnose a variety of data reception issues. The Management Interface provides access to all of the information maintained by the channel’s RTCP control layer.

The data provided by the RTCP interface allows an external entity (an endpoint participating in the RTP/RTCP exchange) to determine the number of packets expected as well as the number of packets lost. The fraction of packets lost during the last reporting interval is maintained by each RTCP-aware endpoint.

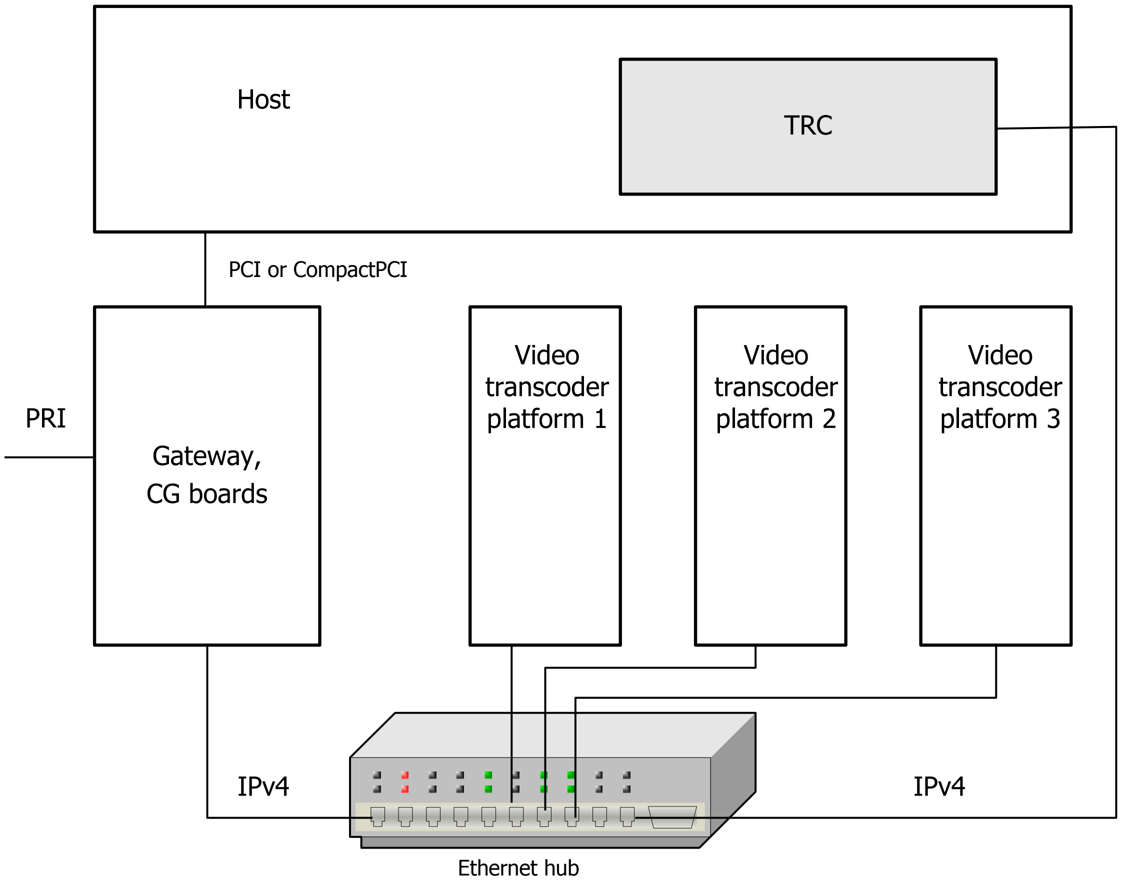
Configurable options

The following table provides a description of the configurable RTCP support options:

Option	Description
Default RTCP usage configurable per VTP	Each video transcoder platform can be configured to support or not support RTCP on each video session. Regardless of whether a VTP is configured to handle RTCP by default, any given channel can override these RTCP defaults.
Participant timeout detection	Each connection can be monitored for inactivity. A session member who has not sent any RTP or RTCP packet in the time period specified is considered timed out. The TRC Control Agent notifies the control application when a timeout occurs but does not perform any other action on the channel. The application can report the timeout or take more serious action such as terminating the channel. By default, a connection is configured to have no inactivity timeout period. In this case, the TRC Control Agent never reports a participant as having timed out.
Do not listen for or send RTCP	Any given transcoder connection can be configured to drop all received RTCP traffic and to issue no outbound RTCP traffic. This mode allows the transcoder to be operated as in currently deployed releases.
Activate tracing of sent/received RTCP traffic	A channel can be made to trace sent and received RTCP messages using the logging and tracing capabilities of the video transcoder platform.

Video gateway implementation example

The following illustration shows the high-level architecture for implementing a multiple-channel video gateway using the TRC and a Dialogic CG board:



Each TRC can supervise and allocate channels in a pool of video transcoder platforms. Multiple host applications can share the associated video transcoder platform resources.

Using the Software Video Transcoder Management Interface (VTMNG)

All video transcoder platforms are managed using the VTMNG API. This API provides a set of requests that can be directed to any specific video transcoder platform. The VTMNG API can be configured to receive responses to requests and either display the returned information or upcall the owning application for further response processing. The VTMNG API also allows the owning management application to register to receive any asynchronous events (traps) that are issued by video transcoder platforms.

By supporting multiple concurrent uses of the VTMNG interface, you can develop an event server to receive all asynchronous indications with a separate operator configuration tool that is used for manual intervention. When a management application initializes the VTMNG API, a UDP listen is posted for the receipt of responses. The VTMNG API will either allow the operating system to select any available UDP port (the default operation) or, the VTMNG API will be provided with the specific UDP port for which to register.

Optionally, the VTMNG API can also register to handle an operator console (keyboard) interface. When the management application completes its own initialization, it calls the VTMNG API polling loop function which handles dispatching of received responses (as well as any asynchronous indications received).

When the calling application issues a request, it calls a VTMNG API function that formats a management message that is sent as a UDP packet. The message is sent to the destination video transcoder platform address indicated by the caller. Management messages are automatically sent to the appropriate video transcoder platform-based process. The *vtmon* process handles all process monitor requests while the trc_agent handles all other management requests. Once the request is processed, a response is generated as a UDP packet sent to the originator of the request. The VTMNG API receives the response and calls the appropriate upcall function to provide control back to the owning application.

The owning application can attach a key when the VTMNG API is initialized (***userkey***). This key is provided on all upcalls. There is also a key that is under the owning application's control on a per-request basis. Set the VTP address field ***sendkey*** to set a send-specific key that is passed with the outgoing request. This key is received as part of the resulting response and provided as ***sendkey*** in the video transcoder platform address record provided on the response upcall.

You can also use the VTMNG API in a raw mode. In this mode, the application is responsible for sending and receiving all management messages as UDP packets. The VTMNG API determines the size of a message and converts the message between network byte-order and local host byte-order.

VTMNG management requests

Management requests are categorized as follows:

Management request	Description
VTMNG_CATEG_VTP	VTP-level control.
VTMNG_CATEG_APP	Application-level control.
VTMNG_CATEG_MON	Monitored Process control.
VTMNG_CATEG_CHN	Channel-level control.
VTMNG_CATEG_ST_TOTAL	Total statistics.
VTMNG_CATEG_ST_CURRMIN	Current minute statistics.
VTMNG_CATEG_HIST_PERMIN	Histogram statistics [per-minute increments].
VTMNG_CATEG_HIST_PERHHR	Histogram statistics [per half-hour increments].

VTMNG management operations

The following management operations can be performed:

Management operation	Description
VTMNG_OP_GET_LIST	Retrieves a list of entity IDs.
VTMNG_OP_GET_ENTITY	Retrieves a copy of single entity record (configuration and status and statistics).
VTMNG_OP_SET_CONFIG	Change an entity's current configuration.
VTMNG_OP_ZERO_STATS	Zeroes all statistics and clears all stored error indications.
VTMNG_OP_NOTIFY	Asynchronous notification.

Not all operations can be performed on all management categories. The following table provides a breakdown of the valid combinations:

Operation	VTP	APP	MON	CHN	TOTAL	CURRMIN	PERMIN	PERHHR
GET_LIST	XXX	[LIST]	[LIST]	[LIST]	XXX	XXX	XXX	XXX
GET_ENTITY	(1)	(1)	(1)	(1)	(1)	(1)	(1)	(1)
SET_CONFIG	(1)	(1)	(1)	XXX	XXX	XXX	XXX	XXX
ZERO_STATS	(1)	(1)	(1)	(1)	(1)	XXX	XXX	XXX

Key:

- XXX not allowed
- (1) valid request; response contains single record
- [LIST] valid request; response contains list of records

The VTMNG API reports all responses and asynchronous notifications through upcalls to owner functions.

It provides information about the message source and the received message. The message source is a video transcoder platform address record and the received message is a management message (VT_MNG_MSG). All VTMNG functions are prototyped in the *vtmng.h* header file. All VTMNG messages are defined in the *transmanage.h* header file.

Operator console tool and sample application

The vtmgr tool provides a ready-to-use operator console application. When no parameters are provided, vtmgr uses the VTMNG API to present an operator command interface that provides control over all transcoder management.

The vtmgr tool can also be executed with options that demonstrate how to use the VTMNG API to develop custom management applications.

Refer to the sample application file *vtmgr.c* for the source code of the *vtmgr* tool. For more information, refer to *vtmgr - Management utility overview* on page 325.

Using the transcoder with Dialogic Video Access

Dialogic Video Access is a toolkit enabling developers to build and deploy carrier grade applications, such as video gateways and enhanced services platforms, for video communication solutions. Video Access components have a modular architecture. Components are used independently or in conjunction with other Video Access components.

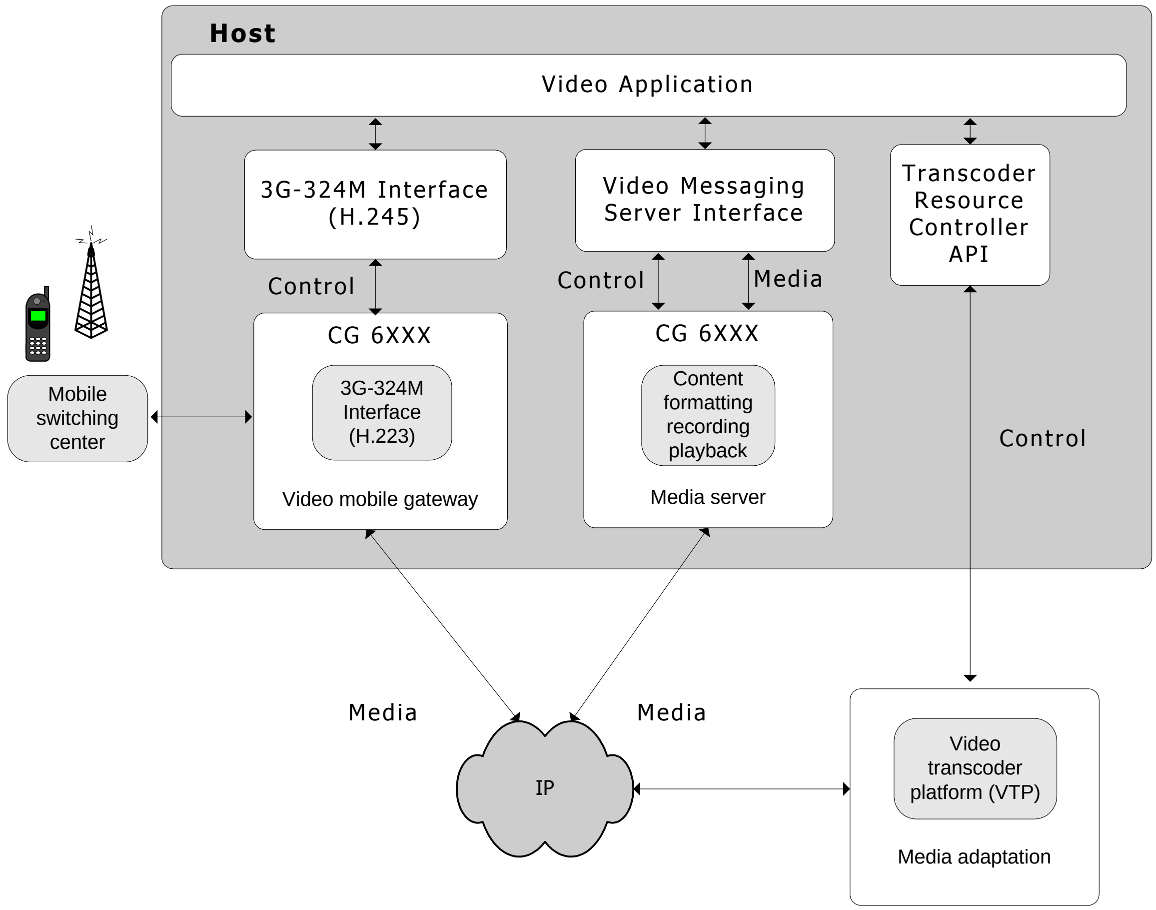
The Dialogic Video Access toolkit contains the following components:

Component	Description
3G-324M Interface	Enables video gateway functions or applications to establish connections with 3G-324M capable wireless terminals using H.245 messaging and H.223 multiplexing using the Natural Access MSPP service. The 3G-324M Interface allows applications to bridge 3G-324M connections to the IP network with optional audio transcoding. For information, refer to the *3G-324M Interface Developer's Reference Manual*.
Video Messaging Server Interface	Controls play, record, and storage in 3GP file format for audio and video data. An application accesses multimedia play and record functions through the Natural Access ADI service. For more information, refer to the *Video Messaging Server Interface Developer's Reference Manual*, the *ADI Service Developer's Reference Manual*, and the *MSPP Service Developer's Reference Manual*.

Dialogic Video Access is an extension to Dialogic Natural Access. The Natural Access development environment provides a thread-safe communication mechanism using Natural Access or Computer Telephony Access (CTA) queues as an event delivery mechanism. For information about purchasing Video Access, contact your Dialogic sales representative. For information about Natural Access, see the *Natural Access Developer's Reference Manual.*

The Video Access application can use the Dialogic Software Video Transcoder to transcode video bit streams into different formats, based on the requirements of the service and the end terminals. For example, in a video messaging service, an application might need to play video content to the user terminal, but the user terminal might not support the video format. In this scenario, the application can use the video transcoder to convert the content so that the terminal can receive and play it.

The following illustration shows how the video transcoder can be used with Video Access components:



Migrating from Software Video Transcoder 2.0

Software Video Transcoder Release 2.0 and Software Video Transcoder Release 2.1 applications can share video transcoder platforms that are running either the Release 2.0 or Release 2.1 Software Video Transcoder server release. However, Software Video Transcoder 2.1 must be installed on both platforms for the Releae 2.1 features to be enabled.

The Software Video Transcoder Release 2.1 TRC API can direct video channels requiring overlay, RTCP support, or both to a video transcoder platform that is running the Releae 2.1 release. The Software Video Transcoder Release 2.0 TRC API does not distinguish between Software Video Transcoder releases.

Because overlay-aware resources can be dedicated to channels that do not require overlays, Dialogic suggests that you do not use Software Video Transcoder 2.0 TRC API instances that share Software Video Transcoder 2.1 video transcoder platforms.

Caution:	Red Hat Linux requirement Software Video Transcoder 2.1 requires Red Hat Linux Enterprise Solutions 4. If you are running Red Hat Linux Enterprise Solutions 3, you must upgrade to Red Hat Linux Enterprise Solutions 4 before you install Video Transcoder 2.1.

Functions

Functions that are no longer used by the Video Transcoder:

- **trcTextVideoChannel**
- **trcImageVideoChannel**

Any applications using these functions must migrate to the new overlay interface.

New functions include:

Function	For more information, refer to...
trcNameVideoChannel	*trcNameVideoChannel* on page 135.
All management interface functions	*Management function summary* on page 109.
All overlay functions	*Overlay functions* on page 108.

Extended functions include:

Function	Extended functionality	For more information, refer to...
trcCreateVideoChannel	The channel type can now optionally indicate overlay and/or RTCP requirements.	*trcCreateVideoChannel* on page 116.
trcStartVideoChannel	The endpoint configuration now includes RTCP control.	*trcStartVideoChannel* on page 145.

New structures

New structures:

- VT_MNG_MSG (management message)
- All management structures
- All overlay control structures

For more information, refer to the *TRC structures overview* on page 219 and to the *Management structures overview* on page 257.

New functionality

New functionality includes:

Functionality	For more information, refer to...
Text and graphic overlay	*Text and graphic overlays* on page 22.
Management interface	*Using the Video Transcoder Management Interface (VTMNG)* on page 28.
RTCP support	*RTP control protocol support* on page 23.

New features

The following table provides a description of some of the new features for Video Transcoder 2.1:

Feature	Description
Overlay interface	Image and text overlay control capabilities.
Management interface	The VTMNG API provides a complete management interface.
RTCP support	Support for the RTCP protocol with the transcoder acting as a translator between RTP data streams.
Call completion records	Records output as asynchronous indications that include traffic statistics.
Transcoder library	Enhanced to support overlays. Reorganized the execution engine in preparation for future performance enhancements.
Rendering engine	Full-featured image and text rendering capabilities performed by a new process dedicated to rendering operations.
vtmgr	Tool that provides a console-based interface for managing video transcoder platforms.

Feature	Description
vtmon	Process that executes when the video transcoder platform starts up. Monitors all other video transcoder processes. Any lost process will trigger vtmon to perform automatic recovery.
License Manager	Upgraded to the latest Dialogic License Manager version.
Video Access 3.0	Upgraded supported client-side OS set to match those supported by Video Access 3.0.

3. Starting the Video Transcoder

Starting the video transcoder platform

The Video Transcoder software installation modifies the startup sequence of the video transcoder platform.

The default startup uses the Video Transcoder Process Monitor (*vtmon*) to start and monitor all other transcoder processes. *vtmon* can detect the failure of any other transcoder process and restart the subset of processes that are required to automatically bring the video transcoder back into full service.

The *vtmon* process is started by the Red Hat Linux-provided *inittab* service. The configuration file */etc/inittab* stores the command that causes *inittab* to start *vtmon* as part of the video transcoder platform's standard boot sequence. If *vtmon* terminates, the Linux operating system (through *inittab*) automatically restarts *vtmon*.

You can use an alternative startup mechanism if you do not need automatic process recovery. With this method, the */etc/init.d/nmsXC* script calls the operator-controlled *startXC.sh* script to start all video transcoder processes.

Use the *monitorXC.sh* script to switch between the two startup methods. Specify *monitorXC.sh status* to view the current startup mode. Valid startup modes are monitor ON (default) or monitor OFF. To deactivate automatic process recovery, enter the following command:

```
monitorXC.sh off
```

To re-activate automatic process recovery, enter the following command:

```
monitorXC.sh on
```

Note: Running the *monitorXC.sh on* command causes the process monitor (*vtmon*) to run immediately. All transcoder port licenses must be installed before performing a manual *monitorXC.sh on*.

The Dialogic License Manager daemon (*nmslm*) is also started as part of the video transcoder platform startup sequence. The license manager is started first so that transcoder licenses are available when the video transcoder processes are started.

Video transcoder platform startup process

The following table lists the video transcoder platform processes that are created during the video transcoder startup sequence. The processes are listed in startup sequence order.

Process	Description
vtmon	Process monitor. Performs automatic recovery of lost processes. *vtmon* can be removed from the startup sequence if you do not need automatic recovery.
trcr	Overlay rendering process.
trc_agent	Acts as a server to the TRC. Assigns transcoder channels to specific trcp files and forwards channel control commands to trcp files.
trcp	Performs all video transcoding for one or two channels. The agent creates an appropriate number of trcp processes to support the configured channel count.
xscontrol	Reboots the video transcoder platform when requested by **trcResetVTP**.

Configuring the video transcoder platform

Each video transcoder platform is configured through a set of configuration files. The configuration is set when the video transcoder platform starts up and cannot be changed without performing a video transcoder platform restart. Other configuration information is modified using the management interface. This type of configuration can be altered dynamically.

Configuration files are installed as: */opt/nms/video/<**base filename**>.example.cfg*

If the installation detects that a configuration file already exists, the example file is not copied over the pre-existing file. If no pre-existing file exists, then the example is copied as the active file.

The following table describes the video transcoder platform configuration files:

File	Description
encodeh263.cfg	Customized H.263 encoder configuration. This configuration file should not require any modifications for standard operations. Edit this configuration file to change the default behavior of the H.263 encoder.
encodempeg.cfg	Customized MPEG4 encoder configuration. This configuration file should not require any modifications for standard operations. Edit this configuration file to change the default behavior of the MPEG4 encoder.
trcr.cfg	Configuration file for the overlay text and image overlay rendering process. Use this file to set global rendering parameters and to define pre-rendered fonts and cached images.

File	Description
usageLevel.cfg	Customized processor usage estimation control. This configuration file should not require any modifications for standard operations. Edit this configuration file to change the default usage cost that is assigned to various types of transcoder channels.
vtmon.cfg	Video Transcoder Process Monitor (vtmon) configuration file. This configuration file should not require any modifications for standard operations. Use the Management Interface (vtmgr or other custom management application) to view and modify the process monitor configuration.
vtp.cfg	Contains video transcoder platform-level information. This file is used by the TRC control agent (trc_agent). Use the Management Interface (vtmgr or other custom management application) to view and modify the video transcoder platform-level configuration.

Configuring text and image overlay rendering

Use the following global parameters to control overall rendering behavior:

Variable	**Description**
debuglogmask	Set of logging bits that can be activated to obtain detailed tracing information of render operations. Default: 0x00000003 (trace any detected WARNING or ERROR condition). This variable should not require any modifications for standard operations.
SharedMem	Set the size of the shared memory area that is used to pass rendered overlay content between the *trcr* process and the transcoder channel processes (*trcp*). Default: 12000000 bytes (between 11 and 12 megabytes).

Specifying cached fonts

Font descriptions can be pre-defined by placing entries in the [fonts] section of *trcr.cfg*. Text rendering is performed more efficiently when pre-rendered fonts are used because the processing overhead takes place when the video transcoder platform is started as opposed to when a given text overlay is requested.

When defining a pre-rendered font, use the font command and specify the following information:

- A short name to identify the font.
- The name of the true type font file describing the font.
- A range of font sizes to be pre-rendered.

The following fonts are listed in the default *trcr.cfg* file:

```
[fonts]
#                                                                   Prerendered font sizes
#    Name          Font File Name                                   From  To    Increment
#    ------------  ------------------------------------------------ ----  ----  ---------
font arial         "./fonts/freefont-20060126/FreeSans.ttf"         12    24    4
font arial_bold    "./fonts/freefont-20060126/FreeSansBold.ttf"     12    24    4
font arial_italic  "./fonts/freefont-20060126/FreeSansOblique.ttf" 12    24    4
font serif         "./fonts/freefont-20060126/FreeSerif.ttf"        12    24    4
font serif_bold    "./fonts/freefont-20060126/FreeSerifBold.ttf"    12    24    4
font serif_italic  "./fonts/freefont-20060126/FreeSerifItalic.ttf" 12    24    4
```

Specifying cached images

Images can also be specified in the *trcr.cfg* file. This defines the set of images that are placed into the image cache when the video transcoder platform is started. The image cache is used to preprocess image files that are used often. Images are resized in advance and shared among all transcoder processes (*trcp*).

When defining a cached render image, use the image command and provide the following information:

- A unique ASCII name that the image will be referred to as
- Image filename
- Width, height, and resizing method
- Formatting alignment
- Background color to use to render the image

The following cached images are defined by the default *trcr.cfg* file:

```
[images]
#     Name              Image File Name                                W     H   Resize
    Alignement  Bg Color
#     ---------------  --------------------------------------------  ----  --- ------     ---
-------  ----------
image NMS_N_32x32      "file://images/NMS_Logo/NMS_N-347x452.png"      32    32  fit
       center      0x00000000
image NMS_N_16x16      "file://images/NMS_Logo/NMS_N-347x452.png"      16    16  horizontal
topleft     0x00000000
image NMS_Logo_100x50  "file://images/NMS_Logo/NMS_Logo-1427x452.png"  100   50  fit
       center      0xffffffff
```

Customizing encoder behavior

Configuration files are provided allowing for customization of either H.263 or MPEG-4 encoding. Normally, there is no need to alter encoding configuration because the defaults are set to provide the most commonly desired behavior.

The following encoder options can be specified in the *encodeh263.cfg* or *encodempeg.cfg* configuration files:

Option	Description
Enc_Drop_Early_Frames	Set flag indicating whether the encoder can drop frames that are being requested before initial information has been output. Valid values: 0 = Do not drop early frames 1 = Drop early frames
Enc_Drop_Low_Qual_Frames	Set flag indicating whether the encoder should drop outbound frames containing low video quality. Valid values: 0 = Do not drop low quality frames 1 = Drop low quality frames
Enc_Time_Resolution	Specify the default encoder time resolution.
Enc_Partitioned	Specify if data partitioning is enabled. Valid values: 0 = Disable partitioning 1 = Enable partitioning
Enc_Frames_Per_I	Specify whether to encode frames as I-frames. Valid values: 0 = First I-frame, then all P-frames 1 = All encoded frames are I-frames

Option	Description
Enc_Num_MB_Refresh	Encoder number for macroblock refresh for intra coding of P-frames. Only valid when Enc_Frames_Per_I is set to 0. The config field sets the target number of macroblocks to encode as part of each outbound P-frame. It then outputs information about changed and unchanged areas of the video frame. This allows for recovery of lost video frame change information at the receiving endpoint. The encoder cycles through the macroblocks that make up the video frame and passes information about the specified macroblocks with each outbound frame.
Enc_Packet_Size	Encoder packet size (MPEG4-only).
Enc_Time_Period	Specifies the time increment of each frame. Valid values: 0 = Use the time increment of each frame, otherwise this value provides a fixed time increment.
Enc_AC_Predict	Specify whether AC prediction is enabled. Valid values: 0 = Disable AC prediction 1 = Enable AC prediction (MPEG4-only)
Enc_Use_Type2_MB	Specify whether to use Type-2 macroblocks. Valid values: 0 = Disable use of Type 2 MB's 1 = Enable (MPEG4-only)
Enc_2nd_Initial_Iframe	Specify whether to duplicate the initial output I-frame (sometimes used as H.263 standard behavior).

Agent configuration options

Configuration is performed through the management interface. To protect configuration files from being accidentally deleted or overwritten, configuration file examples use the following naming conventions:

*<**name**>.example.cfg*

During the initial installation, these example files are automatically copied to names without the *example* in the name. For each example configuration file, **.example.cfg* becomes **.cfg*. The example configuration files are only copied when non-example files are missing. Only the example files are removed on uninstall. This allows for a software upgrade that maintains the previous configuration.

This topic provides the following example configuration files:

- Video transcoder platform configuration file example
- Transcoder overlay rendering configuration file example

Video transcoder platform configuration file example

All VTP-level configuration is stored in the *vtp.cfg* file. When the TRC Control Agent (trc_agent) starts up, it reads *vtp.cfg* to obtain its initial configuration. All configuration is then controlled through the management interface. When any management request to set the configuration is received, the trc_agent automatically re-generates the *vtp.cfg* file to reflect the changes.

The following example configuration file shows the default settings:

```
################################################################################
#
# Video Transcoder Platform (VTP) Control Application Configuration File
#
# This file specifies all VTP-level configuration.
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!! This file is automatically updated due to dynamic configuration:
# !!!!! Any updates to VTP-level configuration that are received during system
# !!!!! operation are automatically stored to this file.
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#-------------------------------------------------------------------------------
# vtpName: name of this VTP instance
#   SPECIAL VALUE: @ = use hostname
#-------------------------------------------------------------------------------
# vtpDesc: VTP description (version/revision, etc.)
#   SPECIAL VALUE: @ = use internal desc line (with trc_agent version/revision)
#-------------------------------------------------------------------------------
# vtpInitState: set whether VTP will init to an enabled or disabled state
#   ENABLED   = VTP is enabled upon startup (allowing channel assignments)
#   DISABLED  = VTP is disabled upon startup (no channel assignment allowed)
#-------------------------------------------------------------------------------
# rtcpMode = set whether VTP acts as an RTCP translator by default
#   DISABLED = do not listen for RTCP messages and do not send RTCP messages
#   ENABLED  = listen for received RTCP and send RTCP as appropriate
#-------------------------------------------------------------------------------
# decodePartial = set whether partial frames should be passed to the decoder
#   DISABLED = do not pass partial frames to the decoder (drop partials)
#   ENABLED  = pass partial frames to decoder
#-------------------------------------------------------------------------------
# licenseHighWater: percentage of licenses in use considered high water mark
#   VALID RANGE: 0-100
#-------------------------------------------------------------------------------
# licenseLowWater: percentage of licenses in use considered low water mark
#   VALID RANGE: 0-100 (must be less than or equal to licenseHighWater
#-------------------------------------------------------------------------------
# usageHighWater: percentage of estimated usage considered high water mark
#   VALID RANGE: 0-100
#-------------------------------------------------------------------------------
# usageLowWater: percentage of estimated usage considered low water mark
#   VALID RANGE: 0-100 (must be less than or equal to usageHighWater
#-------------------------------------------------------------------------------
# spxMaxChans: maximum number of simplex transcoding channels to allow
#    VALID RANGE: 2-<total port licenses> (must be even)
#-------------------------------------------------------------------------------
# trcpCount: number of transcoder processes to create
#    VALID RANGE: positive values = TRCP count [2-spxMaxChans]
#                 negative values = simplex channels per TRCP [1-spxMaxChans]
#-------------------------------------------------------------------------------
# mediaAddress: IP address used for media endpoint access to VTP
#   SPECIAL VALUE: @ = use same IP address for control and media
#-------------------------------------------------------------------------------
# overlayExclusive = whether VTP is reserved for channels requiring overlays
#   DISABLED = VTP can be assigned chans with or without overlay requirements
#   ENABLED  = VTP cna only be assigned channels requiring overlay capability
#-------------------------------------------------------------------------------
# maxRtpPayload: maximum size of any outbound RTP packet payload
#   VALID RANGE: 32-1460
#-------------------------------------------------------------------------------
# apiTimeout: TRC API watchdog timeout (time allowed for TRC API response)
#   UNITS: milliseconds
#   SPECIAL VALUE: 0 = no TRC API watchdog timeout (wait infinitely)
#-------------------------------------------------------------------------------
# initTimeout = time (after connect) to wait for INIT REQ from TRC API
#   UNITS: milliseconds
#   SPECIAL VALUE: 0 = no INIT REQ watchdog timeout (wait infinitely)
#-------------------------------------------------------------------------------
# appLostTimeout = time (after disconnect) before considering app lost
#   UNITS: milliseconds
```

```
#   SPECIAL VALUE: 0 = no app connection lost timeout (wait infinitely)
#-------------------------------------------------------------------------------
# debugLogMask: global trc_agent debug log mask (set of VSLOG_xxx bits)
#-------------------------------------------------------------------------------
# trcpLogMask: global trcp debug log mask (set of VSLOG_xxx bits)
#-------------------------------------------------------------------------------
# logToConsole = whether to-file logging should be forked to console
#   DISABLED = do not fork to-file log entries to console
#   ENABLED  = be verbose: fork to-file log entries to console
#-------------------------------------------------------------------------------
# rtcpInTimeout: default RTCP idle (no RTP or RTCP RX) input endpoint timeout
#   UNITS: milliseconds
#   SPECIAL VALUE: 0 = no RTCP input endpoint idle timeout (wait infinitely)
#-------------------------------------------------------------------------------
# rtcpOutTimeout: default RTCP idle (no RTCP RX) output endpoint timeout
#   UNITS: milliseconds
#   SPECIAL VALUE: 0 = no RTCP output endpoint idle timeout (wait infinitely)
#-------------------------------------------------------------------------------
# trapMask: mask of events VTP will issue traps for (VTMNG_EVENT_xxx)
#-------------------------------------------------------------------------------
# trapAddress: IP address that all async indications (traps) are issued to
#   SPECIAL VALUE: @ = do not issue any traps
################################################################################

#===============================================================================
# VTP top-level configuration:
vtpName           = @
vtpDesc           = @
vtpInitState      = ENABLED
rtcpMode          = DISABLED
decodePartials    = DISABLED
licenseHighWater  = 80
licenseLowWater   = 60
usageHighWater    = 80
usageLowWater     = 60
spxMaxChans       = 60
trcpCount         = -2
mediaAddress      = @
overlayExclusive  = DISABLED
maxRtpPayload     = 1342
apiTimeout        = 5000
initTimeout       = 5000
appLostTimeout    = 300000
debugLogMask      = 0x00000003
trcpLogMask       = 0x00000003
logToConsole      = DISABLED
rtcpInTimeout     = 0
rtcpOutTimeout    = 0
trapMask          = 0x00100007
trapAddress       = @
#===============================================================================
```

For detailed descriptions of the configurable elements, refer to the Management functions section.

Transcoder overlay rendering configuration file example

The transcoder overlay rendering configuration file includes two sections that define the fonts and images that must be cached by the rendering process:

Section	Description
[fonts]	Indicates the fonts that the render process (*trcr)* will pre-process. Outline font files do not include a bitmap image of each character. They include a description of each character that explains how to draw the character, allowing the characters to be drawn in almost any resolution. This requires more processing to render each character. To minimize processing, the rendering process can cache the pre-rendered bitmap representation of characters. For example, you can decide which font, in which sizes will be required and ask the rendering process to create the associated bitmaps at startup.
[images]	Used to pre-process and pre-load image files to shared memory. For example, if a company uses a logo or icons repeatedly, they can be pre-loaded and shared among all transcoder processes (*trcp*). Otherwise, a new copy of each image must be loaded for each overlay. Also, if a menu has a graphic bullet on each row and each bullet is a separate overlay, the image must be loaded for each overlay. Instead, if the image is cached through the [images] section, the same image can be referenced in many overlays.

The following example shows the configuration file for the overlay rendering process:

```
#############################
# GLOBAL DEFINITION
#############################
[global]
debuglogmask=0x00000003
SharedMem=12000000
#############################################################
# Font cache definitions
#
# The font cache is used to prerender fonts into bitmap fonts.
# A name is associated to a specific font file and a range of
# sizes.
#
# Available font files under fonts/freefont-20060126/
#
# FreeMonoBoldOblique.ttf
# FreeMonoBold.ttf
# FreeMonoOblique.ttf
# FreeMono.ttf
# FreeSansBoldOblique.ttf
# FreeSansBold.ttf
# FreeSansOblique.ttf
# FreeSans.ttf
# FreeSerifBoldItalic.ttf
# FreeSerifBold.ttf
# FreeSerifItalic.ttf
# FreeSerif.ttf
#
#############################################################
[fonts]
#                                                                         Prerendered font
sizes
#    Name           Font File Name                                        From    To
```

```
     Increment
#    ------------  ----------------------------------------------------  ------  ------
 ---------
font arial         "./fonts/freefont-20060126/FreeSans.ttf"              12      24
     4
font arial_bold    "./fonts/freefont-20060126/FreeSansBold.ttf"          12      24
     4
font arial_italic  "./fonts/freefont-20060126/FreeSansOblique.ttf"       12      24
     4
font serif         "./fonts/freefont-20060126/FreeSerif.ttf"             12      24
     4
font serif_bold    "./fonts/freefont-20060126/FreeSerifBold.ttf"         12      24
     4
font serif_italic  "./fonts/freefont-20060126/FreeSerifItalic.ttf"       12      24
     4
#############################################################
#
# Image cache definition.
#
# The image cache is used to preprocess image files that
# are used often. Images are resized in advanced and shared
# among all application.
#
#############################################################
[images]
#     Name             Image File Name                                 W     H   Resize
    Alignement  Bg Color
#     ---------------  --------------------------------------------  ----  --- ------
    ----------  ----------
image NMS_N_32x32      "file://images/NMS_Logo/NMS_N-347x452.png"      32    32  fit
       center      0x00000000
image NMS_N_16x16      "file://images/NMS_Logo/NMS_N-347x452.png"      16    16
 horizontal topleft     0x00000000
image NMS_Logo_100x50  "file://images/NMS_Logo/NMS_Logo-1427x452.png"  100   50  fit
       center      0xffffffff
#end of cfg file
Process Monitoring configuration file example
```

All configuration related to process monitoring is stored in the *vtmon.cfg* file. When the video transcoder process monitor (*vtmon*) starts up, it reads *vtmon.cfg* to obtain its initial configuration. All configuration is then controlled through the management interface. When any management request to set the configuration is received, *vtmon* automatically regenerates the *vtmon.cfg* file to reflect the changes.

The following example configuration file shows the default settings:

```
################################################################################
#
# Video Transcoder Process Monitor Configuration File
#
# This file specifies all process monitoring configuration.
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# !!!!! This file is automatically updated due to dynamic configuration:
# !!!!! Any updates to monitor configuration that are received during system
# !!!!! operation are automatically stored to this file.
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
#
#-------------------------------------------------------------------------------
# initState <processName> <state>
#   where <state>:
#     CREATE    = create process and then monitor the process
#     LOCATE    = locate the process (not created by video transcoder monitor)
#     SKIP      = skip the process (not created and not monitored)
#-------------------------------------------------------------------------------
# lostAct <processName> <action>
#   where <action>:
#     NONE      = take no action on process lost (just report lost process)
#     PROCRUN   = re-run the process
#     DISABLE   = trigger VTP disable... once disabled, perform a restart
#     RESTART   = perform an immediate restart: stop all, then start all
#     TRCPCLEAN = stop all VT proc's, terminate all trcp's, then start all
#-------------------------------------------------------------------------------
# execFile <processName> <filename>
#   where <filename> = fully qualified filename of process executable
#-------------------------------------------------------------------------------
# cmdLine <processName> <commandLine>
#   where <commandLine> = options string used as command line on proc start
#-------------------------------------------------------------------------------
# varDef <processName> <varNum> <varName> <varString>
#   where <varNum>    = variable number
#         <varName>   = name of variable
#         <varString> = variable definition string
################################################################################

#===============================================================================
# VTMON configuration:
initState vtmon = skip
lostAct   vtmon = none
execFile  vtmon = /opt/nms/video/vtmon
cmdLine   vtmon =
varDef    vtmon 1  logDirs = 5
varDef    vtmon 2  MAGICK_HOME = /opt/nms/video/lib/ImageMagick-6.2.8
varDef    vtmon 3  MAGICK_CONFIGURE_PATH = /opt/nms/video/lib/ImageMagick-6.2.8/config
varDef    vtmon 4  MAGICK_FILTER_MODULE_PATH = /opt/nms/video/lib/ImageMagick-
6.2.8/modules-Q16
varDef    vtmon 5  MAGICK_CODER_MODULE_PATH = /opt/nms/video/lib/ImageMagick-
6.2.8/modules-Q16/coders
varDef    vtmon 6  LD_LIBRARY_PATH = /opt/nms/lib:/opt/nms/video/lib

initState trc_agent = create
lostAct   trc_agent = trcpclean
execFile  trc_agent = /opt/nms/video/trc_agent
cmdLine   trc_agent =

initState trcr = create
lostAct   trcr = trcpclean
execFile  trcr = /opt/nms/video/trcr
cmdLine   trcr = -c /opt/nms/video/trcr.cfg

initState xscontrol = create
lostAct   xscontrol = procrun
execFile  xscontrol = /opt/nms/video/xscontrol
cmdLine   xscontrol =

#===============================================================================
```

For detailed descriptions of the configurable elements, refer to the Management functions section.

Configuring channel processor usage estimates

By default, the video transcoder platform estimates channel processor usage based on a channel's data rate. The video transcoder platform uses the following default reference points and usage costs to calculate usage estimates:

Frame resolution and operation	Data rate reference points	Frame rate reference points	Usage cost
QCIF encode	64 kbit/s	15 fps	63
QCIF decode	64 kbit/s	15 fps	63
CIF encode	384 kbit/s	25 fps	408
CIF decode	384 kbit/s	25 fps	408

To change the way that the video transcoder platform calculates usage estimates, you must provide a configuration file to the trc_agent. You can do either or both of the following in the configuration file:

- Base the usage calculations on the frame rate instead of or in addition to the data rate.
- Specify other reference points for the calculation.

When the video transcoder platform estimates a channel’s processor usage, it first checks whether a point with the specified data rate and frame rate is in the configuration file. If a match is found, the specified usage cost is used for the estimate. If a match is not found, the video transcoder platform performs the estimation calculation.

The following table describes the configuration file commands:

Command	Description
cif	Defines reference points for usage estimates with a CIF frame resolution using the following syntax: cif *action dataRate frameRate usageCost* where: *action*: Operation for which to calculate the usage estimate. Valid values are encode or decode. *dataRate*: Data rate to use as a reference point in the usage estimate, in kbit/s. *frameRate*: Frame rate to use as a reference point in the usage estimate, in fps. *usageCost*: Usage cost for the specified rates. This value is expressed as [percent CPU utilization *.100]. For example, a value of 63 means 0.63% of the processor is needed.
highWater	Sets the high water mark, which is the processor usage level at which

Command	Description
	the video transcoder platform considers itself 100% saturated using the following syntax: highWater *usageLeve*l where *usageLevel* is the process usage level at which the video transcoder platform considers itself 100% saturated. This value is expressed as [percent CPU utilization *.100]. For example, to consider 50% processor usage as the high water mark, specify 5000.
method	Determines which variables to use for calculating usage cost estimates using the following syntax: method ***options*** where ***options*** is one of the following: dataRate: Use the ***dataRate*** variable only. frameRate: Use the ***frameRate*** variable only. sums: ***dataRateModifier frameRateModifier***. Use both the ***dataRate*** and ***frameRate*** variables. The system multiplies each rate by its given modifier value and then sums the two rates. When method sums is used, the configuration line looks like the following example: method sums <***dataRateModifier***> <***frameRateModifier***> The usage estimate in this case uses the following equation: usageCost = <***dataRateModifier***> * <***endpointDataRate***> + <***frameRateModifer***> * <***endpointFrameRate***>
qcif	Defines reference points for usage estimates with a QCIF frame resolution using the following syntax: qcif ***action dataRate frameRate usageCost*** where: ***action***: Operation for which to calculate the usage estimate. Valid values are encode or decode. ***dataRate***: Data rate to use as a reference point in the usage estimate, in kbit/s. ***frameRate***: Frame rate to use as a reference point in the usage estimate, in fps. ***usageCost***: Usage cost for the specified reference point. This value is expressed as [percent CPU utilization *.100]. For example, a value of 63 means 0.63% of the processor is needed.

Usage level configuration file example

The following example configuration file shows a simplified CPU usage assignment scheme used for estimating CPU usage:

```
#------------------------------------------------------------------------------
# Estimated Processor Usage Information
# (used for VTP resource assignment decisions made by the TRC API)
#------------------------------------------------------------------------------
#
# Defines CPU usage costs for both encode and decode operations
# (at each supported frame resolution [qcif and cif]).
# Also sets the high water mark (usage level) at which VTP should be
# considered fully saturated.
# NOTE: The unit for usage values is "percent * 100" so 5000 = 50% utilization.
#
# A refined estimate of CPU cost can be obtained for a given platform by
# running a variety of different configuration test cases and monitoring
# the used CPU (using top, for example).
# Different values can be entered for encoding and decoding operations
# if the test data reveals an imbalance in the handling of operations.
#
# The following example shows a simplified CPU usage assignment scheme
# in which encoding and decoding is given the same weight
# (allowing a quick configuration using only a single load test for each
# frame resolution in use [same format in and out]).
#
# EXAMPLE:
#
#   QCIF test point: 64 kbps @ 15 fps
#        A channel load of 37 simplex channels used 48.24% of the CPU
#        ( (48.24%*100)/37 channels ) / 2 = 63 [per operation charge]
#
#   CIF  test point: 384 kbps @ 25 fps
#        A channel load of 6 simplex channels used 48.98% of the CPU
#        ( (48.98%*100)/6 channels ) / 2 = 408 [per operation charge]
#
#   qcif decode  64 15  63
#   qcif encode  64 15  63
#
#   cif  decode 384 25 408
#   cif  encode 384 25 408
#
# NOTE: One qcif encode & decode and one cif encode & decode is the minimum
#       configuration. Provide additional sample points for any category
#       where a specific usage cost is desired (skipping estimation).
#
# SYNTAX: qcif|cif decode|encode <dataRate> <frameRate> <usageCost>
#------------------------------------------------------------------------------

# QCIF      Data Frame Usage
#           Rate Rate  Cost
#========== ==== ===== =====
qcif decode   32     7    32
qcif encode   32     7    32

qcif decode   48    11    46
qcif encode   48    11    46

qcif decode   64    15    63
qcif encode   64    15    63

# CIF       Data Frame Usage
#           Rate Rate  Cost
#========== ==== ===== =====
cif  decode   96     7   124
cif  encode   96     7   124

cif  decode  192    15   224
cif  encode  192    15   224
```

```
cif  decode  384    25   408
cif  encode  384    25   408

#-------------------------------------------------------------------------------
# Set method used for CPU estimation (<estimateMethod>):
#   dataRate  - linear extrapolation of dataRate
#   frameRate - linear extrapolation of frameRate
#   sums      - compute usage using sum of:
#                 (dataRate * dataRateModifier) + (frameRate * frameRateModifier)
#
# SYNTAX: method dataRate
#         method frameRate
#         method sums <dataRateModifier> <frameRateModifier>
#-------------------------------------------------------------------------------
method dataRate

#-------------------------------------------------------------------------------
# consider VTP at max capacity when usage reaches 5000 (50% CPU [estimation])
# SYNTAX: highWater <usageLevel>
#-------------------------------------------------------------------------------
highWater 5000
```

Using the vtmgr for configuring

Use the management interface tool (*vtmgr*) to view and modify video transcoder platform-level configuration information. The *vtmgr* tool is included with the TRC package.

Use the vtp command to view all current video transcoder platform-level configuration data. Each video transcoder platform-level configurable parameter can be modified independently. Certain management information is also stored on a per-process basis. This information is used by the process monitor when it starts a given transcoder process. Use the mon command to view the configuration of a given monitored process.

Transcoder processes must be running for the management interface to function. In addition, the management interface can only be used to control process monitoring if the transcoder is operating in monitor ON mode. When operating in monitor OFF mode, any *vtmgr* requests to the process monitor will not be responded to.

Verifying connectivity

After all video transcoder platforms are started, verify that the client system can connect to all of the video transcoder platforms. To do this, create a configuration file defining the IP address for each video transcoder platform. For more information, refer to *trcInitialize* on page 129.

Once the configuration file is created, use the *trccheck* utility to verify communication with all listed video transcoder platforms as shown in the following example:

```
[1] create text file describing 2 VTPs (called trcExample.cfg):
    vtp 10.10.0.1
    vtp 10.10.0.2
[2] trccheck –c trcExample.cfg
```

trccheck attempts to establish a control connection with each of the video transcoder platforms. The TRC API is used to connect to all of the video transcoder platforms listed in the configuration file. *trccheck* performs the following tasks:

Task	Description
1	Initializes the TRC API, causing the TRC API to begin establishing connections to all video transcoder platforms listed in the configuration file.
2	Enters a polling loop, waiting for the TRC API to report that connection is established with all listed video transcoder platforms.
3	Displays a connectivity summary report and terminates.

4. Developing host applications

Overview of developing host applications

You develop host applications by using TRC functions to allocate and release video transcoder resources. This topic describes how to use TRC functions, and it also presents guidelines for developing host applications.

Using TRC functions

The TRC communicates with the video transcoders through TCP/IP sockets and sends commands and receives acknowledgements, if necessary. The TRC is also responsible for communications recovery if the TCP/IP fails (for example, if the video transcoder is reset), and for notifying the application about transcoder resource availability. All video transcoders have a command processor that is responsible for TCP/IP communication with the TRC.

During TRC initialization, the application provides a pointer to a callback function that the TRC uses to signal the system changes, for example, channel start and stop or communications failure. The TRC programming model is event based, where the events are asynchronous and the callbacks are associated with the generated events.

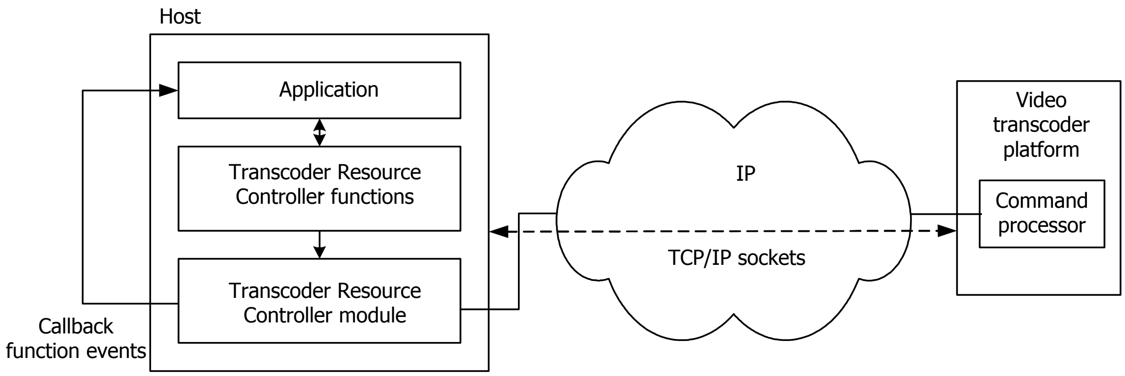
Callbacks occur from a thread separate from the application thread. The callback function must use a thread-safe method to present these events to the main application threads. The following table provides a description of the recommended methods for doing this:

Method	Description
Use CTA queues	The callback function enqueues the corresponding message within the CTA queue for the channel and postpones event processing until the application services that queue.
Use a pipe between the callback thread and the main application thread.	The received event is written out one side of the pipe during callback execution. The callback function returns control to the TRC before the target thread processes the event. The target thread reads from the other side of the pipe to process the event information in a thread-safe manner.

For more information about callback functions, refer to *trcInitialize* on page 129.

TRC function overview

The following illustration shows an overview of the TRC functions:

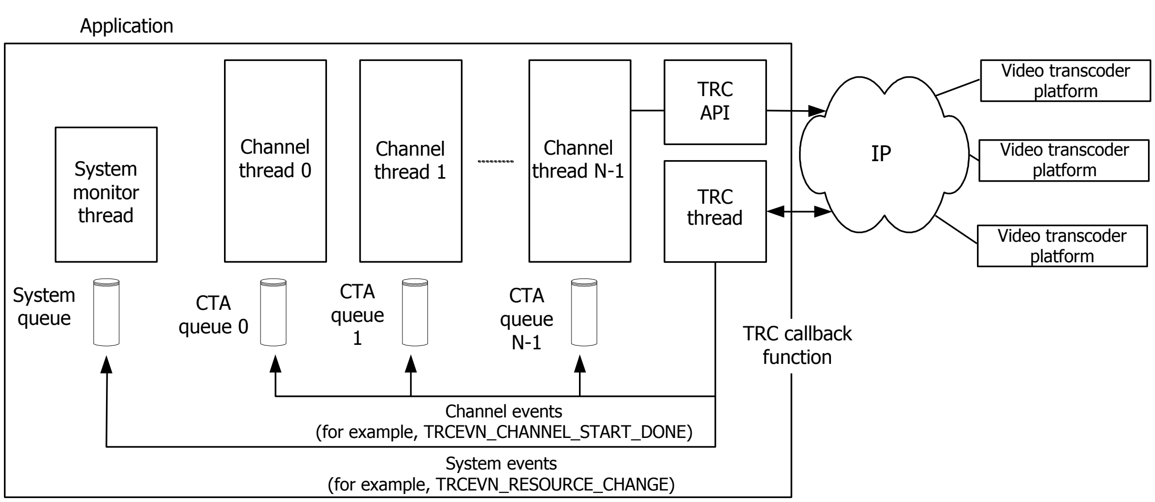


Guidelines for developing host applications

When developing host applications using the TRC, follow these guidelines:

- Design a TRC application so that a channel control context is maintained for each channel the application creates.
- Use a single threaded model or develop a model in which a separate thread controls each channel.
- Monitor control messages (acknowledgements and asynchronous indications) coming from the video transcoders.
- Use a thread-safe mechanism to present event indications to the main application thread or to a given channel control thread, as described in *Using TRC functions* on page 55.

The following illustration shows a TRC application programming model that uses CTA queues to manage asynchronous callback events:



For information on events that can be returned from the TRC, refer to *Transcoder resource controller events* on page 312.

Application tasks

The host application works with the TRC to control the transcoding process. This involves starting the transcoding process, performing tasks while transcoding, stopping the transcoding process, and handling exceptions, such as channel failures or video transcoder platform failures.

You typically perform the following tasks to start the transcoding process:

Task	Action	For more information, refer to...
Start the transcoding process	Initialize the TRC.	*Initializing the TRC* on page 58
	Create a video channel.	*Creating a video channel* on page 58
	Start the transcoding process on the specified channel, specifying the configuration of each endpoint that is being connected by the channel.	*Starting the transcoding process* on page 59
	Retrieve information about the newly started channel.	*Monitoring video channels* on page 61
Tasks to perform while transcoding	Monitor video transcoder platform connections while transcoding.	*Monitoring transcoder resources* on page 61
	Monitor video channels while transcoding.	*Monitoring video channels* on page 61
	Optionally, request that the transcoder modify the video stream output while transcoding on the channel. You can request that the transcoder: Display text messages as overlays on the output video stream. Display static images as overlays on the output video stream. Insert an I-frame into the output video stream.	*Creating an overlay* on page 70
Stop the transcoding process	Stop the transcoding process on the video channel.	*Stopping the transcoding process* on page 62
	Destroy the video channel.	*Destroying a video channel* on page 62
	Shut down the TRC.	*Shutting down the TRC* on page 63

Handling exceptions	If a channel failure occurs, destroy the channel. If the TRC is not able to connect to a video transcoder platform, reset the video transcoder platform.	*Responding to channel failures* on page 63 *Resetting a video transcoder platform* on page 63

Initializing the TRC

Before you can use TRC functions, you must first initialize the TRC by invoking **trcInitialize**. This function directs the TRC to establish connections with all video transcoder platforms listed in the configuration file. You can establish connections with up to five video transcoder platforms.

Note: If any transcoding channels are left running from a previous usage of the TRC using the same application name provided to **trcInitialize**, the TRC places those channels in an idle state.

A successful return from **trcInitialize** indicates that the TRC was successfully configured and is bringing up all video transcoder platform connections. The TRC sends the application a TRCEVN_RESOURCE_CHANGE event each time a video transcoder platform connection is established.

trcInitialize does not block the application. A successful return from **trcInitialize** indicates that the initialization process started with connection being established to each video transcoder platform. Channels cannot be created until at least one video transcoder platform connections is completely established.

After the TRC is initialized, it monitors the status of each video transcoder platform. If there is a loss of connectivity to a video transcoder platform, the TRC sends a TRCEVN_RESOURCE_CHANGE event to the application. The event indicates that the number of available transcoder resources was reduced, until the video transcoder platform connection is automatically re-established. Once a connection with a video transcoder platform fails, the TRC periodically attempts to reconnect to the video transcoder platform. If the connection is re-established, the TRC sends a TRCEVN_RESOURCE_CHANGE event to the application, indicating an increased number of available resources.

Creating a video channel

Invoke **trcCreateVideoChannel** to create a video channel. Once a channel is created, you cannot change the options without destroying the channel. This function returns a TRC channel handle that manages the channel. It also assigns RTP ports and dedicates port licenses, as follows:

- For a simplex channel, this function assigns one RTP receiving (Rx) port. It dedicates a single port license to the channel.
- For a full-duplex channel, this function assigns two RTP Rx ports to provide endpoints for the receive traffic over both sides of the connection. It dedicates two port licenses to the channel.

When you create a channel, you must specify if the channel requires overlay support and RTCP support. By specifying these transcoding features when the channel is created, the TRC can select a video transcoder platform that is able to provide these services. Once a channel is created, you cannot change the options without destroying the channel. So, even if a channel only requires overlays in certain circumstances, it must still be created with the overlay option, even if the option ends up not being used.

When the TRC receives acknowledgement from the selected video transcoder platform, it issues the TRCEVN_CREATE_CHANNEL_DONE event. If the event indicates success, invoke **trcInfoVideoChannel**. This function provides the assigned IP address of the selected video transcoder platform and the RTP receiving port numbers associated with the channel.

You can create video channels any time the number of reported available licenses is greater than or equal to the number of licenses needed for the requested channel type. The TRC is aware of the maximum number of video channels that each video transcoder supports, as well as the current number of channels in use.

The TRC creates the next video channel by randomly selecting from the set of least used video transcoder platforms. Each video transcoder platform determines its current usage level by taking into account the types of channels that are currently defined. The specific usage cost assigned to a given channel depends on the channel's configuration. The method used to determine utilization can be based on the frame resolution, data rate, and frame rate in use for each channel direction. For more information, refer to *Configuring channel processor usage estimates* on page 49.

If video transcoder platform resources are not available, the **trcCreateVideoChannel** request fails immediately with a return code of TRCERR_NORESOURCES.

It is possible for a create request to be accepted by the TRC but failed by the video transcoder platform. This can occur due to a timing condition in which a different TRC instance has obtained the channel before the first application's request could be received by the target video transcoder platform. In this case, the TRCEVN_CREATE_CHANNEL_DONE event indicates the failure with error code TRCERR_NORESOURCES. It is the application's responsibility to retry failed create attempts.

Starting the transcoding process

Invoke **trcStartVideoChannel** to start the transcoding process. This starts a simplex or full-duplex video transcoding process on the specific transcoder on which the channel was created. Invoke trcStartVideoChannel upon receiving a video call and completing the media session that identifies all port and video information.

The TRC returns the status of the **trcStartVideoChannel** call through the TRCEVN_START_CHANNEL_DONE event. If the video channel cannot start, the status provides a reason code.

For more information, refer to *trcStartVideoChannel* on page 145 and *Transcoder channel control* on page 20.

The following type of information is passed to the TRC when a transcoding channel starts:

- General endpoint information

  Specifies the encoding characteristics that the TRC expects on the input bit stream and that it generates on the output bit stream. These encoding characteristics include:
  - Encoding type (H.263 or MPEG-4)

- Profile
- Level
- Data rate
- Frame rate (QCIF or CIF)
- Packetization mode

- Channel input information

  Specifies if RTCP flow is expected from the input endpoint with the transcoder receiving sender reports.

  Note: Legacy TRC API versions provided control over an optional input jitter buffer. This option introduced a delay between receiving input video data and issuing output video data. The use of the jitter buffer provided a way to process inbound RTP data streams in which packets could arrive out of order. Reordering at the RTP packet level is now performed for all channel types, without the need for any fixed delay period in the outbound video information.

  To support legacy applications, the jitterMode and jitterLatency fields are still defined as part of an input endpoint's configuration, but any request for a static jitter buffer is treated as if no jitter buffer were requested.

- Channel output information

  Specifies the destination IP address and port number of the endpoint, as well as the RTP payload ID and type of service the transcoder uses in all outbound packets. If outputting to an MPEG-4 endpoint, you can also specify the timestamp resolution.

  Also specifies whether to expect RTCP flow to the remote endpoint with the transcoder issuing sender reports to the output endpoint.

  Channel output fields also include a variety of optional output configuration values for the encoder.

- Decoder and encoder configuration information

  Optionally specify to send decoder configuration information for each direction in which transcoding is performed. Uses the options data record. For example, DCI for MPEG-4.

If you do not know a configuration value for an endpoint, set the value to TRC_CONFIG_DEFAULT indicating that a default value should be used.

Monitoring transcoder resources

To monitor transcoder resources as seen through the control interface, you can:

- Monitor the status of all video transcoder platform connections
- Determine the current channel usage across all video transcoder platforms

The following table describes how to use these methods to monitor transcoder resources:

Invoke this function...	To...
trcVTPStatus	Monitor the status of all video transcoder platform connections. This function provides the current state and usage status of each video transcoder platform, including a unique video transcoder platform ID assigned by the TRC. Invoke this function any time after **trcInitialize** is called.
trcUsage	Determine the current channel usage across all video transcoder platforms that are being controlled through the TRC. This function returns the: Total number of licenses available. Number of channels in use by this application. Number of channels in use by all applications that share the video transcoder platform resources.

Monitoring video channels

To monitor video channels as seen through the control interface, you can:

- Retrieve complete information about a specific video channel
- Monitor the status of all video channels that your application controls

Retrieving complete information about a specific video channel

Retrieve complete information about a specific video channel by invoking **trcInfoVideoChannel**. This information includes the current channel state, the video transcoder platform addressing information, and the configuration information for the video channel.

Invoke **trcInfoVideoChannel** after you successfully create a channel and receive a successful TRCEVN_CREATE_CHANNEL_DONE event from the TRC.

trcInfoVideoChannel provides the video transcoder platform addressing information that is assigned to the channel.

Monitoring the status of all video channels

Monitor the status of all video channels that your application controls by invoking **trcChannelStatus**. You can invoke this function any time after you have initialized the TRC.

Channel summary information is provided for each channel that the application has created and not yet destroyed. The information includes the current state of the channel and the ID of the video transcoder platform to which the channel is assigned. A single application instance can use up to 512 channels.

If needed, use **trcResetVTP** to reboot a video transcoder platform. This causes all channels assigned to the given video transcoder platform to fail. These channels include those in use by the application calling **trcResetVTP**, as well as all channels in use by other applications that share the same video transcoder platform.

Note: Resetting a video transcoder platform can adversely affect other applications.

Stopping the transcoding process

Use **trcStopVideoChannel** to stop the transcoding process. This function stops all video transcoding for the given video channel. You can stop transcoding on a channel at any time.

The TRC returns the status of the **trcStopVideoChannel** call through the TRCEVN_STOP_CHANNEL_DONE event. If the video channel cannot stop, the status provides a reason code.

Destroying a video channel

Use **trcDestroyVideoChannel** to destroy a video channel. This removes the information for the specified channel from the TRC database. It also stops the channel, if it is still active.

Once you invoke **trcDestroyVideoChannel**, the handle that was returned earlier by **trcCreateVideoChannel** becomes invalid. The TRC performs all actions needed to cleanly terminate the channel, but no further event indications are issued to the application.

Due to the asynchronous nature of thread execution, it is possible that events issued before the call to **trcDestroyVideoChannel** may have already been queued by the application's event notification function. To avoid this issue, always call **trcStopVideoChannel** and wait for the TRCEVN_STOP_CHANNEL_DONE event, before calling **trcDestroyVideoChannel**.

Note: If the connection is lost between the TRC and the video transcoder platform, the TRC cannot complete a stop operation. This keeps the channel in a stopping state. This is the one case where you may not want to wait for the stop to complete, before you invoke **trcDestroyVideoChannel**.

Shutting down the TRC

When you want to stop all transcoding channels and terminate the TRC instance, shut down the TRC by invoking **trcShutdown**. This function directs the TRC to begin breaking all TCP/IP connections with the available video transcoders. The TRC destroys all active transcoder channels (channels for which **trcDestroyVideoChannel** was not previously called), before it disconnects the TCP/IP sessions.

Once you call **trcShutdown**, you must wait to receive the asynchronous callback event TRCEVN_SHUTDOWN_DONE as the indication that the TRC has completely shutdown all video transcoder connections. After shutdown occurs, you must re-initialize the TRC through **trcInitialize** to use any resources controlled by the TRC.

Responding to channel failures

If a video channel encounters problems, the TRC sends one of the following asynchronous events through the callback function:

- TRCEVN_CHANNEL_LOST
- TRCEVN_CHANNEL_FAILED

The following table provides a description of the channel failure types:

Type of channel failure	Description
Lost channel	Indicates a connection problem and is considered a recoverable failure. For a recoverable failure, the TRC automatically attempts to re-establish the channel. Wait until the TRC sends a TRCEVN_CHANNEL_RECOVERED event, indicating that the channel is re-established.
Failed channel	If a critical channel failure occurs, the TRCEVN_CHANNEL_FAILED event is returned and you must terminate the channel by calling **trcDestroyVideoChannel**. If you receive a TRCEVN_CHANNEL_LOST event (recoverable channel failure) and later receive a TRCEVN_CHANNEL_FAILED event (critical failure), the TRC was attempting to re-establish control of a lost channel when it determined that the channel could not be recovered.

Resetting a video transcoder platform

If a TRC is not able to connect to a video transcoder platform, invoke **trcResetVTP** to force the video transcoder platform to reboot. You can invoke this function any time after the TRC is initialized.

Note: This reset mechanism is provided only for legacy support. Use the Management API to reset the video transcoder platform. For more information, refer to *Video Transcoder Management Interface (VTMNG)* on page 17.

The following table provides a description of what happens when trcResetVTP is invoked:

Stage	Description
1	The TRC attempts to re-connect to the video transcoder platform once the TRC receives the indication that the TCP/IP connection to the video transcoder platform is lost.
2	The TRC issues a TRCEVN_RESOURCE_CHANGE event to indicate the loss of the transcoder resources that were provided by the resetting video transcoder platform.
3	When the video transcoder platform completes the reset, the connection between the TRC and the video transcoder platform is automatically re-established.
4	The TRC issues another TRCEVN_RESOURCE_CHANGE event to indicate the recovered transcoder resources.

Multiple applications can share the transcoder resources being provided by the video transcoder platform. If any application invokes **trcResetVTP**, all applications using this video transcoder platform will have their channels fail as the video transcoder platform reboots. This effect must be taken into consideration before deciding that a reset of the video transcoder platform is required.

The reset request is sent to a separate process on the video transcoder platform that is dedicated to providing the remote reset capability. In cases where the physical connection to the video transcoder platform has failed, **trcResetVTP** cannot trigger the reset.

Modifying the output video stream

After you start transcoding on a channel, you can request that the transcoder modify the output video stream for that channel while it transcodes.

You can request that the video transcoder insert an I-frame into the output video stream by invoking **trcIframeVideoChannel**. Invoke this function any time after the video channel is started. The TRC returns the status of the **trcIframeVideoChannel** call through the TRCEVN_IFRAME_CHANNEL_DONE event. If the I-frame cannot be generated, the status provides a reason code.

You can also modify the output video stream to:

- Display overlays with text content
- Display overlays with graphic content

For more information, refer to *Text and graphic overlays* on page 22.

5. Creating text and graphic overlays

Working with overlays

The Video Transcoder supports two basic types of overlays:

- Text overlays are created by rendering text strings into bitmap images.
- Image overlays are created from converting and resizing standard image files (gif, png, or jpeg).

A single channel direction defines up to 32 overlays of either type of overlay. A full duplex channel can have up to 64 overlays, but no more that 32 overlays in each direction.

Once created, an overlay can be in a started or stopped state. The stopped state allows the overlay to be temporarily hidden. Stopped overlays are still counted as defined overlays.

Any dynamic content associated with an overlay remains allocated until the overlay is destroyed.

Text overlays

Use text overlays to create menus, display messages, or display advertisements. Depending on the length of the information to display, it may be necessary to automatically scroll the text content or to use an application intervention (user-selected) to scroll through the text.

The following text overlay modes are available:

- Multi-line text overlay
- Single-line text overlay

Multi-line text overlay

Multi-line text overlays use fixed-length lines to display text. The length of the line is defined by the width of the overlay area. Text that cannot fit on the current line is wrapped to the next line. The number of available lines depends on the overlay area height. Depending on the content options, once the overlay area is filled with text, additional text can either be clipped or the content of the overlay can be scrolled to allow the new text to display.

The following example shows a multi-line text overlay:

Multi-line overlay scrolling

Multi-line text overlays usually scroll vertically. The following scrolling modes are available:

Scrolling mode	Description
No scrolling	Text displays on all available lines until the overlay area is filled. Additional text does not display.
Content based scrolling	Text scrolls in until the last line is shown, at which point the scrolling stops. The portion of text the fits the overlay area remains visible until the overlay is stopped or destroyed. The scroll rate is provided by the application in milliseconds per pixel. For example, a scroll rate of 100 ms/pixel causes the overlay to scroll at 10 pixels per second. Content-based scrolling cannot be looped.
Continuous scrolling	Text is scrolled in until the last line is reached and the begins again from the first line. The content loops continuously until the overlay is stopped or destroyed. The scroll rate is provided by the application in milliseconds per pixel. For example, a scroll rate of 100 ms/pixel causes the overlay to scroll at 10 pixels per second.

Multi-line overlay text wrapping

Text content to display in a multi-line overlay may contain end-of-line information to indicate where text must start on a new line. In the absence of appropriate end-of-line information, text wrapping is performed, forcing text to continue to a new line when the current line is full. The following text wrapping options are available:

Text wrapping option	Description
No text wrapping	Only end-of-line information will cause the creation of a new line. Any text extending beyond the available space is clipped.
Word wrap	The text line is filled until the next word cannot fit. A new line is created where the text continues with the next word. If the word cannot fit on one line, it is clipped.

Single-line text overlay

Single-line text overlays use a virtually infinite line to display text. The overlay area acts as a window through which the text scrolls horizontally. Any end-of-line information in the text content is ignored. Text wrapping does not apply to single line text overlays.

The following example uses multiple single-line text overlays to create a menu:

Single-line overlay scrolling

Single-line text overlays are usually scrolled right to left. The scrolling rate is specified in milliseconds/pixel. For example, a scroll rate of 100 ms/pixel causes the overlay to scroll by at a rate of 10 pixels each second. This form of scrolling can be used to create a marquee style news headline area at the bottom of the video frame.

Overlay formatting

The following table provides a description of the formatting options available for text overlays:

Option	Description
Text attributes	Set the following attributes to control the rendering of text overlays: Font (either cached or dynamic) Size Decoration options (underline, outline) Alignment (right, left, or centered) Foreground/background color Background transparency These attributes apply to all text in an overlay.
Text data	Supplied by the application through the TRC interface.
Font files	Font files are local to each video transcoder platform. The following fonts are available by default: Sans serif (similar to Arial) Serif (similar to Times) Mono-spaced (similar to Courier) To install additional fonts on a video transcoder platform, add the font descriptions to the rendering process configuration file (*trcr.cfg*). These fonts are available in the following styles: Normal Italic Bold Bold italic Underline Bold underline Bold italic underline

Graphic overlays

Overlays with graphic content provide the ability to display static images on top of a video stream in real time. Use image overlays to create icons, logos, or backgrounds.

Graphic modes

The following table provides a description of the overlay graphic modes:

Graphic mode	Description
Static image	Displays a static image. Can be used for icons in menus or logos.
Horizontally/vertically scrolled image	Displays an image by scrolling it horizontally or vertically at a user defined rate (pixels/seconds) in the overlay area. Scrolling can be looped so the content continuously scrolls through the overlay area. Can be used to create moving menu backgrounds.

Graphic data

Image files that contain graphic data to use as overlay content can be accessed through the video transcoder's file system or through an HTTP server.

The TRC interface does not transfer actual images, instead, it uses references to the image files. The application is responsible for ensuring that the required image files are available to the video transcoder. The video transcoder supports:

- JPEG, GIF, and PNG graphic file formats.
- Images that contain transparent areas (GIF or PNG) or semi-transparent areas (PNG only) are supported. For semi-transparency, all transparency information must be provided by an appropriate alpha channel in the PNG file.

The application can indicate whether an image larger than the overlay area is clipped or reduced to fit the area.

Colors and transparency

The TRC interface uses a consistent representation for color based on its red, blue, and green components with the addition of a transparency component. The transparency component indicates the level of transparency or translucence of the colored item.

For information about creating customized colors, refer to *Customizing colors* on page 78.

Creating an overlay

An overlay is defined by two main structures: the overlay area and the overlay content. The overlay area describes the type, size, and position of the overlay. The overlay content describes the media to display in the area.

Overlay area

An overlay area is defined by the following parameters:

Parameter	Description
Overlay type	Defines the overlay as either: Single-line text overlay Multi-line text overlay Graphic overlay
Position	Position given by the horizontal and vertical coordinate of the top left corner of the rectangular area. The values can be expressed either in pixels or as a percentage of the video area.
Size	Size expressed as a horizontal and vertical size. The values can be expressed in pixels as a percentage of the overlay area size or derived from the actual content size.
Initial state	An overlay can be created in either a started or stopped state. By default, overlays display as soon as the content is rendered. Creating an overlay in the stopped state allows the application to synchronize the display of multiple overlays by waiting for an indication that the content of all overlays has been rendered before starting the overlays.
Background color	Defines a background color for the overlay area. Default color is none, which means the area is invisible unless some content is displayed in it. Using a background color other than none makes the overlay area visible as a rectangle of the specified color even if it has no content.
Foreground color	Defines the default color used for the overlay borders and text. The colors can be overridden by the border and font definition.
Border	Defines the color and width (in pixels) of the overlay area border. The border is drawn inside of the overlay area. Adding a border reduces the actual useable overlay area.
Font	Only used for single and multi-line text overlays. The font definition provides the name of the cached or dynamic font and related options, such as the size, color, and decoration (underline or outline).

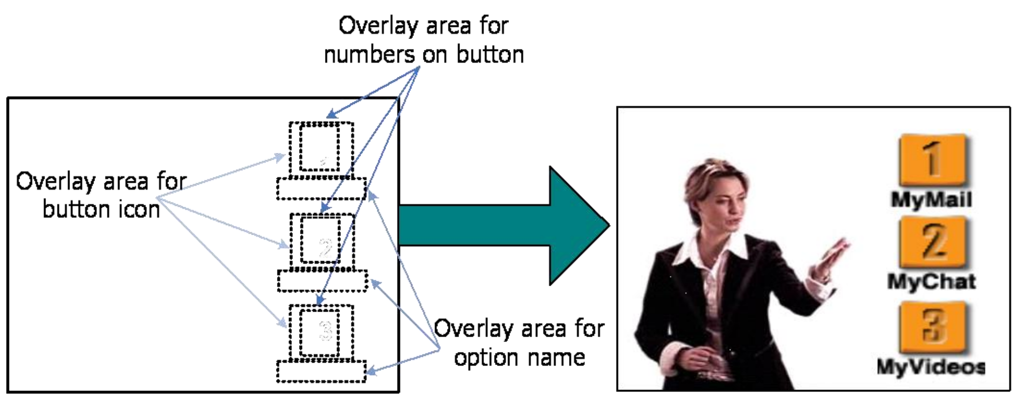
Parameter	Description
Layer	Defines the order in which the overlay areas should be placed above the others. The overlay area with the highest layer value is displayed on top. The video stream is always at the lowest layer and constitutes the background.

Specifying position and size as a percentage value relative to the frame resolution allows these parameters to be independent from the actual video stream’s frame resolution (CIF/QCIF).

Furthermore, the size of an overlay can be derived from the size of its content. Setting the horizontal, vertical, or both size components to the TRC_OVL_COORDINATE_UNIT_UNUSED unit type allows the overlay area to assume the associated size. If the content is too large to fit the video frame given in the defined overlay position, content will be clipped to the video frame limit.

Border and font definition both include a color parameter that can override the default overlay colors. Setting these values to TRC_OVL_COLOR_DEFAULT allow the border or font to use the default overlay colors.

The following illustration shows an example of the overlay areas needed to create a video menu sequence. In this example, overlay areas containing text are used on top of an image to create buttons dynamically. The buttons and text are overlaid on top of the video of the hostess.



Text content

The content definition of an overlay can specify either text or graphic content. Text content is defined by the following parameters:

Parameter	Description
Content type	Identifies the content definition as text.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Scrolling information</td><td>Describes how the content of the overlay should scroll inside the overlay area. The available scrolling modes are:
<table>
<tr><th>Scrolling mode</th><th>Description</th></tr>
<tr><td>Continuous scrolling</td><td>Content scrolls into the video frame until it reaches the end of the content. It then starts over at the beginning of the content, creating a continuous loop.

For example, you can use continuous scrolling to create a marquee style text line that continuously scrolls horizontally at the bottom of the video frame.</td></tr>
<tr><td>Content scrolling</td><td>Content scrolls into the video frame until it reaches the end of the content and then stops scrolling. Content that fits the overlay area continues to display. An event is sent to the application to indicate the end of scrolling was reached.

Use content scrolling when successive messages need to display. For example, when using a multi-line text overlay to display a series of news articles, vertical content scrolling allows each article to scroll in to the last line. At the last line, an event is sent to indicate the end of scrolling. The application can then destroy the overlay and create a new one with the next news article.</td></tr>
<tr><td>No scrolling</td><td>Default. Content appears in the video frame without scrolling.</td></tr>
</table>
</td></tr>
<tr><td>Options</td><td>Use the following options with text content:

Word wrap: Mostly for multi-line text. If set, it will fill lines until a word does not fit in the remaining space on the line and start a new line. If not set, it assumes that any required line feed is imbedded in the text and clips any text extending beyond the overlay area.

Alignment: Text can be either aligned to the left, right, or centered.</td></tr>
</table>

Parameter	Description
Size	By default, the content size is the same as the overlay area's size. When scrolling content, you can specify a size different than the overlay area size. This allows the size component associated with the direction of the scroll to be larger than the overlay area so that content that is larger than the overlay area can be scrolled through. The size of the overlay content can also be derived from the actual size of the rendered content by using the TRC_OVL_COORDINATE_UNIT_UNUSED unit type. You can also have only one dimension derived from the rendered content size while setting the remaining dimension to a predefined value.
Overlay data	Text string to be rendered. Carriage return and line feed characters both cause text to continue on the next line. The tab character is replaced by 8 spaces. Any other control characters or invalid character is replaced by a period (.).

Graphic content

Graphic content is defined by the following parameters:

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Content type</td><td>Identifies the content definition as graphic.</td></tr>
<tr><td>Scrolling information</td><td>Describes how the content of the overlay should be scrolled inside the overlay area. The available scrolling modes are:
<table>
<tr><th>Scrolling mode</th><th>Description</th></tr>
<tr><td>No scrolling</td><td>Default. Content appears in the video frame without scrolling.</td></tr>
<tr><td>Continuous scrolling</td><td>Content scrolls into the video frame until it reaches the end of the content. It then starts over at the beginning of the content, creating a continuous loop.
For example, you can use continuous scrolling to continuously scroll an abstract background image behind a translucent menu.</td></tr>
<tr><td>Content scrolling</td><td>Content scrolls into the video frame until it reaches the end of the content and then stops scrolling. Content that fits the overlay area continues to display. An event is sent to the application to indicate the end of scrolling was reached.
Use content scrolling to create a slide show where each photo is scrolled in. Photos can be resized to fit the overlay, so that once they are fully visible, the scrolling stops. When the application receives the event indicating that scrolling has stopped, it can allow the image to display for a certain amount of time and then destroy the overlay and create an overlay for the next photo in the slide show.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>Options</td><td>Specify how the source image file should be resized to fit the overlay area. The following options are available:
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Fit horizontally</td><td>Image is resized proportionally so its horizontal size matches the horizontal size of the overlay area. This may cause vertical clipping.</td></tr>
<tr><td>Fit vertically</td><td>Image is resized proportionally so its vertical size matches the vertical size of the overlay area. This may cause horizontal clipping.</td></tr>
<tr><td>Fit</td><td>Image is resized proportionally so it fits the overlay area. No clipping of the image can occur but some part of the overlay may show the background color if not transparent.</td></tr>
</table>
When scrolling, the content size may be different from the overlay size. In this case, the resizing is based on the requested content size and not the actual overlay area size.</td></tr>
<tr><td>Size</td><td>By default, the content size is the same as the overlay area’s size.

When scrolling content, you can specify a size different than the overlay area size. If required, you can specify the size component associated with the direction of the scroll to be larger than the overlay area so that content that is larger than the overlay area can be scrolled through. The other size component should match its overlay area counterpart.

The size of the overlay content can also be derived from the actual size of the rendered content by using the TRC_OVL_COORDINATE_UNIT_UNUSED unit type. You can also have only one dimension derived from the rendered content size while setting the remaining dimension to a predefined value.</td></tr>
<tr><td>Overlay data</td><td>A URL representing the file to use as content. The URL prefix can be either:

http:// - Retrieve an image from a web server.

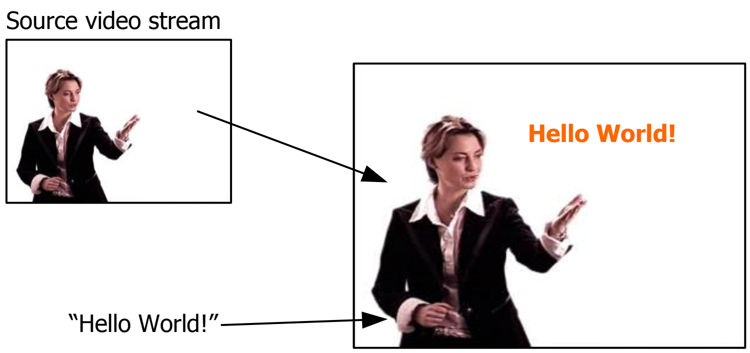
file:// - Retrieve an image from the video transcoder platform's file system or through NFS.

local:// - For a cached image define in trcr.cfg.

If no prefix is specified, file:// is assumed.</td></tr>
</table>

Creating and controlling a text overlay on a simplex channel

This topic describes the steps required to create, display, and destroy a simple text overlay on a simplex channel. In this case, the overlay is created in the started state, so it will display as soon as the content is successfully rendered.



The tTrcOvlConfig structure defines the overlay area. For this example, the overlay type is TRC_OVL_TYPE_SINGLELINE_TEXT.

The position and size of the overlay can be set either explicitly by specifying a number of pixels or relatively as a percentage of the video frame size or the content size. Use the tTrcOvlCoordinates structure to set the position and size of the overlay. The tTrcOvlCoordinates structure contains two coordinates (x and y) and associated units (xUnit and yUnit) to indicate what each coordinate represents. The relevant unit values for overlay position are:

Value of xUnit or yUnit	Meaning of associated x or y value for position
TRC_OVL_COORDINATE_UNIT_PIXEL	An absolute value in pixels.
TRC_OVL_COORDINATE_UNIT_PERCENT	A relative value that represents a percentage of the video frame's equivalent dimension.

The following table shows how the position of the pixel in the middle of the video frame is expressed depending on the resolution (QCIF/CIF) and unit type (absolute/relative):

Resolution	Unit type position	x	y	Unit value settings
QCIF	absolute	88	72	xUnit: TRC_OVL_COORDINATE_UNIT_PIXEL yUnit: TRC_OVL_COORDINATE_UNIT_PIXEL
CIF	absolute	176	144	xUnit: TRC_OVL_COORDINATE_UNIT_PIXEL yUnit: TRC_OVL_COORDINATE_UNIT_PIXEL

Resolution	Unit type position	x	y	Unit value settings
QCIF or CIF	relative	50	50	xUnit: TRC_OVL_COORDINATE_UNIT_PERCENT yUnit: TRC_OVL_COORDINATE_UNIT_PERCENT

In addition, overlay size can use the TRC_OVL_COORDINATE_UNIT_UNUSED unit type to indicate that the coordinate value is not provided and the rendered content size should be used. This can either be the size of the image to display in the overlay or the resulting rendered string message. When the overlay size is derived from the rendered content size, it is the application's responsibility to ensure that the content size is coherent with the video frame size. No resizing of the content is performed. For text content, if no vertical size is provided, the height of the overlay area is assumed to be one line, based on the font point size. The horizontal size will be the length in pixels of the resulting rendered text string.

The tTrcOvlConfig structure also includes a font fields (of type tTrcOvlFont) that describe the font used in the overlay. All characters within an overlay use the same font, in the same size, with the same options.

The font name is provided as a text string that can either refer to a predefined cached font from the *trcr.cfg* [fonts] section or to a dynamic font. To use a cached font, specify the font name defined in *trcr.cfg.* For example, the default *trcr.cfg* file contains the following cached font definitions:

```
[fonts]
#                                                                Prerendered font sizes
#    Name           Font File Name                               From   To    Increment
#    ------------   -------------------------------------------- ----   ----  ---------
font arial          "./fonts/freefont-20060126/FreeSans.ttf"        12     24    4
font arial_bold     "./fonts/freefont-20060126/FreeSansBold.ttf"    12     24    4
font arial_italic   "./fonts/freefont-20060126/FreeSansOblique.ttf" 12     24    4
font serif          "./fonts/freefont-20060126/FreeSerif.ttf"       12     24    4
font serif_bold     "./fonts/freefont-20060126/FreeSerifBold.ttf"   12     24    4
font serif_italic   "./fonts/freefont-20060126/FreeSerifItalic.ttf" 12     24    4
```

To use the *FreeSans.ttf* font file in size 16, Arial is used as the font name and the size field of the tTrcOvlFont structure is set to 16. Only predefined sizes can be used with cached fonts. So, in this example, 18 would not be a valid choice for Arial .

Alternatively, it is possible to use dynamic rendering by prefixing the provided font name with *font://* followed by the required path and font file name. For example, the following paths refer to the *FreeSans.ttf file* installed on the video transcoder platform:

- *font:///opt/nms/video/ fonts/freefont-20060126/FreeSans.ttf*
- *font://fonts/freefont-20060126/FreeSans.ttf*

In this case, the font does not have to be defined in *trcr.cfg*. Any size supported by the font file in this example can be used with dynamic rendering. Dynamic rendering is more flexible but imposes more processing overhead at runtime.

Predefined color values

A specific font foreground and background color can be specified or set to TRC_OVL_COLOR_DEFAULT so the overlay default colors are used. The following color values are predefined:

Color macro name	Value
TRC_OVL_COLOR_RED	0xFF0000FF
TRC_OVL_COLOR_BLUE	0x0000FFFF
TRC_OVL_COLOR_GREEN	0x00FF00FF
TRC_OVL_COLOR_YELLOW	0xFFFF00FF
TRC_OVL_COLOR_MAGENTA	0xFF00FFFF
TRC_OVL_COLOR_CYAN	0x00FFFFFF
TRC_OVL_COLOR_WHITE	0xFFFFFFFF
TRC_OVL_COLOR_BLACK	0x000000FF
TRC_OVL_COLOR_TRANSPARENT	0xFFFFFF00
TRC_OVL_COLOR_DEFAULT	0x00000000 (use default color, this does not appear as transparent)

To explicitly specify the red, blue, and green color value and transparency level as a 32 bit unsigned value use the following settings:

0x***RRGGBBTT*** where:

- ***RR*** is the 2 digit hexadecimal value for red
- ***GG*** is the 2 digit hexadecimal value for green
- BB is the 2 digit hexadecimal value for blue
- ***TT*** is the 2 digit hexadecimal value for transparency (where 0xFF represents fully opaque and 0x00 represents fully transparent)

Customizing colors

Customized colors can be created using the following macros (for this table, it is assumed that RGBA is a variable of type tTrcOvlColor and R, G, B, and A are unsigned byte values between 0 and 255):

Macro name	Use
TRC_OVL_SET_RED(RGBA, R)	Sets the red component.
TRC_OVL_SET_GREEN(RGBA, G)	Sets the green component.
TRC_OVL_SET_BLUE(RGBA, B)	Sets the blue component.

Macro name	Use
TRC_OVL_SET_ALPHA(RGBA, A)	Sets the alpha component (transparency).
TRC_OVL_SET_RGB(RGBA, R, G, B, A)	Sets all components at once in variable RGBA.

The color component can also be retrieved from a tTrcOvlColor variable using the following macros:

Macro name	Use
TRC_OVL_GET_RED(RGBA)	Returns the red component.
TRC_OVL_GET_GREEN(RGBA)	Returns the green component.
TRC_OVL_GET_BLUE(RGBA)	Returns the blue component.
TRC_OVL_GET_ALPHA(RGBA)	Returns the alpha component (transparency).

For example, the following macro could be used to set orange colored text:

```
trcOvlColor textForeGound;
TRC_OVL_SET_RGB(textForeGround, 255, 240, 0, 255);
```

The 255 value for the alpha component represents total opacity. An alpha value of 0 represents total transparence, while value in between represents increasing opacity.

For more information, refer to *tTrcOvlColor* on page 242.

Customizing the font style

The font style allows the text to be underlined or outlined. The outline style is useful when text is displayed over live video. It ensures that the text is always readable regardless of the colors in the background video:

Font style macro	Description
TRC_OVL_FONTSTYLE_NORMAL	Text is rendered without any decoration.
TRC_OVL_FONTSTYLE_UNDERLINED	Text is rendered underlined.
TRC_OVL_FONTSTYLE_OUTLINED	Text is rendered with a one pixel outline. The outline is either black or white depending on which contrasts more with the font foreground color.

For more information, refer to *tTrcOvlConfig* on page 242.

Procedure

Complete the following procedure to create and control a text overlay on a simplex channel:

Step	Action
1	Create a video transcoding channel that specifies the TRC_CH_SIMPLEX and TRC_CH_OVERLAY option in the type argument of **trcCreateVideoChannel**. For more information, refer to *Creating a video channel* on page 58.
2	Start the channel using trcStartVideoChannel. For more information, refer to *trcStartVideoChannel* on page 145.

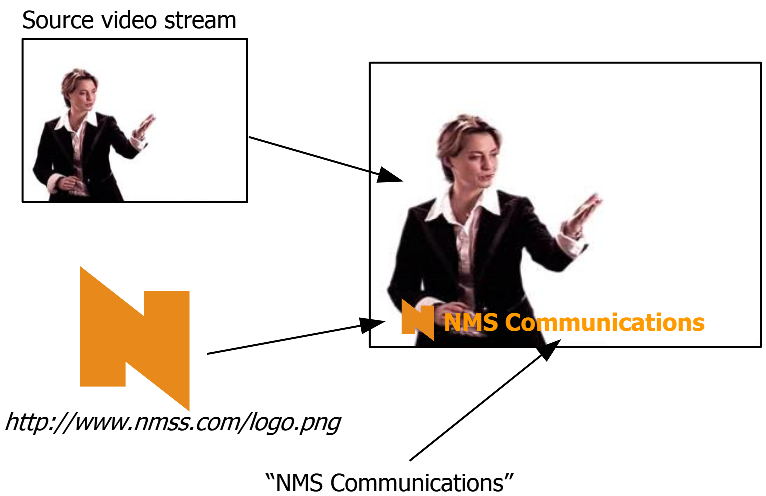
<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3</td><td>Set up the definition of the overlay area using the tTrcOvlConfig structure by specifying:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_TRC_OVL_TYPE_SINGLELINE_TEXT</td></tr>
<tr><td>layer</td><td>1</td></tr>
<tr><td>initState</td><td>TRC_OVL_INITSTATE_STARTED</td></tr>
<tr><td>size.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>size.x</td><td>40 (for example,40%)</td></tr>
<tr><td>size.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>size.y</td><td>15 (for example, 15%)</td></tr>
<tr><td>position.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>position.x</td><td>60 (for example, 60%)</td></tr>
<tr><td>position.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>position.y</td><td>30 (for example, 30%)</td></tr>
<tr><td>fgColor</td><td>TRC_OVL_COLOR_WHITE (overridden by font color)</td></tr>
<tr><td>bgColor</td><td>TRC_OVL_COLOR_TRANSPARENT</td></tr>
<tr><td>border.width</td><td>0</td></tr>
<tr><td>font.name</td><td>Arial</td></tr>
<tr><td>font.size</td><td>20</td></tr>
<tr><td>font.style</td><td>TRC_OVL_FONTSTYLE_NORMAL</td></tr>
<tr><td>font.fgColor</td><td>TRC_OVL_SET_RGB (textForeGround, 255, 240, 0, 255)</td></tr>
<tr><td>font.bgColor</td><td>TRC_OVL_COLOR_DEFAULT (use overlay background color)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>Set up the definition of the overlay content using the tTrcOvlContent structure by specifying:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_CONT_TYPE_TEXT</td></tr>
<tr><td>scroll.type</td><td>TRC_OVL_SCROLL_TYPE_NONE</td></tr>
<tr><td>options</td><td>TRC_OVL_TEXT_CONT_ALIGN_LEFT</td></tr>
<tr><td>size.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>size.x</td><td>100 (100% of the overlay’s horizontal size)</td></tr>
<tr><td>size.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>size.y</td><td>100 (100% of the overlay’s vertical size)</td></tr>
<tr><td>ovlData</td><td>Points to the string “Hello World!”.</td></tr>
</table></td></tr>
<tr><td>5</td><td>Invoke trcCreateOverlay by specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: TRC_DIR_SIMPLEX.
tTrcOvlConfig structure.
tTrcOvlContent structure.
Overlay user data to include with the application in all callbacks related to this overlay.</td></tr>
<tr><td>6</td><td>Wait to receive a TRCEVN_CREATE_OVL_DONE event.
When this event is received, it only indicates that the video transcoder platform has accepted the request. It does not indicate that the overlay is being displayed.
This event carries the overlay handle that uniquely identifies this new overlay along with the user data provided by the application on the call to trcCreateOverlay. The handle will be required on all subsequent call related to this overlay.</td></tr>
<tr><td>7</td><td>The reception of TRCEVN_CHANNEL_OVL_EVENT with result set to TRC_OVLEVT_TRCR_RENDER_SUCCESS indicates that the content has been rendered (either the text string was converted into a bitmap representation or the image file was resized).
Because the overlay is being created in the started state, this also means that the overlay is being displayed. For such an overlay, this event is mostly informational.</td></tr>
</table>

Step	Action
8	To temporarily hide the overlay, invoke **trcStopOverlay** specifying the: Channel handle obtained from Step 1. Overlay handle obtained from Step 6.
9	Wait to receive a TRCEVN_STOP_OVL_DONE event. This event indicates that the overlay is no longer being displayed, but the overlay content is still available.
10	To make the overlay appear again, invoke **trcStartOverlay** specifying the: Channel handle obtained from Step 1. Overlay handle obtained from Step 6.
11	Wait to receive a TRCEVN_START_OVL_DONE event. This event indicates that the overlay is being displayed. The trcStartOverlay function call does not cause the content to be rendered again, so no TRC_OVLEVT_TRCR_RENDER_SUCCESS should be expected.
12	If the overlay is no longer needed, invoke **trcDestroyOverlay** specifying the: Channel handle obtained from Step 1. Overlay handle obtained from Step 6.
13	Wait to receive a TRCEVN_DESTROY_OVL_DONE event. This event indicates that the overlay was destroyed and the associated content was released from shared memory (except for the cached images which are never released).

Creating and controlling multiple overlays

This topic provides an example that describes how to create separate overlays that display synchronously. Because rendering different types of content requires varying amounts of time, it may be necessary to wait until content from all overlays is rendered before displaying them. This ensures that all overlays display synchronously instead of having them appear in sequence, as content becomes available.



The following example shows how to overlay an image and a text string over both directions of a full duplex transcoding channel:

Step	Action
1	Create a video transcoding channel that specifies the TRC_CH_FDX and TRC_CH_OVERLAY option in the type argument **trcCreateVideoChannel**. For more information, refer to *Creating a video channel* on page 58.
2	Start the channel using trcStartVideoChannel. For more information, refer to *trcStartVideoChannel* on page 145.
3	Set up the overlay area definition for the logo using the tTrcOvlConfig structure: Field: type — Setting: TRC_OVL_TYPE_IMAGE Field: initState — Setting: TRC_OVL_INITSTATE_STOPPED

Field	Setting
type	TRC_OVL_TYPE_IMAGE
initState	TRC_OVL_INITSTATE_STOPPED

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>Set up the definition of the overlay content for the logo using the tTrcOvlContent structure:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_CONT_TYPE_GRAPHIC</td></tr>
<tr><td>scroll.type</td><td>TRC_OVL_SCROLL_TYPE_NONE</td></tr>
<tr><td>options</td><td>TRC_OVL_GRAPH_CONT_FIT (to make sure logo is not clipped and is resized to fit within the overlay area)</td></tr>
<tr><td>ovlData</td><td>http://www.nmss.com/Logo.png</td></tr>
</table></td></tr>
<tr><td>5</td><td>Invoke trcCreateOverlay for first direction (FDX1) specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: TRC_DIR_FDX1.
tTrcOvlConfig structure for the logo from Step 3.
tTrcOvlContent structure for the logo from Step 4.
Overlay user data to identify the logo overlay for TRC_DIR_FDX1.</td></tr>
<tr><td>6</td><td>Invoke trcCreateOverlay for second direction(FDX2) specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: TRC_DIR_FDX2.
tTrcOvlConfig structure for the logo from Step 3.
tTrcOvlContent structure for the logo from Step 4.
Overlay user data to identify the logo overlay for TRC_DIR_FDX2.</td></tr>
<tr><td>7</td><td>Set up the definition of the overlay area for the text message using the tTrcOvlConfig structure:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_TYPE_SINGLELINE_TEXT</td></tr>
<tr><td>initState</td><td>TRC_OVL_INITSTATE_STOPPED</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>8</td><td>Set up the definition of the overlay content for the text message using the tTrcOvlContent structure:
<table><tr><th>Field</th><th>Setting</th></tr><tr><td>type</td><td>TRC_OVL_CONT_TYPE_TEXT</td></tr><tr><td>scroll.type</td><td>TRC_OVL_SCROLL_TYPE_NONE</td></tr><tr><td>options</td><td>TRC_OVL_TEXT_CONT_ALIGN_LEFT</td></tr><tr><td>ovlData</td><td>"NMS Communications"</td></tr></table></td></tr>
<tr><td>9</td><td>Invoke trcCreateOverlay for first direction (FDX1) specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: TRC_DIR_FDX1.
tTrcOvlConfig structure for the text message from Step 7.
tTrcOvlContent structure for the text message from Step 8.
Overlay user data to identify the text message overlay for TRC_DIR_FDX1.</td></tr>
<tr><td>10</td><td>Invoke trcCreateOverlay for second direction (FDX2) specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: RC_DIR_FDX2.
tTrcOvlConfig structure for the text message from Step 7.
tTrcOvlContent structure for the text message from Step 8.
Overlay user data to identify the text message overlay for TRC_DIR_FDX2.</td></tr>
<tr><td>11</td><td>Wait to receive a TRCEVN_CREATE_OVL_DONE event for all created overlays.
One event will be received for each of the four overlays. Receiving this event does not indicate that the overlay is being displayed, only that the video transcoder platform has accepted the request.
Each TRCEVN_CREATE_OVL_DONE event carries the overlay handle that uniquely identifies the new overlay along with the user data provided by the application on the call to trcCreateOverlay for that specific overlay. The handle will be required on all subsequent calls related to these overlays.</td></tr>
<tr><td>12</td><td>Wait to receive a TRCEVN_CHANNEL_OVL_EVENT event with result set to TRC_OVLEVT_TRCR_RENDER_SUCCESS for all four overlays.
Because the overlays were created in the stopped state, this event does not indicate that the overlay is being displayed, only that the associated content was rendered and is ready.</td></tr>
</table>

Step	Action
13	To make the overlays appear, invoke **trcStartOverlay** repeatedly for each overlay specifying the: Channel handle obtained from Step 1. Overlay handle obtained for each overlay in Step 11.
14	Wait to receive a TRCEVN_START_OVL_DONE event for all four overlays. This event indicates that the overlay is being displayed.
15	The two overlays display in both directions.
16	If the overlays are no longer needed, invoke **trcDestroyOverlay** repeatedly for each overlay specifying the: Channel handle obtained from Step 1. Overlay handle obtained for each overlay in Step 11.
17	Wait to receive a TRCEVN_DESTROY_OVL_DONE event. This event indicates that the overlay was destroyed and the associated content has been released from shared memory (except for cached images which are never released).

In an example like this where the same logo and message is reused regularly on multiple streams, using cached images is a more efficient solution. Only one copy of a cached image resides in shared memory, minimizing memory usage. Additionally, cached images are resized at system startup, avoiding the resize operation at runtime, preserving processor cycles for transcoding.

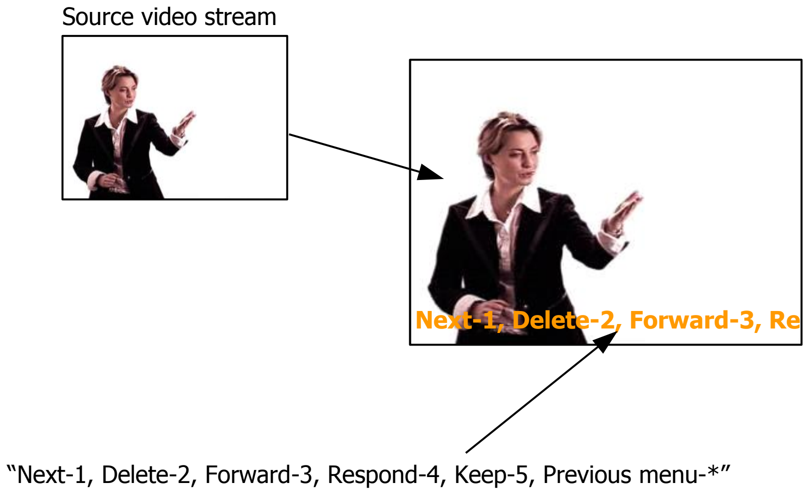
Text can also be cached in a similar fashion using an image file that contains a graphic representation of the required messages. Use an off line image editing software to create the image file. In this case, a graphic overlay is used instead of a text overlay because the content is an image file and not a string of text.

Overlay scrolling

Use scrolling to display content that is larger than the overlay area or the video frame. For example, a text string containing the list of available options can be shown at the bottom of the video image. The following string lists the choices available from a video mail service:

“Next-1, Delete-2, Forward-3, Respond-4, Keep-5, Previous menu-*”

The following example shows this string as an overlay:



Avoid using borders on scrolled overlays. Once content is rendered, borders become part of the content, causing the border to scroll along the rest of the content, which is not generally the desired effect. To have a border around a scrolled overlay, create a second same sized overlay at a layer above the scrolled overlay. This second overlay is created with the required border and a transparent background.

Scrolling is controlled from the scroll field of the tTrcOvlContent structure. This field is a tTrcOvlScroll structure that describes the type, direction, and speed of scrolling. Scrolling can only occur in one direction, diagonal scrolling is not supported. For more information, refer to *tTrcOvlScroll* on page 249.

Generally, scrolling is used with content larger than the overlay area. The size field of the tTrcOvlContent structure is used to provide the content size. The xUnit and yUnit of tTrcOvlCoordinates allow the size of the content to be specified in the following ways:

Value of xUnit or yUnit	Meaning of associated x or y value
TRC_OVL_COORDINATE_UNIT_UNUSED	No size is provided for this coordinate unit. Use the actual size of the rendered content. For text content, setting xUnit to this value is useful to avoid having to determine the length of the bitmap representing the resulting rendered text string. For text, setting yUnit to this value means using the overlay area height (same as yUnit = TRC_OVL_COORDINATE_UNIT_PERCENT with y = 100). For graphic content, this coordinate unit type indicates that the actual image size should be used. It is the responsibility of the application to make sure that the image dimensions are compatible with the overlay area dimension.
TRC_OVL_COORDINATE_UNIT_PIXEL	Actual size in pixels.
TRC_OVL_COORDINATE_UNIT_PERCENT	Size relative to the matching coordinates of the overlay size. Typically this will be a value larger than 100% to indicate a multiple of the overlay size.

Overlay scrolling example

In the following example, the overlay area is defined to use the full width of the video frame (100%), while its height is undefined. This allows the height of the overlay area to assume the height of the content. The content size is also left undefined to allow it to assume the height and width of the bitmap resulting from rendering the specified string. Additionally, the overlay scrolls continuously until it is destroyed.

Step	Action
1	Create a video transcoding channel specifying the TRC_CH_SIMPLEX and TRC_CH_OVERLAY option in the type argument of trcCreateVideoChannel. For more information, refer to *Creating a video channel* on page 58.
2	Start the channel using trcStartVideoChannel. For more information, refer to *trcStartVideoChannel* on page 145.

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>3</td><td>Set up the definition of the overlay area using the tTrcOvlConfig structure:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_TRC_OVL_TYPE_SINGLELINE_TEXT</td></tr>
<tr><td>layer</td><td>1</td></tr>
<tr><td>initState</td><td>TRC_OVL_INITSTATE_STARTED</td></tr>
<tr><td>size.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>size.x</td><td>100 (100% uses the whole width of the video frame)</td></tr>
<tr><td>size.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_UNUSED</td></tr>
<tr><td>size.y</td><td>0 (The value is irrelevant because vertical size is based on content size)</td></tr>
<tr><td>position.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>position.x</td><td>0 (0% start from leftmost position in video frame)</td></tr>
<tr><td>position.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_PERCENT</td></tr>
<tr><td>position.y</td><td>80 (Displays at 80% from the top of the video frame)</td></tr>
<tr><td>fgColor</td><td>TRC_OVL_COLOR_WHITE (overridden by font color)</td></tr>
<tr><td>bgColor</td><td>TRC_OVL_COLOR_TRANSPARENT</td></tr>
<tr><td>border.width</td><td>0</td></tr>
<tr><td>font.name</td><td>Arial</td></tr>
<tr><td>font.size</td><td>20</td></tr>
<tr><td>font.style</td><td>TRC_OVL_FONTSTYLE_NORMAL</td></tr>
<tr><td>font.fgColor</td><td>TRC_OVL_SET_RGB (textForeGround, 255, 240, 0, 255)</td></tr>
<tr><td>font.bgColor</td><td>TRC_OVL_COLOR_DEFAULT (use overlay background color)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>4</td><td>Set up the definition of the overlay content using the tTrcOvlContent structure:
<table>
<tr><th>Field</th><th>Setting</th></tr>
<tr><td>type</td><td>TRC_OVL_CONT_TYPE_TEXT</td></tr>
<tr><td>scroll.type</td><td>TRC_OVL_SCROLL_TYPE_CONTINUOUS</td></tr>
<tr><td>speedDir.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_MS_PER_PIXEL</td></tr>
<tr><td>speedDir.x</td><td>-100 (Overlay scrolls from right to left at a rate of 1 pixel every 100 ms or 10 pixels per second.)</td></tr>
<tr><td>size.xUnit</td><td>TRC_OVL_COORDINATE_UNIT_UNSUSED</td></tr>
<tr><td>size.x</td><td>0 (The value is irrelevant because horizontal content size is based on rendered bitmap size.)</td></tr>
<tr><td>size.yUnit</td><td>TRC_OVL_COORDINATE_UNIT_UNUSED</td></tr>
<tr><td>size.y</td><td>0 (The value is irrelevant because vertical content size is based on rendered bitmap size.)</td></tr>
<tr><td>options</td><td>TRC_OVL_TEXT_CONT_ALIGN_LEFT</td></tr>
<tr><td>ovlData</td><td>"Next-1, Delete-2, Forward-3, Respond-4, Keep-5, Previous menu-*"</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>Invoke trcCreateOverlay specifying:
The channel handle obtained from Step 1.
The appropriate channel direction: TRC_DIR_SIMPLEX.
tTrcOvlConfig structure.
tTrcOvlContent structure.
Overlay user data that will be included to the application in all callbacks related to this overlay.</td></tr>
<tr><td>6</td><td>Wait to receive a TRCEVN_CREATE_OVL_DONE event.
When this event is received, it only indicates that the video transcoder platform has accepted the request. It does not indicate that the overlay is being displayed.
This event carries the overlay handle that uniquely identifies this new overlay along with the user data provided by the application on the call to trcCreateOverlay. The handle will be required on all subsequent call related to this overlay.</td></tr>
</table>

Step	Action
7	The reception of TRCEVN_CHANNEL_OVL_EVENT with result set to TRC_OVLEVT_TRCR_RENDER_SUCCESS indicates that the content was rendered (either the text string was converted into a bitmap representation or the image file was resized). Because the overlay is being created in the started state, this also means that the overlay is being displayed. For such an overlay, this event is mostly informational.
8	If the overlay is no longer needed, invoke **trcDestroyOverlay** specifying the: Channel handle obtained from Step 1. Overlay handle obtained from Step 6.
9	Wait to receive a TRCEVN_DESTROY_OVL_DONE event. This event indicates that the overlay was destroyed and the associated content was released from shared memory (except for cached images, which are never released).

In this example, the overlay scrolls continuously until it is destroyed. It is also possible for the overlay to scroll until the end of the content is reached by setting scroll.type to TRC_OVL_SCROLL_TYPE_CONTENT. Once the end of the content is reached, a TRCEVN_CHANNEL_OVL_EVENT is issued with the result TRC_OVLEVT_VTC_SCROLL_END.

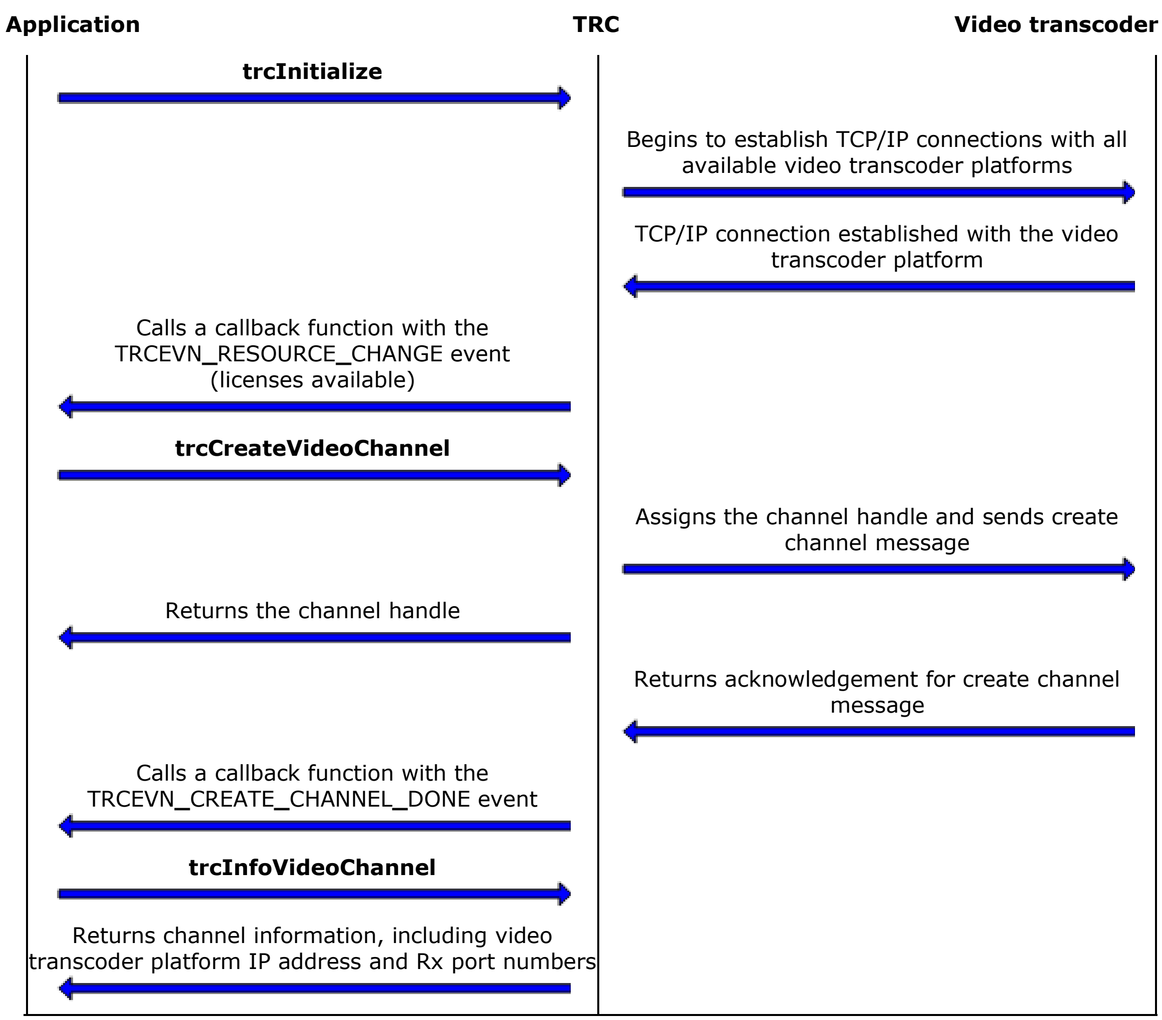
6. Transcoder resource controller call flows

Normal call flow

This topic describes the communication between the application, the TRC, and the video transcoder platform for a normal call flow. It depicts a normal call flow in two parts:

- Normal call flow - Part 1
  Depicts the communication between the application, the TRC, and the video transcoder platform for initializing the TRC, creating a video channel, and obtaining channel information.
- Normal call flow - Part 2
  Depicts the communication between the application, the TRC, and the video transcoder platform for starting a channel, which starts the transcoding. It also illustrates the communication for stopping a channel, destroying a channel, and shutting down the TRC.

Normal call flow - Part 1



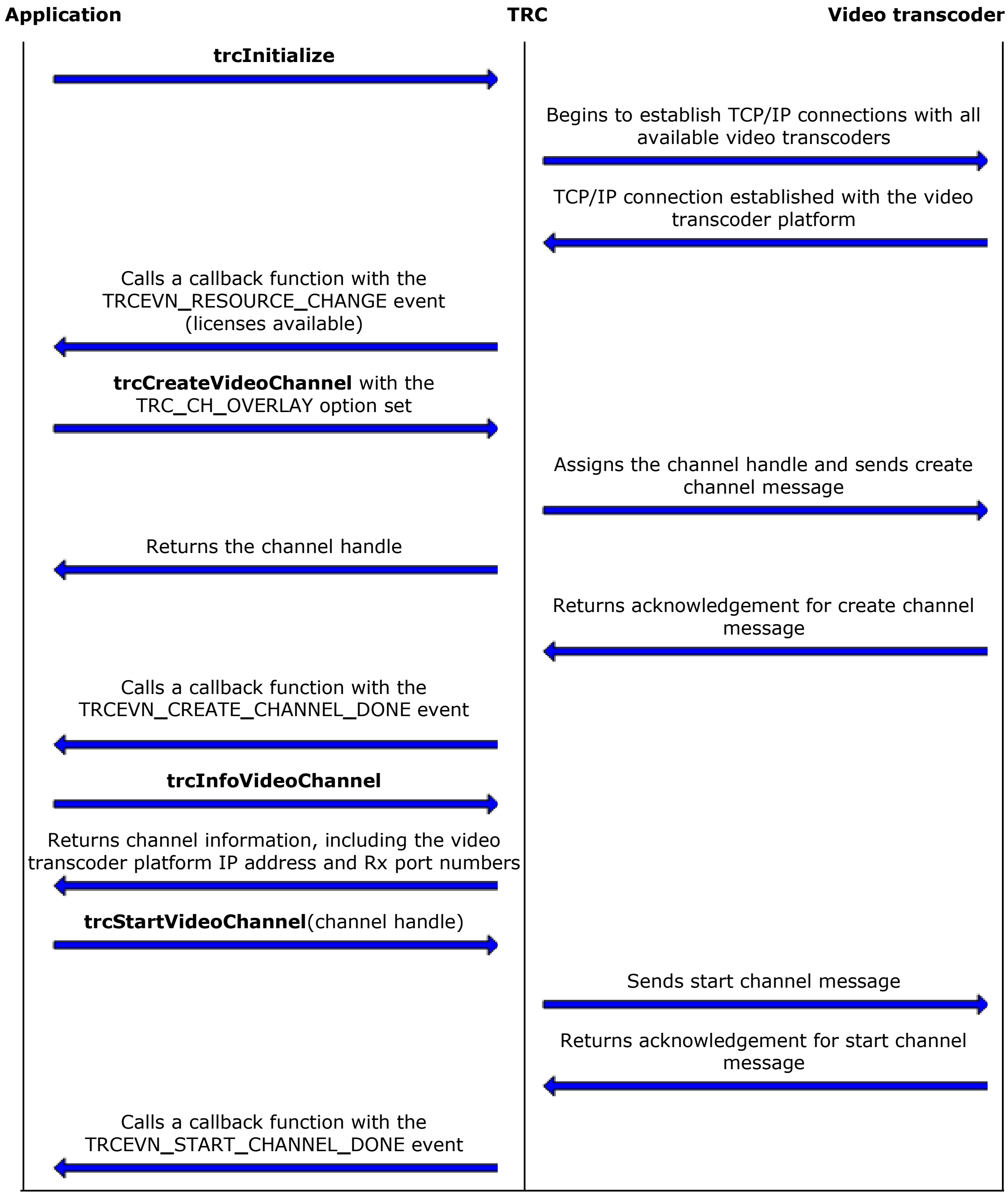
Normal call flow - Part 2

Call flow with overlay

This topic describes the communication between the application, the TRC, and the video transcoder platform for a call flow with overlay. It depicts call flow with overlay in four parts:

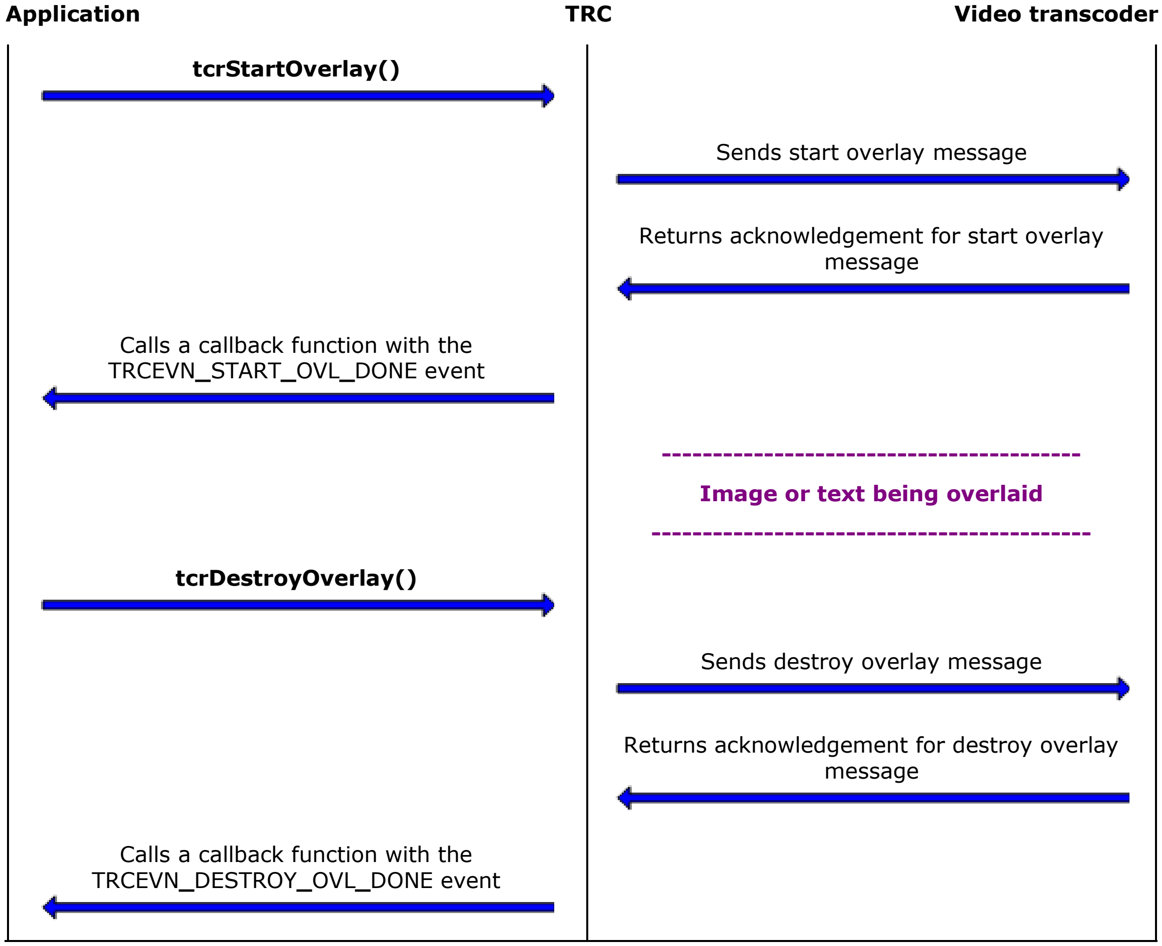
- Call flow with overlay - Part 1
  Depicts the communication between the application, the TRC, and the video transcoder platform for initializing the TRC, obtaining channel information, creating a video channel, and starting a video channel.
- Call flow with overlay - Parts 2 and 3
  Depicts the communication between the application, the TRC, and the video transcoder platform for overlaying text or graphic over the transcoded stream. Shows the overlay being created in a started state, stopped, started again, and destroyed.
- Call flow with overlay - Part 4
  Describes the communication for stopping a channel, destroying a channel, and shutting down the TRC.

Call flow with overlay - Part 1

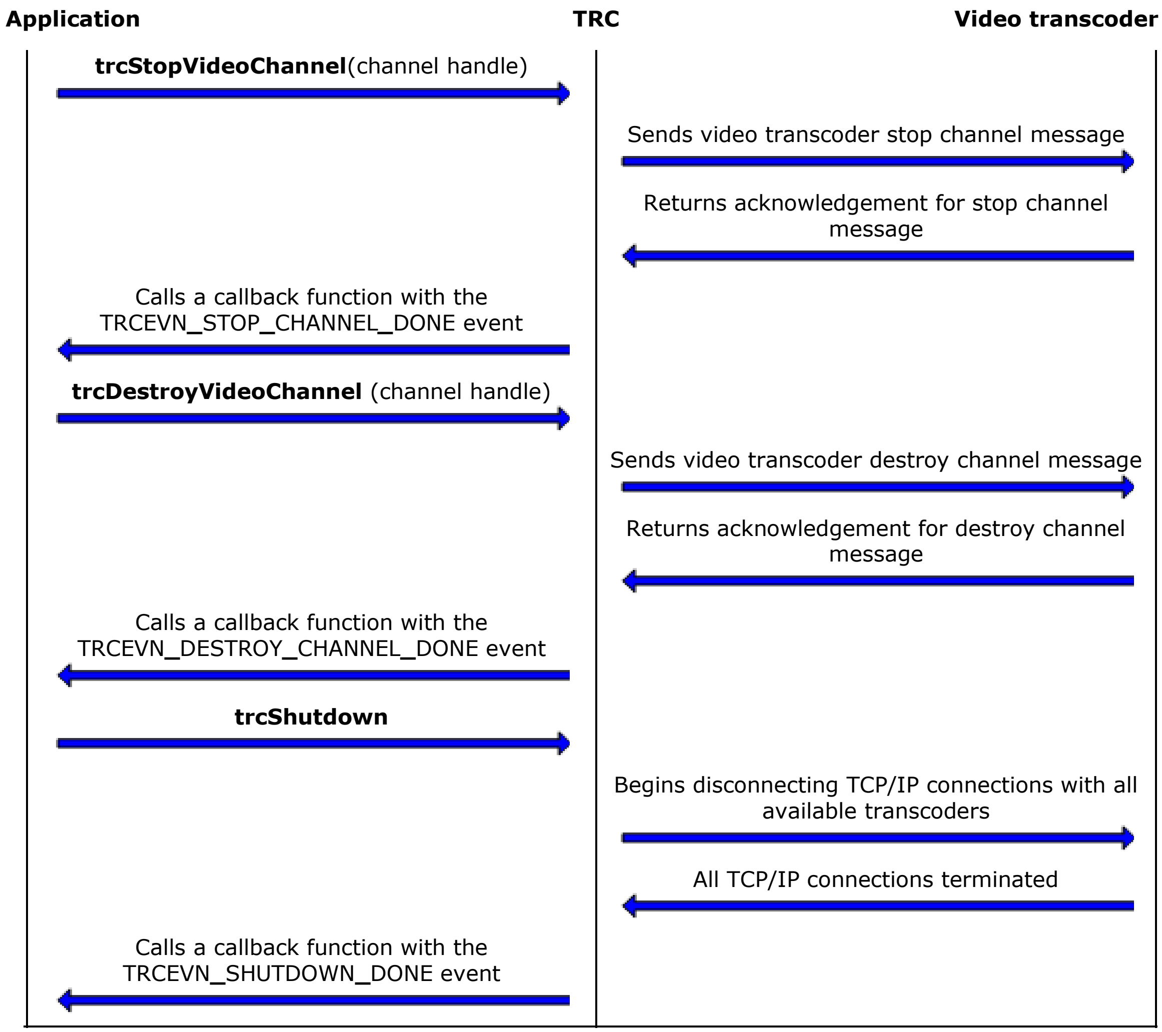


Call flow with overlay - Part 2

Call flow with overlay - Part 3



Call flow with overlay - Part 4



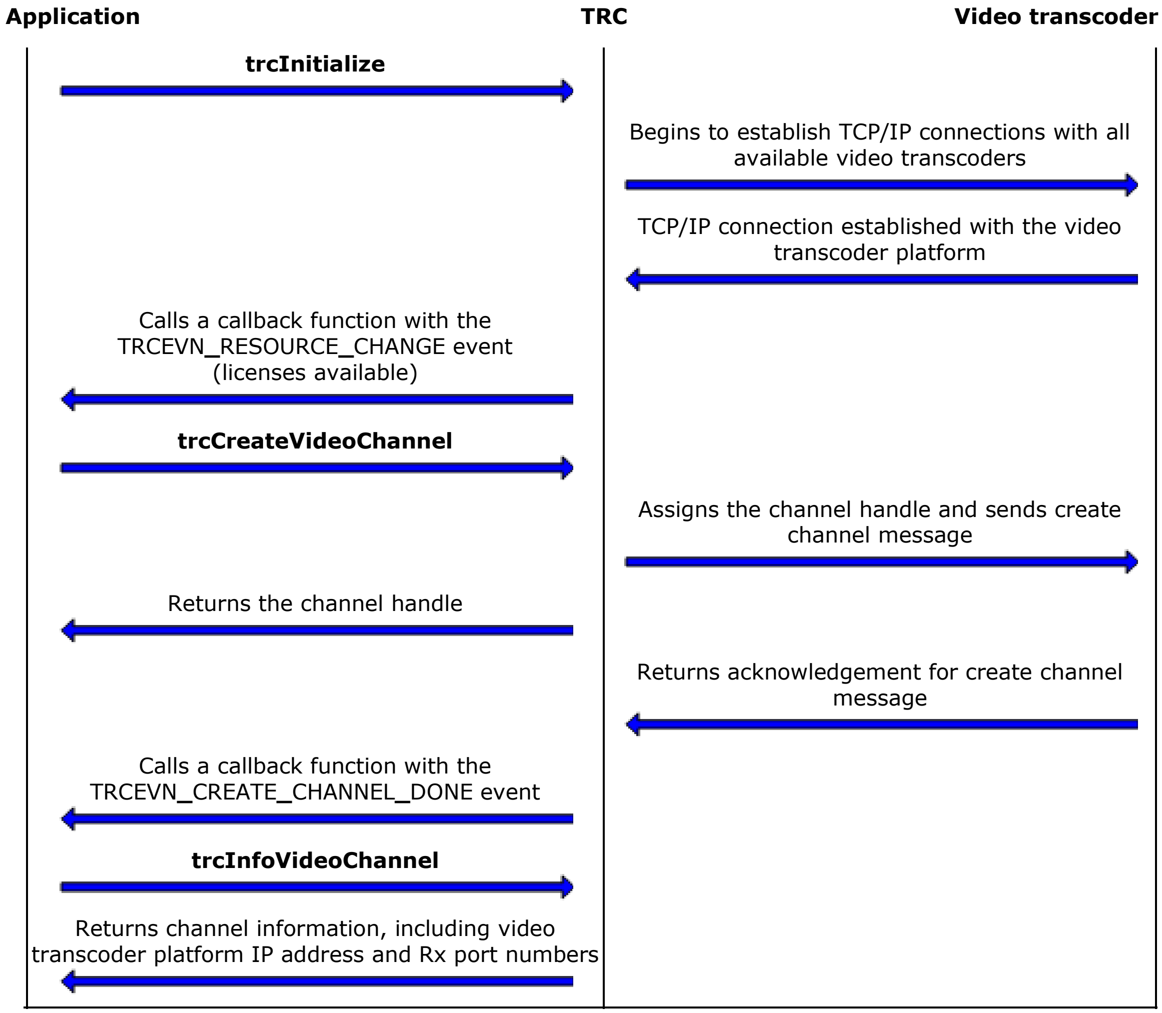
Call flow with a recoverable channel loss

This topic describes what happens when the TRC loses and then recovers contact with the video transcoder platform. It depicts a call flow with a recoverable channel loss in three parts:

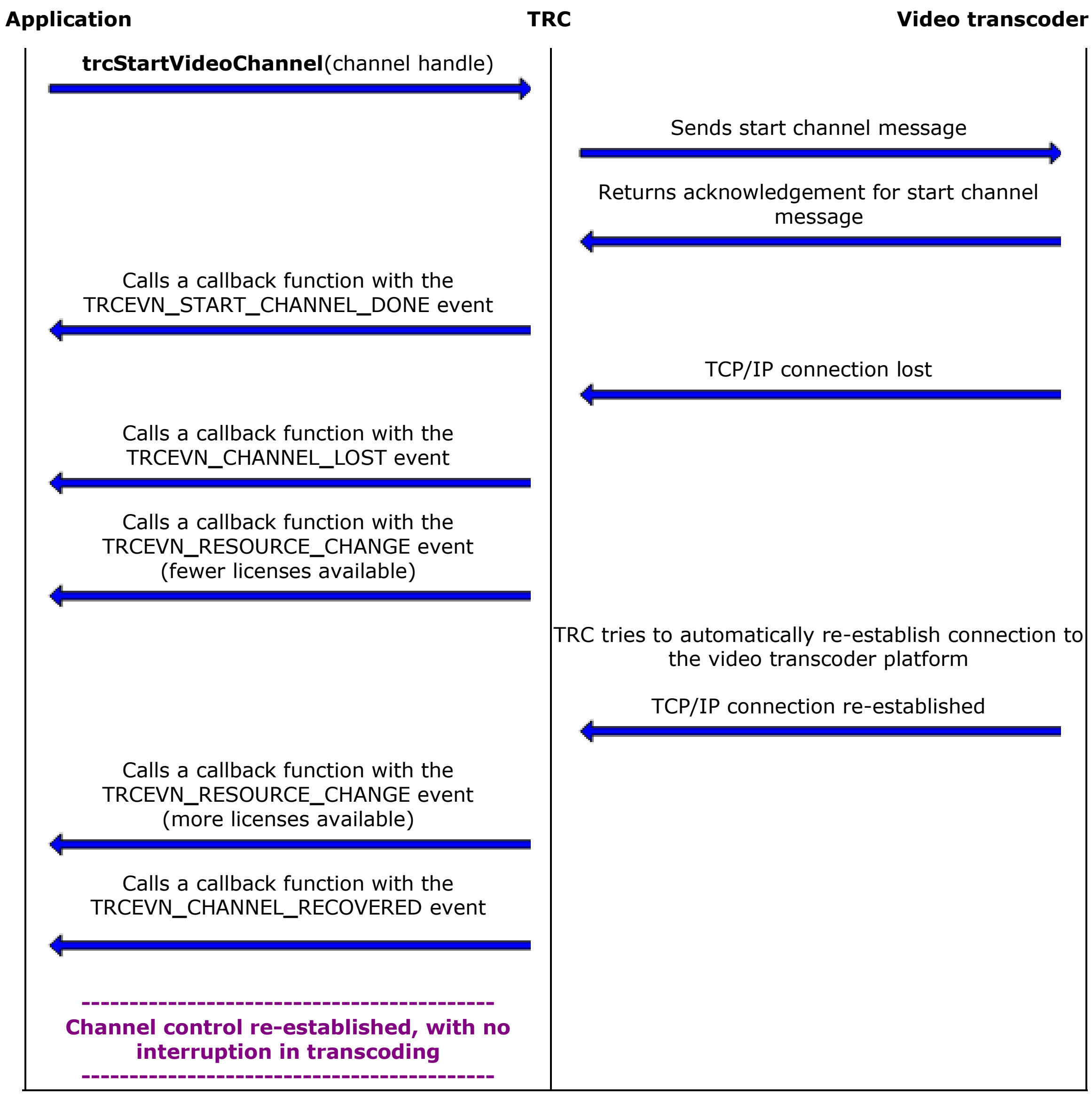
- Call flow with a recoverable channel loss - Part 1
  Depicts the communication between the application, the TRC, and the video transcoder platform for initializing the TRC, creating a video channel, and obtaining channel information.
- Call flow with a recoverable channel loss - Part 2
  Depicts the communication between the application, the TRC, and the video transcoder platform for starting a channel, which starts the transcoding. It also depicts how the video transcoder platform can recover from a communication loss with the TRC.

- Call flow with a recoverable channel loss - Part 3
  Describes the communication for stopping a channel, destroying a channel, and shutting down the TRC.

Call flow with a recoverable channel loss - Part 1



Call flow with a recoverable channel loss - Part 2



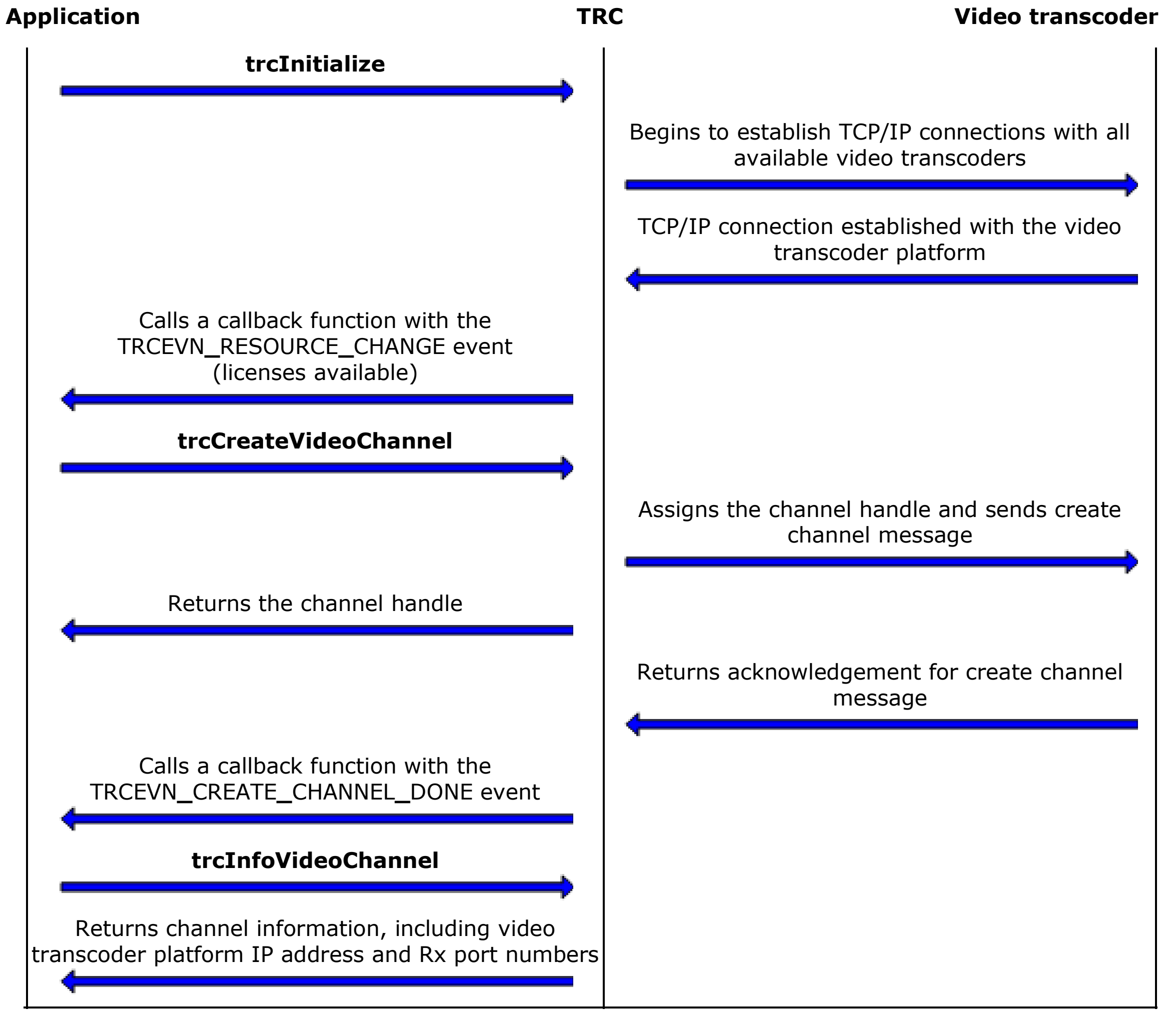
Call flow with a recoverable channel loss - Part 3

Call flow with a non-recoverable channel loss

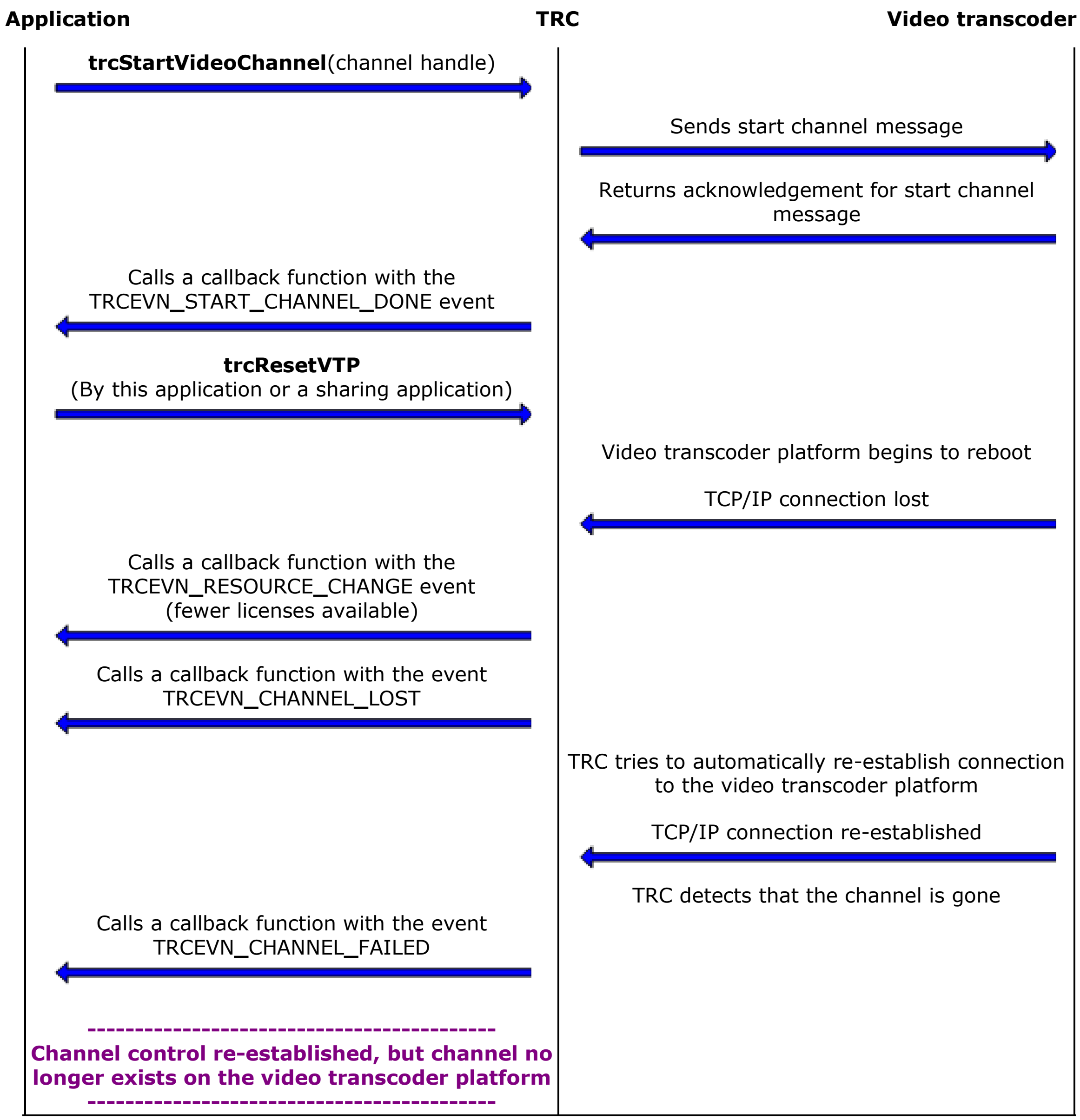
This topic describes the call flow when the TRC loses contact with the video transcoder platform because the video transcoder platform was reset. It depicts a call flow with a non-recoverable channel loss in three parts:

- Call flow with a non-recoverable channel loss - Part 1
  Depicts the communication between the application, the TRC, and the video transcoder platform for initializing the TRC, creating a video channel, and obtaining channel information.
- Call flow with a non-recoverable channel loss - Part 2
  Depicts the communication between the application, the TRC, and the video transcoder platform for starting a channel, which starts the transcoding. It also describes how the video transcoder platform is reset, and how it detects and communicates that the channel is gone.
- Call flow with a non-recoverable channel loss - Part 3
  Describes the communication for destroying the channel, and shutting down the TRC.

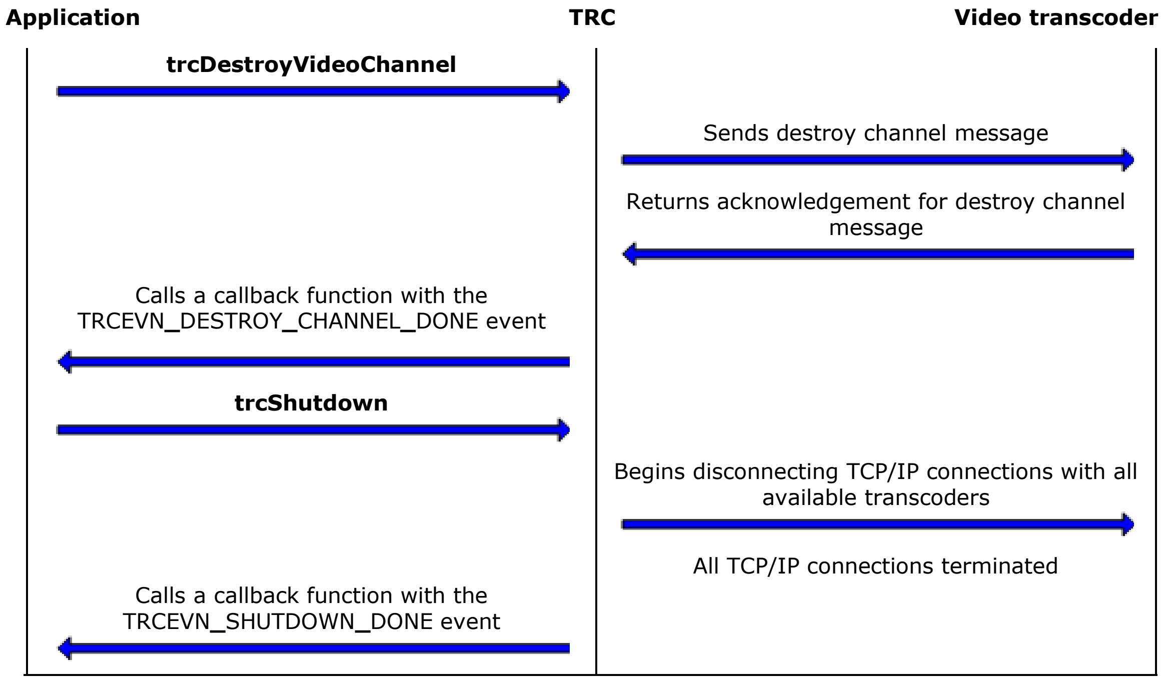
Call flow with a non-recoverable channel loss - Part 1



Call flow with a non-recoverable channel loss - Part 2



Call flow with a non-recoverable channel loss - Part 3



7. Function summary

Transcoder resource controller functions

Transcoder resource controller functions include:

- Channel creation functions
- Monitoring functions
- Overlay functions
- Setup functions
- Transcoding functions

Channel creation functions

All channel creation functions are asynchronous. These functions provide a successful return code indicating that the final status of the operation is provided through an asynchronous event.

Function	Description
trcCreateVideoChannel	Requests the creation of a video channel on an internally selected video transcoder platform.
trcDestroyVideoChannel	Releases a previously allocated transcoder channel.
trcNameVideoChannel	Allows the control application to assign an ASCII name to a channel.

Monitoring functions

All monitoring functions are synchronous.

Function	Description
trcChannelStatus	Retrieves status information for all channels created by the calling application.
trcInfoVideoChannel	Retrieves information about a specific channel, including video transcoder platform-based addressing information assigned when the channel is created.
trcResetVTP	Causes the specified video transcoder platform to reboot, terminating all channels assigned to this video transcoder platform from all applications. The function returns immediately. The video transcoder platform reboot completes when the RESOURCE_CHANGE event includes the rebooted video transcoder platform.
trcSetTrace	Defines the level of tracing for the TRC.

Function	Description
trcUsage	Retrieves channel usage information, including the number of available ports and active channels.
trcValueName	Provides the ASCII name associated with a TRC value (error code, event, and so on).
trcVTPStatus	Retrieves status information for all video transcoder platforms.

Overlay functions

All overlay functions are asynchronous. These functions provide a successful return code indicating that the final status of the operation is provided through an asynchronous event.

Function	Description
trcCreateOverlay	Creates a new overlay.
trcDestroyOverlay	Destroys an existing overlay.
trcStartOverlay	Starts an inactive overlay.
trcStopOverlay	Stops an active overlay.

Setup functions

Function	Synchronous/ Asynchronous	Description
trcInitialize	Synchronous	Initializes connections to all video transcoder platforms. This function is considered synchronous because there is no INITIALIZE DONE event. A successful return from trcInitialize indicates that the initialization was successful, but connections to the video transcoder platforms will be established at a later time. The TRCEVN_RESOURCE_CHANGE event notifies the application of any changes to video transcoder platform connectivity for the TRC.
trcShutdown	Asynchronous	Begins terminating all TCP/IP connections with the available video transcoders.

Transcoding functions

All transcoding functions are asynchronous. These functions provide a successful return code indicating that the final status of the operation is provided through an asynchronous event.

Function	Description
trcIframeVideoChannel	Causes the transcoder to output a full video frame information packet (I-frame).
trcStartVideoChannel	Starts the specified channel, setting adaptation requirements with optional MPEG-4 decoder configurations.
trcStopVideoChannel	Stops all transcoding for a specified video channel.

Management function summary

All management functions are asynchronous. These functions provide a successful return code indicating that the final status of the operation is provided through an asynchronous event.

Function	Description
vtMngEventApp	Issues a request to perform an application-level event.
vtMngEventChn	Issues a request to perform a channel-level event.
vtMngEventMon	Issues a request to perform a process-monitor event.
vtMngEventVtp	Issues a request to perform a video transcoder platform-level event.
vtMngGetApp	Issues a request for details of a particular application.
vtMngGetAppList	Issues a request for the list of connected applications.
vtMngGetChn	Issues a request for details about a particular channel.
vtMngGetChnList	Issues a request for the list of defined channels.
vtMngGetHistPerHHr	Issues a request for a historical statistics record generated each half hour.
vtMngGetHistPerMin	Issues a request for a historical statistics record generated each minute.
vtMngGetMon	Issues a request for details about a particular monitored process.
vtMngGetMonList	Issues a request for the list of monitored processes.
vtMngGetStCurrMin	Issues a statistics request for the current minute.

Function	Description
vtMngGetStTotal	Issues a request for the total (overall) statistics.
vtMngGetVtp	Issues a request for all video transcoder platform-level information.
vtMngInit	Initializes and activates the management communication interface.
vtMngMsg2Host	Converts message from network-byte order to host-byte order.
vtMngMsg2Network	Converts message from host-byte order to network-byte order.
vtMngMsgSize	Returns the total byte size of a message.
vtMngPollLoop	Enters a polling loop, upcalling the management application when a management response or notification is received from a video transcoder platform (or upcalls when a keyboard event occurs).
vtMngSetApp	Issues a request to modify the configuration of a particular application.
vtMngSetChn	Issues a request to modify the configuration of a particular channel.
vtMngSetMon	Issues a request to modify a particular monitored process configuration.
vtMngSetVtp	Issues a request to modify the video transcoder platform-level configuration.
vtMngShutdown	Shuts down all use of the video transcoder management interface.
vtMngValueName	Returns an ASCII name for the given value code of the specified value type.
vtMngZeroApp	Issues a request to get and then reset (zero) an application's statistics.
vtMngZeroChn	Issues a request to get and then reset (zero) a channel's statistics.
vtMngZeroMon	Issues a request to get and then reset (zero) the statistics for a monitored process.
vtMngZeroTotal	Issues a request to get and then reset (zero) the total statistics.
vtMngZeroVtp	Issues a request to get and then reset (zero) the video transcoder platform-level statistics.

8. Transcoder resource controller functions

Using the TRC function reference

This section provides an alphabetical reference to the TRC functions. A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. If a function argument is a structure, the complete structure is shown.
Return values	The return value for a function is either TRC_SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; a subsequent event indicates the completion status of the operation. Refer to the *TRC error summary* on page 308 for a list of errors that the TRC module functions return.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. Additional information such as reason codes and return values appears in the value field of the event. If there are no events listed, the function is synchronous. For more information, refer to *Transcoder resource controller events* on page 312.

trcChannelStatus

Obtains a summary view of all channels currently in use by the calling application.

Prototype

U32 **trcChannelStatus** (tTrcChAll ***chanStatus***)

Argument	Description
chanStatus	Pointer to the tTrcChAll structure. Refer to the Details section for more information.

Return values

Return value	Description
TRC_SUCCESS	Status information was provided successfully.
TRCERR_INVALID_CHANNEL_PARAM	An invalid address was provided as the return structure address.
TRCERR_LIB_NOT_INITIALIZED	Library is not initialized. Call **trcInitialize** first.

Events

None.

Details

The controlling application can call **trcChannelStatus** to obtain current channel status information any time after calling **trcInitialize**. This information is not required for TRC control, but is provided so that the application can monitor overall channel status.

After a successful call to **trcChannelStatus**, the TRC returns channel information in the tTrcChAll structure and its substructures. For more information, refer to *tTrcChAll* on page 222.

Example

```
result = trcChannelStatus( &chanStatus );
if (result == TRC_SUCCESS)
{
    printf( "%d channels defined\n", chanStatus.chanDefined );
    for (i = 0; i < chanStatus.chanDefined; i++)
        {

            printf( "Channel state [%s] on VTP %d:\n",
                    trcValueName( TRCVALUE_CHSTATE,
                    chanStatus.chan[i].status.state ),
                    chanStatus.chan[i].vtpId );
        }
}
else
{
    printf( "Error [%s] while requesting channel status information\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

trcCreateOverlay

Creates a new overlay and optionally set its content.

Prototype

U32 **trcCreateOverlay** (TRC_HANDLE ***trcChHandle***, U16 ***direction***, tTrcOvlConfig ***ovlConfig***, tTrcOvlContent ***ovlContent***, TRC_OVL_USERKEY ***ovlUserKey***)

Argument	Description
trcChHandle	Handle to a video transcoding channel created by **trcCreateVideoChannel**.

Argument	Description
direction	Direction to which the overlay should be output. Valid values are: <table><tr><th>Value</th><th>Description</th></tr><tr><td>TRC_DIR_SIMPLEX</td><td>Command applies to the only connection leg in a simplex channel (endpoint A transcoding to endpoint B).</td></tr><tr><td>TRC_DIR_FDX1</td><td>Command applies to the full-duplex connection leg that is transcoding from endpoint A to endpoint B.</td></tr><tr><td>TRC_DIR_FDX2</td><td>Command applies to the full-duplex connection leg that is transcoding from endpoint B to endpoint A.</td></tr></table>
ovlConfig	Overlay configuration structure. For more information, refer to *tTrcOvlConfig* on page 242.
ovlContent	Structure describing the overlay's content. For more information, refer to *tTrcOvlContent* on page 245.
ovlUserKey	Application-provided key to be associated with the overlay. This key is provided on all overlay-related asynchronous event notifications.

Return values

Return value	Description
TRC_SUCCESS	Overlay create request was successfully issued. The application receives a TRCEVN_CREATE_CHANNEL_OVL_DONE event when the request completes.
TRCERR_INVALID_CHANNEL_HANDLE	Channel handle is not valid.
TRC_OVLEVT_TRCP_INVALID_OVL_DATA	Overlay data too large.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_OUT_OF_MEMORY	Cannot allocate enough memory to send request.
TRCERR_RING_FULL	Ring buffer is full.

Events

Event	Description

Event	Description
TRCEVN_CREATE_OVL_DONE	Overlay creation request is complete. The event results are: TRC_SUCCESS Overlay was successfully created. data[TRCDATA_OVERLAY_OVLHANDLE] Contains the handle that identifies the newly created overlay. This handle can be used on subsequent calls to control this overlay. data[TRCDATA_OVERLAY_USERKEY] Contains the value of *ovlUserKey* passed on the call to trcCreateOverlay. When an error result is received, the content of data[TRCDATA_OVERLAY_OVLHANDLE] will be 0. No overlay handle is returned. TRC_OVLEVT_TRCP_OVL_CREATE_FAILED The overlay creation failed because of problems with the overlay configuration. Refer to the log file for additional information. TRC_OVLEVT_TRCP_DOES_NOT_EXIST The overlay creation failed because the direction provided does not exist. TRC_OVLEVT_TRCP_INVALID_STATE The target channel is unable to process overlays at this time.

Details

The application can invoke this service at any time after receiving the TRCEVN_CREATE_CHANNEL_DONE event and before calling **trcDestroyVideoChannel**. Overlays can be created and started before starting the video channel with **trcStartVideoChannel**, but they only become active once the channel is started.

Example

The following example shows how to create an overlay:

```
tTrcOvlConfig   ovlConfig;
tTrcOvlContent  ovlContent;
U32 result;
ovlConfig.type              = TRC_OVL_TYPE_SINGLELINE_TEXT;
ovlConfig.initState         = TRC_OVL_INITSTATE_STARTED;
ovlConfig.layer             = 1;
ovlConfig.size.xUnit        = TRC_OVL_COORDINATE_UNIT_PERCENT;
ovlConfig.size.x            = 10;
ovlConfig.size.yUnit        = TRC_OVL_COORDINATE_UNIT_PERCENT;
ovlConfig.size.y            = 20;
ovlConfig.position.xUnit    = TRC_OVL_COORDINATE_UNIT_PIXEL;
ovlConfig.position.x        = 100;
ovlConfig.position.yUnit    = TRC_OVL_COORDINATE_UNIT_PIXEL;
ovlConfig.position.y        = 20;
ovlConfig.bgColor           = TRC_OVL_COLOR_TRANSPARENT;
ovlContent.type             = TRC_OVL_CONT_TYPE_TEXT;
ovlContent.options          = TRC_OVL_TEXT_CONT_ALIGN_LEFT;
```

```
ovlContent.scroll.type      = TRC_OVL_SCROLL_TYPE_NONE;
ovlContent.ovlData          = "Test String";
result = trcCreateOverlay( chHandle, TRC_DIR_SIMPLEX, &ovlConfig, &ovlContent,
                           ovlUserKey );
if (result == TRC_SUCCESS)
{
    printf("trcCreateOverlay() request sent successfully\n");
}
else
{
    printf( "Unexpected result from trcCreateOverlay() = 0x%08x [%s]\n",
            result,
            trcValueName(TRCVALUE_RESULT, result) );
}
return( result );
```

The following example shows how to handle the callback event that occurs when the create overlay completes:

```
/*******************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*******************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );

    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_CREATE_CHANNEL_OVL_DONE:
            swicth (pMsg->result)
            {
                case TRC_SUCCESS:
                    printf( "Success: Overlay userKey=%0x$08X handle=%0x%08X\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY],
                            pMsg->data[TRCDATA_OVERLAY_HANDLE] );
                    printf( "created on channel userKey=%p handle=%p\n",
                            pMsg->userKey,
                            pMsg->trcChHandle );
                break;
                case TRC_OVLEVT_TRCP_OVL_CREATE_FAILED:
                case TRC_OVLEVT_TRCP_DOES_NOT_EXIST:
                case TRC_OVLEVT_TRCP_INVALID_STATE:
                    printf( "Failure: Overlay userKey=0x%08X\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY] );
                    printf( "could not be created on channel userKey=%p handle=%p\n",
                            pMsg->userKey,
                            pMsg->trcChHandle );
                break;
                default:
                    printf( "Unexpected error!!!\n");
                    printf( "Failure: Overlay userKey=0x%08X\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY] );
                    printf( "could not be created on channel userKey=%p handle=%p\n",
                            pMsg->userKey,
                            pMsg->trcChHandle );
              }
    }
    return( 0 );   /* always return 0 (successfully received event) */
}
```

trcCreateVideoChannel

Creates a simplex or full-duplex video channel, requesting the assignment of transcoder resources to be dedicated to the given channel.

Prototype

U32 **trcCreateVideoChannel** (TRC_HANDLE ***trcChHandle***, U16 ***type***, void ****userKey***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>trcChHandle</td><td>Pointer to the location where the assigned TRC channel handle is returned. This handle is used as the control object for this channel.</td></tr>
<tr><td>type</td><td>Video channel type. Valid values are:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>TRC_CH_FDX</td><td>Channel type. Full-duplex video channel.</td></tr>
<tr><td>TRC_CH_OVERLAY</td><td>Channel option. Channel required overlay support.</td></tr>
<tr><td>TRC_CH_RTCP</td><td>Channel option. Channel requires RTCP support.</td></tr>
<tr><td>TRC_CH_SIMPLEX</td><td>Channel type. Simplex video channel.</td></tr>
</table>
The channel type must always include either TRC_CH_FDX or TRC_CH_SIMPLEX. All other type modifiers are optional.</td></tr>
<tr><td>userKey</td><td>Pointer to the application-provided key to be associated with the channel. This key is provided on all channel-related asynchronous event notifications.</td></tr>
</table>

Return values

Return value	Description
TRC_SUCCESS	Channel create request was successfully issued with the returned TRC channel handle as the channel control object. The application receives a TRCEVN_CREATE_CHANNEL_DONE event when the request completes.
TRCERR_INVALID_TYPE	Channel type must be TRC_CH_SIMPLEX or TRC_CH_FDX.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_NORESOURCES	No transcoder resources are available.

Events

Event	Description
TRCEVN_CREATE_CHANNEL_DONE	Channel creation request is complete. The event results are: TRC_SUCCESS Channel was created with transcoder resources assigned. Call **trcInfoVideoChannel** for the resource address and RTP port numbers. TRCERR_NORESOURCES There were insufficient resources available on the video transcoder platform when the request was received. This error indicates that a different TRC instance has obtained the desired resources before this create request could be serviced. The application must destroy the failed channel and re-issue a new create request.

Details

After initializing the TRC, the application must wait to be notified through the TRCEVN_RESOURCE_CHANGE event that transcoder resources are available. Once transcoder resources are available, the application creates video channels by calling **trcCreateVideoChannel**.

The TRC verifies that transcoder resources are available before it issues a create request. If no video transcoder platform connections currently exist or if all channels are already in use, the function returns with TRCERR_NORESOURCES.

If resources are available, then the TRC selects a video transcoder platform to assign to the channel. The TRC selects the video transcoder platform from the group of video transcoder platforms that are currently running the smallest number of channels. A randomized selection algorithm is used by the TRC as a method of distributing channel assignments across the least-currently-used video transcoder platforms available to the TRC.

After processing the create request, the video transcoder sends a message to the TRC indicating the completion status. The TRC issues the callback function with the TRCEVN_CREATE_CHANNEL_DONE event and either TRC_SUCCESS or a TRC error code in the result field.

After a channel is successfully created, all transcoding resources for the given channel are reserved and the channel is placed in the STOPPED state (resources are reserved but not actively transcoding). Transcoding does not begin for a created channel until the channel is started using **trcStartVideoChannel**.

Overlays can be created before a channel is started. In this case, any started overlays will display as soon as the channel is started.

To create a channel that can support overlays, the application adds the overlay option to the ***type*** field. For example:

- TRC_CH_FDX + TRC_CH_OVERLAY

- TRC_CH_SIMPLEX + TRC_CH_OVERLAY

Note: A client application can use a mix of video transcoder platforms that are running Video Transcoder 2.0 and Video Transcoder 2.1. When both versions are used, channels requesting TRC_CH_OVERLAY are directed to the video transcoder platform that is running Video Transcoder 2.1. Channels that do not request overlay can be directed to either platform.

See also

trcDestroyVideoChannel

Examples

The following example shows how to create a simplex channel:

```
result = trcCreateVideoChannel( &myChObject->trcChHandle, TRC_CH_SIMPLEX,
                                myChObject );
if (result == TRC_SUCCESS)
{
    printf( "Create simplex channel request in progress\n" );
}
else
{
    printf( "Create simplex channel request failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to create a full-duplex channel requiring both the overlay and RTCP features:

```
result = trcCreateVideoChannel( &myChObject->trcChHandle,
                                TRC_CH_FDX + TRC_CH_OVERLAY +
                                TRC_CH_RTCP,
                                myChObject );
if (result == TRC_SUCCESS)
{
    printf( "Create full-duplex channel request in progress\n" );
}
else
{
    printf( "Create full-duplex channel request failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when the create completes:

```
/***************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
***************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );

    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
            switch( pMsg->event )
    {
        case TRCEVN_CREATE_CHANNEL_DONE:
            if (pMsg->result == TRC_SUCCESS)
```

```
            {
                printf( "Channel created [myChObject=%p]\n",
                        pMsg->userKey );
                printf( "(call trcInfoVideoChannel for resources)\n" );
            }
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcDestroyOverlay

Destroys an existing overlay and any associated content.

Prototype

U32 **trcDestroyOverlay** (TRC_HANDLE ***trcChHandle***, TRC_OVL_HANDLE ***ovlHandle***);

Argument	Description
trcChHandle	Handle to a video transcoding channel created by **trcCreateVideoChannel**.
ovlHandle	Overlay handle created by **trcCreateOverlay**.

Return values

Return value	Description
TRC_SUCCESS	Overlay destroy request was successfully issued. The application receives a TRCEVN_DESTROY_CHANNEL_OVL_DONE event when the request completes.
TRCERR_OUT_OF_MEMORY	Cannot allocate enough memory to send request.
TRCERR_INVALID_CHANNEL_HANDLE	Channel handle is not valid.
TRC_OVLEVT_TRCP_INVALID_OVL_HANDLE	Overlay handle is not valid.
TRCERR_RING_FULL	Ring buffer is full.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.

Events

Event	Description
TRCEVN_DESTROY_OVL_DONE	trcDestroyOverlay is complete. The overlay is no longer being displayed and all related resources were released. This event is issued with the following values: data[TRCDATA_OVERLAY_OVLHANDLE] Contains the overlay handle of the destroyed overlay. data[TRCDATA_OVERLAY_USERKEY] Contains the overlay user key of the destroyed overlay.

Examples

The following example shows how to destroy an overlay:

```
result = trcDestroyOverlay( chHandle, ovlHandle );
if (result == TRC_SUCCESS)
{
    printf( "trcDestroyOverlay() request sent successfully\n" );
}
else
{
    printf( "Unexpected result from trcDestroyOverlay() = 0x%08x [%s]\n",
            result,
            trcValueName(TRCVALUE_RESULT, result) );
}
return( result );
```

The following example shows how to handle the callback event that occurs when the destroy overlay completes:

```
/*****************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*****************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );

    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_DESTROY_OVL_DONE:
            if (pMsg->result == TRC_SUCCESS)
            {
                printf( "Success: Overlay userKey=0x%08X\n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY] );
                printf( "destroyed on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
            else
            {
                printf( "Failure: Overlay userKey=0x%08X\n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY]);
                printf( "could not be destroyed on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcDestroyVideoChannel

Releases a previously allocated transcoder channel.

Prototype

U32 **trcDestroyVideoChannel** (TRC_HANDLE ***trcChHandle***)

Argument	Description
trcChHandle	Valid TRC channel handle returned from **trcCreateVideoChannel**.

Return values

Return value	Description
TRC_SUCCESS	Channel destroy request was accepted. The TRC channel handle is no longer associated with the channel. The TRC performs all actions required to cleanly terminate the channel.

Return value	Description
TRCERR_INVALID_CHANNEL_ID	Identified channel does not exist.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.

Events

Event	Description
TRCEVN_DESTORY_CHANNEL_DONE	Channel destroy request is complete. The event result is: TRC_SUCCESS Channel was destroyed. Activities in progress when the channel was destroyed were aborted. The application will not receive further events related to the destroyed channel.

Details

trcDestroyVideoChannel releases any existing overlays and their associated content. It removes all of the information for a given channel from the TRC database and stops the video channel, if it is still active.

This function sends a destroy channel message to the transcoder and returns to the application without waiting for an acknowledgement from the transcoder. When the TRC receives the acknowledgement, it calls the callback function provided in **trcInitialize** with the TRCEVN_DESTROY_CHANNEL_DONE event. The result field indicates the successful completion or failure of the channel destroy request.

Once a channel is destroyed, the TRC channel handle is no longer valid as the control object for the given channel. The application will not receive other asynchronous events related to this channel.

In the case of a failed call to **trcCreateVideoChannel**, no call to **trcDestroyVideoChannel** is required, since the channel control object was never created. All other channel conditions require a call to **trcDestroyVideoChannel** as the trigger to the TRC to purge all internal handling of the given channel, even in the case where the TRCEVN_CREATE_CHANNEL_DONE event reports an error.

Example

The following example shows how to destroy a channel:

```
result = trcDestroyVideoChannel( trcChHandle );
if (result == TRC_SUCCESS)
{
    printf( "Destroy channel request in progress\n" );
}
else
{
    printf( "Channel destroy request failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when the stop completes:

```
/******************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
******************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
   S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
   S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
   printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
   switch( pMsg->event )
   {
        case TRCEVN_DESTROY_CHANNEL_DONE:
            printf( "Channel destroy done [userKey 0x%X]\n",
                    pMsg->userKey );
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcIframeVideoChannel

Causes the transcoder to insert a full image frame information record (an I-frame) into the outbound video bit stream.

Prototype

U32 **trcIframeVideoChannel** (TRC_HANDLE ***trcChHandle***, U16 ***direction***)

Argument	Description
trcChHandle	Valid TRC channel handle returned from **trcCreateVideoChannel**.
direction	Direction over which an I-frame is to be output. Valid values are: TRC_DIR_SIMPLEX - Generate an I-frame over the only connection leg in a simplex channel (endpoint A transcoding to endpoint B). TRC_DIR_FDX1 - Generate an I-frame over the full-duplex connection leg that is transcoding from endpoint A to endpoint B. BTRC_DIR_FDX2 - Generate an I-frame over the full-duplex connection leg that is transcoding from endpoint B to endpoint A.

Return values

Return value	Description
TRC_SUCCESS	Generate I-frame request was successfully issued. The application receives a TRCEVN_IFRAME_CHANNEL_DONE event when the request completes.
TRCERR_INVALID_CHANNEL_HANDLE	Identified channel does not exist.

Return value	Description
TRCERR_INVALID_CHANNEL_STATE	Channel has not been started.
TRCERR_INVALID_DIRECTION	Invalid direction provided. The direction for simplex channels must be TRC_DIR_SIMPLEX. The direction for full-duplex channels must be TRC_DIR_FDX1 or TRC_DIR_FDX2.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_SOCKET_FAILURE	Unable to send the request over communication socket to the video transcoder platform (connection error).

Events

Event	Description
TTRCEVN_IFRAME_CHANNEL_DONE	Generate I-frame request is complete. The event results are: TRC_SUCCESS The I-frame was successfully generated. TRCERR_INVALID_CHANNEL_STATE Channel is not in a valid state for I-frame generation.

Details

trcIframeVideoChannel causes the transcoder to insert an I-frame (a complete description of the video frame) into the outbound bit stream. The command can generate an I-frame for a simplex channel or for either leg of a full-duplex channel. An I-frame can be requested any time after a channel is started.

After processing the generate I-frame request, the video transcoder sends a message to the TRC indicating the completion status. The TRC issues the callback function with the TRCEVN_IFRAME_CHANNEL_DONE event and either TRC_SUCCESS or a TRC error code in the result field.

See also

trcStartVideoChannel

Example

The following example shows how to generate an I-frame for a simplex channel:

```
result = trcIframeVideoChannel( trcChHandle, TRC_DIR_SIMPLEX );
if (result == TRC_SUCCESS)
{
    printf( "I-Frame request in progress\n" );
}
else
{
    printf( "I-Frame request failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to generate an I-frame for both sides of a full-duplex channel:

```
result = trcIframeVideoChannel( trcChHandle, TRC_DIR_FDX1 );
if (result == TRC_SUCCESS)
{
    result = trcIframeVideoChannel( trcChHandle, TRC_DIR_FDX2 );
}
if (result == TRC_SUCCESS)
    {
        printf( "I-Frames requested for both sides of channel\n" );
    }
else
    {
        printf( "I-Frame request failed [%s]\n",
                trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when an I-frame is issued:

```
/******************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
******************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
    switch( pMsg->event )
        {
            case TRCEVN_IFRAME_CHANNEL_DONE:
                {
                    S8 *dirName = trcValueName( TRCVALUE_DIRECTION,
                                  pMsg->data[TRCDATA_IFRAME_CHANNEL_DIRECTION] );
                    printf( "Channel I-Frame complete, direction [%s]\n",
                            dirName );
                    printf( "\t[userKey 0x%X]\n", pMsg->userKey );
                    break;
                }
        }
return( 0 ); /* always return 0 (successfully received event) */
}
```

trcInfoVideoChannel

Obtains detailed information from the TRC regarding a specific video transcoder channel.

Prototype

U32 **trcInfoVideoChannel** (TRC_HANDLE ***trcChHandle***, tTrcChInfo ****trcChInfo***)

Argument	Description
trcChHandle	Valid TRC channel handle returned from **trcCreateVideoChannel**.
trcChInfo	Pointer to the tTrcChInfo structure, which provides all channel information. Refer to the Details section for more information.

Return values

Return value	Description
TRC_SUCCESS	Channel information was provided successfully.
TRCERR_INVALID_CHANNEL_HANDLE	Identified channel does not exist.
TRCERR_INVALID_CHANNEL_PARAM	Invalid address was provided as the return structure address.

Return value	Description
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.

Events

None.

Details

The controlling application can call **trcInfoVideoChannel** to obtain current channel state and status information any time after receiving a channel handle from **trcCreateVideoChannel**. **trcInfoVideoChannel** is normally called after receiving a successful CREATE_CHANNEL_DONE event to determine the resource addressing information needed to connect external video endpoints to the assigned transcoder RTP endpoints.

After a call to **trcInfoVideoChannel**, the TRC returns all channel information in the tTrcChInfo structure and its substructures. For more information, refer to *tTrcChInfo* on page 224.

Example

```
result = trcInfoVideoChannel( trcChHandle, &trcChInfo );
if (result == TRC_SUCCESS)
{
	printf( "Channel assigned to VTP resource %s\n", trcChInfo.res.ipAddr );
	printf( "Current channel state = %s\n",
		trcValueName( TRCVALUE_CHSTATE, trcChInfo.summary.status.state ));
	if (trcChInfo.summary.status.type == TRC_CH_SIMPLEX)
	{
		printf( "Transcoder receiving RTP bit stream on port %d\n",
			trcChInfo.res.rxPort[TRC_DIR_SIMPLEX] );
	}
	else
	{
		printf( "Transcoder receiving RTP from endpoint A on port %d\n",
			trcChInfo.res.rxPort[TRC_DIR_FDX1] );
		printf( "Transcoder receiving RTP from endpoint B on port %d\n",
			trcChInfo.res.rxPort[TRC_DIR_FDX2] );
	}
}
else
{
	printf( "Error [%s] while requesting channel information\n",
		trcValueName( TRCVALUE_RESULT, result ) );
}
```

trcInitialize

Initializes the TRC so that it can begin establishing connections with all configured video transcoder platforms. **trcInitialize** must be called once before any other TRC function.

Prototype

U32 **trcInitialize** (U16 ***version***, U16 ***revision***, S8 ****appName***, S8 ****configFile***, S8 ****logFileName***, tTrcSendMsg2AppFunc ***pFunc***)

Argument	Description
version	Identifies the version of the TRC for which the application was built. This field is provided so that the TRC can detect an older application built against an incompatible version of the API. The application must use a version of TRC_CTL_VERSION. The value of TRC_CTL_VERSION is incremented any time the TRC API changes in a non-backward compatible manner. All backward compatible API changes are reflected in a change to the *revision* value.
revision	Identifies the revision of the TRC for which the application was built. This field is provided so that the TRC can detect an older application built against a previous compatible version of the API. The application must use a revision of TRC_CTL_REVISION. The value of TRC_CTL_REVISION is incremented with each backward-compatible change to the TRC API, allowing the TRC to perform any actions necessary to maintain backward compatibility support.

Argument	Description
appName	Pointer to the ASCII string that uniquely identifies the calling application as a user of TRC-provided services. This name is used to properly identify the owning application when multiple applications are sharing video transcoder platform resources. The TRC combines the application name with the local host name to produce an identification name. This allows the same application name to be used when each instance of the application is executing on a different client-side system. If multiple application instances exist for a specific client system, then each application name must be unique.
configFile	Pointer to the TRC configuration file name. Refer to the Details section for a sample configuration file.
logFileName	Pointer to the TRC log file name. If NULL, the default is trcapi.log. For more information, refer to **trcSetTrace**.
pFunc	Pointer to the callback function used to pass information to the application.

Return values

Return value	Description
TRC_SUCCESS	TRC initialization process has begun. The TRC is attempting to establish TCP/IP connections with each video transcoder platform listed in the API configuration file.
TRCERR_INVALID_FILE	Invalid configuration file.
TRCERR_INVALID_FUNC	Invalid pointer to the callback function.
TRCERR_INVALID_VERSION	*version* indicated is not compatible with the current version of the TRC. The application must be recompiled before the TRC can function. Refer to the *readme* file for information on migrating to the latest TRC version.
TRCWARN_FUTURE_REVISION	Warning to the application indicating that the revision of the TRC is older than the revision for which the application was built. This warning can be ignored, with the understanding that requested features are limited to those provided by the TRC revision only.

Events

The following events can be issued by the TRC after **trcInitialize** is called:

Event	Description
TRCEVN_CHANNEL_FAILED	Channel has failed. The application must destroy the channel using **trcDestroyVideoChannel**.
TRCEVN_CHANNEL_LOST	Channel connection was lost. The TRC is attempting recovery.
TRCEVN_CHANNEL_RECOVERED	Previously lost channel was recovered.
TRCEVN_RESOURCE_CHANGE	The number of transcoder resources available to the application and to others sharing video transcoder platforms has changed. The event returns the following information in the data array: data[TRCDATA_RESOURCE_CHANGE_AVAILABLE] New number of available transcoder ports. data[RESOURCE_CHANGE_PREVIOUS] Previous number of available ports.

Details

trcInitialize creates the TRC thread that handles all TRC functions, including all asynchronous event notifications. The TRC thread stores the application name and processes the configuration file provided. The configuration file specifies the set of video transcoder platforms that are to be controlled through the TRC. The TRC initiates TCP/IP connections with all available transcoders based on the information in the configuration file.

A successful return from **trcInitialize** indicates that the TRC was successfully configured and is bringing up all video transcoder platform connections. The TRC issues the callback function with the TRCEVN_RESOURCE_CHANGE event as each video transcoder platform connection is established, or any time an established video transcoder platform connection is lost.

The TRC monitors the status of each video transcoder platform connection through heartbeat requests that are sent periodically. Responses to heartbeat requests provide the TRC with the count of channels currently in use by all applications sharing the given video transcoder platform. The TRC uses this information when it selects a video transcoder platform to handle a new channel.

Whenever the total number of resources available to the TRC changes, the TRC informs the application by issuing the callback function with the TRCEVN_RESOURCE_CHANGE event.

This event is issued for the following conditions:

- Initial TRC connection to a video transcoder platform was established.
- TRC connection to a video transcoder platform was lost.
- TRC connection to a video transcoder platform was been re-established.

If a video transcoder platform connection is lost, the TRC uses the TRCEVN_CHANNEL_LOST event to inform the application that each channel assigned to that video transcoder platform was lost. When the connection is re-established, the TRC checks whether each lost channel still exists. If the lost channel exists, the TRC issues the TRCEVN_CHANNEL_RECOVERED event. If the channel no longer exists, the TRC issues the TRCEVN_CHANNEL_FAILED event. The TRC can report TRCEVN_CHANNEL_FAILED for other failure conditions. A failed channel cannot be recovered.

The TRC is also responsible for servicing responses to channel control requests. Responses are reported to the application as TRCEVN_***xxx***_DONE events. These event types are described with the TRC function that issues the associated request.

Configuration file

The configuration file is an ASCII file with commands of the form:

<***keyword***> <***value***> [<***info***>]

The following keywords are supported:

Keyword	Value	Description
vtp	name	Name or IP address of a video transcoder platform to be controlled through the TRC. The video transcoder platform name (or IP address) is used by the TRC to establish a control connection to the given video transcoder platform. It is possible for a video transcoder platform to support both a control address and a media address (the IP address used when connecting all video RTP bit streams). Only the control IP address is specified in the TRC configuration file. Any separate media IP address is provided to the TRC control agent on the video transcoder platform itself. Up to five video transcoder platforms can be specified.

Keyword	Value	Description
heartbeatPeriod	interval	Sets the interval, in milliseconds, at which a heartbeat request is sent to the video transcoder platform. The heartbeat request maintains accurate channel usage information and detects a video transcoder platform connection that has become non-responsive. The default interval is 1000 milliseconds. An interval of 0 indicates that there will be no heartbeat messages. Heartbeat messages are required when multiple applications are sharing video transcoder platform resources. A value of 0 is only valid when all video transcoder platform resources are under exclusive control of a single application.
heartbeatTimeout	duration	Sets the amount of time in milliseconds allowed for a heartbeat response. The TRC waits this amount of time before it considers that the video transcoder platform connection has failed, causing the TRC to disconnect and attempt to reconnect. The default duration is 750 milliseconds. A duration of 0 indicates that there is no timeout for heartbeat responses, and the TRC waits indefinitely.
#		A comment line.

The following example shows a configuration file defining five video transcoder platforms (all controlled over the 10.1.***x***.***x*** network) with an accelerated heartbeat interval and a relaxed timeout duration:

```
# Set of Video Transcoder Platforms (VTPs) controlled by TRC
vtp 10.1.8.1
vtp 10.1.8.2
vtp 10.1.8.3
vtp 10.1.8.4
vtp 10.1.8.5

# Changes from default heartbeat period and heartbeat timeout
heartbeatPeriod = 1500
heartbeatTimeout = 500
```

See also

trcShutdown

Example

The following example shows initializing the TRC:

```
result = trcInitialize( TRC_CTL_VERSION, TRC_CTL_REVISION, "video_mail",
                        "trcapi.cfg", "trcapi.log", trc_callback );
if (result == TRC_SUCCESS)
{
    printf( "TRC initialized, now establishing connections to VTPs\n" );
}
else
{
```

```
    printf( “TRC initialization failed [%s]\n”,
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows handling callback events:

Note: The channel control events (TRCEVN_CHANNEL_LOST, TRCEVN_CHANNEL_FAILED, TRCEVN_CHANNEL_RECOVERED, and TRCEVN_CHANNEL_OVL_EVENT) are only possible after channels are created.

```
/******************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*****************************************************************************
/U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
    printf( “TRC event [%s]: result [%s]\n”, eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_RESOURCE_CHANGE:
            printf( “%d transcoder resources now available\n”,
                   pMsg->data[TRCDATA_RESOURCE_CHANGE_AVAILABLE] );
            printf( “(%d resources previously available\n”,
                   pMsg->data[TRCDATA_RESOURCE_CHANGE_PREVIOUS] );
            break;
        case TRCEVN_CHANNEL_LOST:
            printf( “Channel connection lost [userKey 0x%X]\n”,
                    pMsg->userKey );
            break;
        case TRCEVN_CHANNEL_RECOVERED:
            printf( “Channel connection recovered [userKey 0x%X]\n”,
                   pMsg->userKey );
            break;
        case TRCEVN_CHANNEL_FAILED:
            printf( “Channel has failed [userKey 0x%X]\n”,
                   pMsg->userKey );
            break;
        case TRCEVN_CHANNEL_OVL_EVENT:
            printf( "Received overlay event on channel”
                    “[ch userKey 0x%X] [ch handle 0x%X]\n",
                    pMsg->userKey, pMsg->trcChHandle );
            switch (pMsg->result)
            {
                case TRCP_INFO_TRCR_RENDER_SUCCESS  :
                    printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]”
                            ”being displayed.\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY],
                            pMsg->data[TRCDATA_OVERLAY_OVLHANDLE] );
                    break;
                case TRCP_INFO_VTC_SCROLL_END       :
                    printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]”
                            “end of scrolled content reached.\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY],
                            pMsg->data[TRCDATA_OVERLAY_OVLHANDLE]);
                    break;
                case TRC_OVLEVT_VTC_STARTOVL_FAILED   :
                case TRC_OVLEVT_VTC_STOPOVL_FAILED    :
                case TRC_OVLEVT_VTC_SUBMITC_FAILED    :
                case TRC_OVLEVT_VTC_CREATEOVL_FAILED  :
                case TRC_OVLEVT_VTC_DESTROYOVL_FAILED :
                case TRC_OVLEVT_TRCR_RENDER_FAILED    :
                case TRC_OVLEVT_TRCP_INVALID_OVL_DATA      :
```

```
                    printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]"
                            "error event received [data=%d].\n",
                            pMsg->data[TRCDATA_OVERLAY_USERKEY],
                            pMsg->data[TRCDATA_OVERLAY_OVLHANDLE],
                            pMsg->data[TRCDATA_OVERLAY_EVENTDATA] );
                    break;
            }
            break;
    }
    return( 0 );        /* always return 0 (successfully received event) */
}
```

trcNameVideoChannel

Allows the control application to assign an ASCII name to a channel. This channel name can then be viewed by the management interface.

Prototype

U32 **trcNameVideoChannel**(TRC_HANDLE ***trcChHandle***, S8 ***chName***);

Argument	Description
trcChHandle	Valid channel handle returned from **trcCreateVideoChannel**.
chName	ASCII name to assign to the channel.

Return values

Return value	Description
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_THREAD_USAGE	Improper thread usage by application. API calls cannot be made from within the asynchronous event upcall notification function.
TRCERR_INVALID_CHANNEL_HANDLE	Identified channel does not exist.

Events

None.

Details

Applications can assign names to channels to assist managers in identifying particular channels. For example, a video session could be named using information that is known about the parties involved in the call. When channels are viewed through the management interface, the channels are listed by unique ID values with the channel names. By using identifiable names, a manager can quickly find the channel ID for a specific channel of interest.

The name of a channel can be changed as often as necessary.

See also

trcCreateVideoChannel

Example

```
result = trcNameVideoChannel( trcChHandle, "userA:1111111<FDX>userB:22222222" );
if (result == TRC_SUCCESS)
{
   printf( "Channel name set successfully\n" );
}
else
{
   printf( "Unable to name channel [%s]\n",
           trcValueName( TRCVALUE_RESULT, result ) );
}
```

trcResetVTP

Resets the specified video transcoder platform.

Prototype

U32 **trcResetVTP** (U32 ***vtpId***)

Argument	Description
vtpId	Valid video transcoder platform ID returned from **trcVTPStatus**.

Return values

Return value	Description
TRC_SUCCESS	A video transcoder platform reset was requested, causing the video transcoder platform to reboot. The TRC reports TRCEVN_RESOURCE_CHANGE when the connection is lost and TRCEVN_RESOURCE_CHANGE (again) when the connection is re-established.
TRCERR_INVALID_VTP_ID	Invalid video transcoder platform ID.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_RESET_FAILED	Reset procedure failed. The video transcoder platform cannot be restarted remotely.

Events

No asynchronous completion event is associated with **trcResetVTP**. However, a TRCEVN_RESOURCE_CHANGE event occurs after the video transcoder platform has initiated the reboot, with a subsequent TRCEVN_RESOURCE_CHANGE event when the video transcoder platform has completed the reboot process and the TRC has re-connected to the video transcoder platform.

Details

Use **vtMngEventVtp** or **vtMngEventMon** instead of **trcResetVTP** to handle issues that require recovery of video transcoder platform resources. These management interface functions provide more reset options than **trcResetVTP**. For example, you can perform a warm start in which all processes are terminated and restarted without any system reboot. The only reset that is triggered by the **trcResetVTP** function is a complete reboot of the video transcoder platform.

The **trcResetVTP** function is a legacy way to reboot a video transcoder platform. This function allows an application to recover from conditions in which the TRC is unable to connect to the video transcoder platform. The controlling application can call this function any time after **trcInitialize** has completed to force the video transcoder platform to reboot. When the connection is re-established, the TRC reports the video transcoder platform recovery in the TRCEVN_RESOURCE_CHANGE event.

Multiple applications can share the transcoder resources being provided by the video transcoder platform. If any application issues **trcResetVTP**, all applications using this video transcoder platform will have their channels fail as the video transcoder platform reboots. This effect must be taken into consideration before deciding that a reset of the video transcoder platform is required.

If any video transcoder channel fails, the TRC issues the callback function with the TRCEVN_CHANNEL_LOST event. This event is an early indication of a connection failure. Once the video transcoder platform reboots and the TRC re-establishes its connection with the video transcoder platform, the TRC detects that all channels were terminated. The TRC then issues the TRCEVN_CHANNEL_FAILED event for each channel that had existed on the given video transcoder platform to indicate the critical channel failure.

The reset request is sent to a separate process on the video transcoder platform that is dedicated to providing the remote reset capability. In cases where the physical connection to the video transcoder platform has failed, **trcResetVTP** cannot trigger the reset.

Example

```
result = trcResetVTP( vtpId );
if (result == TRC_SUCCESS)
{
    printf( "VTP reset request successfully issued\n" );
}
else
{
    printf( "VTP reset request failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

trcSetTrace

Defines the logging level for the TRC. This function can be called any time before or after **trcInitialize**.

Prototype

void **trcSetTrace** (U32 ***logToConsoleMask***, U32 ***logToFileMask***)

Argument	Description
logToConsoleMask	Tracing mask for the console.
logToFileMask	Tracing mask for the log file.

Details

Use **trcSetTrace** to diagnose TRC-related issues that are not understood through error codes or event notifications. Tracing information can be directed to the console, to the TRC log file, or to both.

There is no return value for this function since the function always succeeds (even when the TRC has not yet been initialized). Tracing can be configured prior to TRC initialization, so that the **trcInitialize** flow can be traced, if needed.

The following table lists the tracing types that you can specify for the console and the file trace masks:

Tracing type	Value	Description
TRCTR_ACT	(1<<14)	Trace action routines.

Tracing type	Value	Description
TRCTR_ALM	(1<<1)	Trace all alarm conditions. TRCTR_ALM is active by default.
TRCTR_API	(1<<4)	Trace the interface between the application and the TRC API.
TRCTR_CHN	(1<<11)	Trace channel-level control information.
TRCTR_DBG	(1<<21)	Trace internal debugging information.
TRCTR_ERR	(1<<0)	Trace all detected errors. TRCT_ERR is active by default.
TRCTR_EVT	(1<<13)	Trace all events.
TRCTR_HDR	(1<<20)	Trace detailed header information.
TRCTR_MSG	(1<<5)	Trace communication messages between the TRC API and video transcoder platform agent.
TRCTR_STA	(1<<12)	Trace all state changes.
TRCTR_THR	(1<<8)	Trace thread execution control.
TRCTR_TMR	(1<<16)	Trace timers.
TRCTR_TOP	(1<<9)	Trace top-level control information.
TRCTR_VTP	(1<<10)	Trace video transcoder platform-level control information.

The following tracing types are for internal use only:

- TRCTR_ALL
- TRCTR_CON
- TRCTR_FIL

Example

```
/* activate tracing of application interface and all detected errors to the log file */
trcSetTrace( 0, TRCTR_API|TRCTR_ERR );
```

trcShutdown

Triggers the TRC to begin breaking all TCP/IP connections with the available video transcoder platforms. Any active transcoder channels (channels for which **trcDestroyVideoChannel** was not previously called) are destroyed prior to disconnecting the TCP/IP sessions. A call to **trcShutdown** is therefore considered as an implied call to **trcDestroyVideoChannel** for all channels owned by the application.

Prototype

U32 **trcShutdown** (void)

Return values

Return value	Description
TRC_SUCCESS	TRC shutdown successfully initiated.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.

Events

Event	Description
TRC_SHUTDOWN_DONE	TRC is completely shut down.

Details

An application uses **trcShutdown** to completely deactivate all transcoding channels owned by the given application that exist on any of the video transcoder platforms that the TRC controls. After destroying all channels, the TRC disconnects all TCP/IP connections with the video transcoder platforms. **trcShutdown** cleans up all TRC module resources, including all threads and timers used within the TRC. The TRC issues the callback function with the TRCEVN_SHUTDOWN_DONE event, and either TRC_SUCCESS or a TRC error code in the result field. Currently, the TRC always reports a successful shutdown, since there is no error condition that can keep the API from terminating. Applications must test the result code to handle any future error conditions that could result in a failed shutdown attempt.

Once the TRC completes the shutdown, the application must call **trcInitialize** again prior to using any transcoder resources.

Examples

The following example shows terminating channels and disconnecting video transcoder platforms:

```
/* terminate all channels owned by the application and disconnect from all VTPs */
result = trcShutdown( );
if (result == TRC_SUCCESS)
{
    printf( “TRC shutting down\n” );
}
else
{
    printf( “TRC shutdown failed [%s]\n”,
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when the TRC shutdown completes:

```
/*****************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*****************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_SHUTDOWN_DONE:
            printf( "Shutdown complete\n" );
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcStartOverlay

Starts an inactive overlay.

Prototype

U32 **trcStartOverlay** (TRC_HANDLE ***trcChHandle***, TRC_OVL_HANDLE ***ovlHandle***)

Argument	Description
trcChHandle	Handle to a video transcoding channel created by **trcCreateVideoChannel**.
ovlHandle	Overlay handle created by **trcCreateOverlay**.

Return values

Return value	Description
TRC_SUCCESS	Overlay start request was successfully issued. The application receives a TRCEVN_START_CHANNEL_OVL_DONE event when the request completes. The application also receives a TRCEVN_CHANNEL_OVL_EVENT event indication with the result field set to TRC_OVLEVT_TRCR_RENDER_SUCCES S once the content is rendered and displayed.
TRCERR_OUT_OF_MEMORY	Cannot allocate enough memory to send request.

Return value	Description
TRCERR_INVALID_CHANNEL_HANDLE	Channel handle is not valid.
TRC_OVLEVT_TRCP_INVALID_OVL_HANDLE	Overlay handle is not valid.
TRCERR_RING_FULL	Ring buffer is full.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.

Events

Event	Description
TRCEVN_CHANNEL_OVL_EVENT	Indicates an asynchronous overlay event. These events can either be informational or indicate that an error occurred while processing an overlay. The specific type of event is reported in the field result of the tTrcMessage structure. The defined informational messages are: TRC_OVLEVT_TRCR_RENDER_SUCCESS TRC_OVLEVT_VTC_SCROLL_END The defined error conditions are: TRCP_INFO_TRCR_RENDER_SUCCESS TRCP_INFO_VTC_SCROLL_END TRC_OVLEVT_VTC_STARTOVL_FAILED TRC_OVLEVT_VTC_STOPOVL_FAILED TRC_OVLEVT_VTC_SUBMITC_FAILED TRC_OVLEVT_VTC_CREATEOVL_FAILED TRC_OVLEVT_VTC_DESTROYOVL_FAILED TRC_OVLEVT_TRCR_RENDER_FAILED TRC_OVLEVT_TRCP_INVALID_OVL_DATA
TRCEVN_START_OVL_DONE	trcStartOverlay is being processed.

Examples

The following example shows how to start an inactive overlay:

```
result = trcStartOverlay( chHandle, ovlHandle);
if (result == TRC_SUCCESS)
{
    printf( "trcStartOverlay() request in progress\n" );
}
else
{
    printf( "Unexpected result from trcStartOverlay() = 0x%08x [%s]\n",
```

```
            result,
            trcValueName(TRCVALUE_RESULT, result) );
}
return( result );
```

The following example shows how to handle the callback event that occurs when the start overlay completes:

```
/*****************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
****************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );

    printf( “TRC event [%s]: result [%s]\n”, eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_START_OVL_DONE:
            if (pMsg->result == TRC_SUCCESS)
            {
                printf( "Success: Overlay userKey=%p \n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY]);
                printf( "Started on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
            else
            {
                printf( "Failure: Overlay userKey=%p\n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY]);
                printf( "could not be started on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
        break;
        case TRCEVN_CHANNEL_OVL_EVENT:
           printf( "Received overlay event on channel”
                   “[ch userKey 0x%X] [ch handle 0x%X]\n",
                   pMsg->userKey, pMsg->trcChHandle);
           switch (pMsg->result)
           {
               case TRCP_INFO_TRCR_RENDER_SUCCESS  :
                   printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]”
                           ”being displayed.\n",
                           pMsg->data[TRCDATA_OVERLAY_USERKEY],
                           pMsg->data[TRCDATA_OVERLAY_OVLHANDLE]);
                   break;
               case TRCP_INFO_VTC_SCROLL_END       :
                   printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]”
                           “end of scrolled content reached.\n",
                           pMsg->data[TRCDATA_OVERLAY_USERKEY],
                           pMsg->data[TRCDATA_OVERLAY_OVLHANDLE]);
                   break;
               case TRC_OVLEVT_VTC_STARTOVL_FAILED   :
               case TRC_OVLEVT_VTC_STOPOVL_FAILED    :
               case TRC_OVLEVT_VTC_SUBMITC_FAILED    :
               case TRC_OVLEVT_VTC_CREATEOVL_FAILED  :
               case TRC_OVLEVT_VTC_DESTROYOVL_FAILED :
               case TRC_OVLEVT_TRCR_RENDER_FAILED    :
               case TRC_OVLEVT_TRCP_INVALID_OVL_DATA      :
                   printf( "Overlay [ovl userKey 0x%X] [ovl handle 0x%X]”
```

```
                              "error event received [data=%d].\n",
                              pMsg->data[TRCDATA_OVERLAY_USERKEY],
                              pMsg->data[TRCDATA_OVERLAY_OVLHANDLE],
                              pMsg->data[TRCDATA_OVERLAY_EVENTDATA]);
                      break;
              }
          break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcStartVideoChannel

Starts the specified channel and provides a full description of the video transcoding adaptations that the channel performs.

Prototype

U32 **trcStartVideoChannel** (TRC_HANDLE ***trcChHandle***, tTrcChConfig ***chConfig***)

Argument	Description
trcChHandle	Valid channel handle returned from **trcCreateVideoChannel**.
chConfig	Pointer to the tTrcChConfig structure, which configures the endpoints to which the transcoding channel connects.

Return values

Return value	Description
TRC_SUCCESS	Start request was successfully issued. The application receives a TRCEVN_START_CHANNEL_DONE event when the request completes.
TRCERR_INVALID_CHANNEL_HANDLE	Identified channel does not exist.
TRCERR_INVALID_CHANNEL_PARAM	Invalid video type provided. The value of vidType must be TRC_VIDTYPE_MPEG4 or TRC_VIDTYPE_H263.
TRCERR_INVALID_CHANNEL_STATE	Channel is already started or the channel is stopped.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_SOCKET_FAILURE	Unable to send the request over communication socket to the video transcoder platform (connection error).

Events

Event	Description
TRCEVN_START_CHANNEL_DONE	Channel start request is complete. The event results are: TRC_SUCCESS Channel was successfully started. TRCERR_INVALID_CHANNEL_PARAM One or more of the channel parameters provided in the tTrcEndpoint structure is invalid: profile must be SIMPLE for MPEG-4 and BASELINE for H.263. level must be TRC_MPEG4_LEVEL_[0-3] for MPEG-4 and TRC_H263_LEVEL_[10, 20, or 30] for H.263. dataRate cannot exceed 384 kbit/s. frameRate cannot exceed 30 fps. frameRes must be TRC_FRAME_RES_[QCIF or CIF]. packetizeMode must be TRC_PACKETIZE_[2190, 2429, or 3016]. TRCERR_INVALID_CHANNEL_STATE The channel is not in a valid state for the requested command. Call **trcStopVideoChannel** to stop the channel. TRCERR_INVALID_INPUT_PARAM One or more of the channel input parameters is invalid. TRCERR_INVALID_OUTPUT_PARAM One or more of the channel output parameters are invalid. The ipAddr must be a valid ASCII IP address.

Details

trcStartVideoChannel sends a start channel message to the transcoder and returns the interim result. If the function returns TRC_SUCCESS, the TRC calls the callback function provided in **trcInitialize** with the TRCEVN_START_CHANNEL_DONE event. The result field indicates the success or failure of the channel start.

The application uses the tTrcChConfig structure and its substructures to provide the configuration of each endpoint to which the channel is to connect. This includes the video type of each endpoint (H.263 or MPEG-4) with profile, level, data rate, frame rate, frame resolution (QCIF or CIF), and packetization mode. For more information, refer to *tTrcChConfig* on page 223.

Dialogic recommends that you zero-fill the channel configuration structures before setting parameters to produce applications that are forward compatible with future revisions of the TRC. A value of zero for any optional parameter indicates that the option is not in use. The literal TRC_CONFIG_DEFAULT is defined as zero, and can be used for any parameter that supports a default value. If an endpoint field that does not support a default value is set to TRC_CONFIG_DEFAULT, the channel start request fails with an indication that required information is missing.

The following table provides a description of how the **trStartVideoChannel** function uses the configuration structures to perform specific tasks:

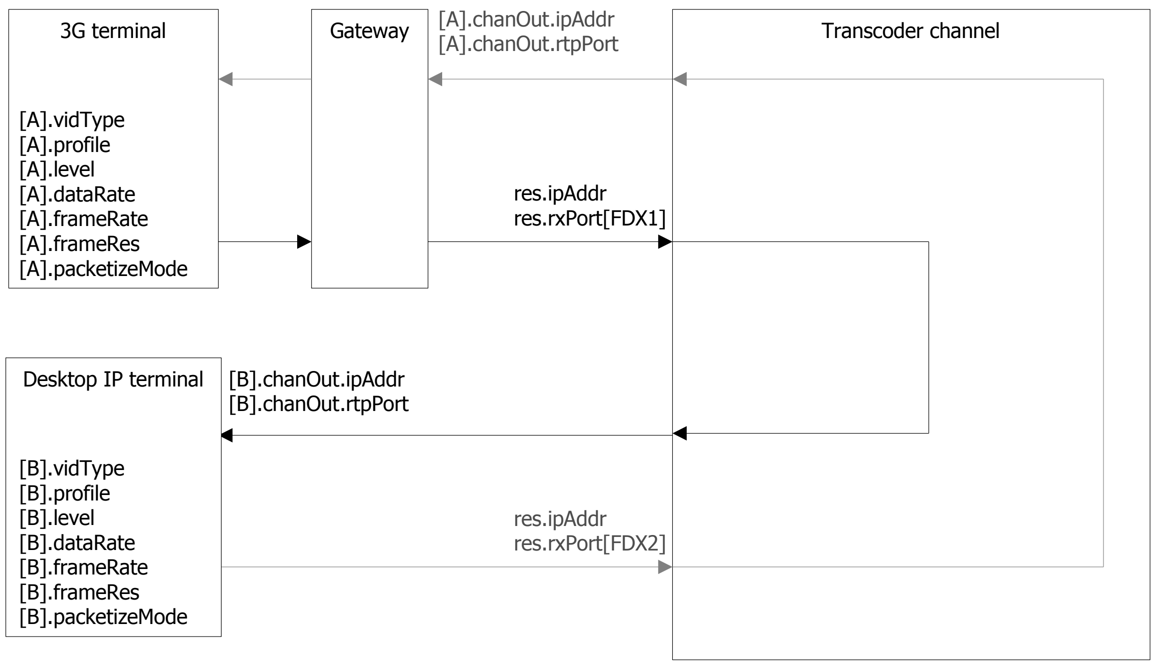
<table>
<tr><th>trStartVideoChannel uses this structure...</th><th>To complete this task...</th></tr>
<tr><td>tTrcEndpoint</td><td>Configure general endpoint fields for send and receive endpoints. For more information, refer to tTrcEndpoint on page 235.</td></tr>
<tr><td>tTrcEndInput</td><td>Configure channel input fields. These fields define the characteristics of an endpoint that sends input to the transcoder channel. For more information, refer to tTrcEndInput on page 228.</td></tr>
<tr><td>tTrcEndOutput</td><td>Configure channel output fields. These fields define the characteristics of an endpoint that receives output from a transcoder channel. For more information, refer to tTrcEndOutput on page 230.</td></tr>
<tr><td>tTrcChOptions</td><td>Uses the tTrcChOptions structure to specify these settings:
<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>optData</td><td>Address of the optional configuration block.</td></tr>
<tr><td>optSize</td><td>Byte length of the block.</td></tr>
</table>

The transcoder supports the optional configuration of decoder and encoder capabilities for each direction in use by a channel. For example, the following options are set to provide decoder configuration information for both directions of a full-duplex channel:
cfg.decoder[TRC_DIR_FDX1].optData = &dciA2B;
cfg.decoder[TRC_DIR_FDX1].optSize = sizeof(dciA2B);
cfg.decoder[TRC_DIR_FDX2].optData = &dciB2A;
cfg.decoder[TRC_DIR_FDX2].optSize = sizeof(dciB2A);
For more information, refer to tTrcChOptions on page 225.</td></tr>
</table>

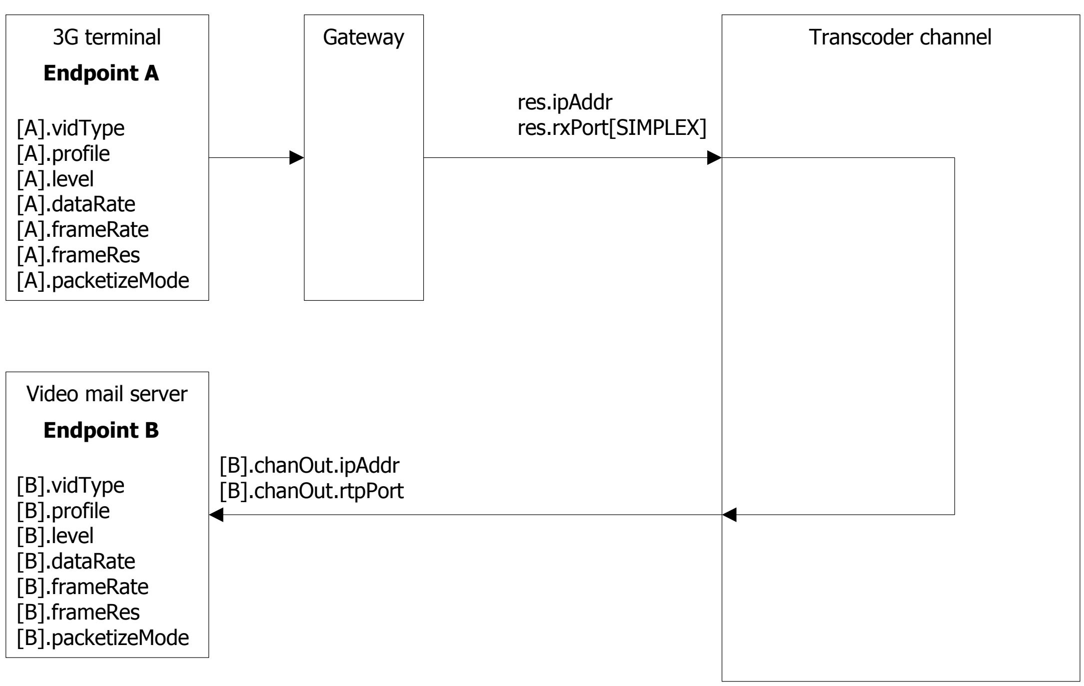
Configuring full-duplex and simplex transcoder channels

The following sample configurations show how to configure full-duplex and simplex transcoder channels.

Configuring a full-duplex channel

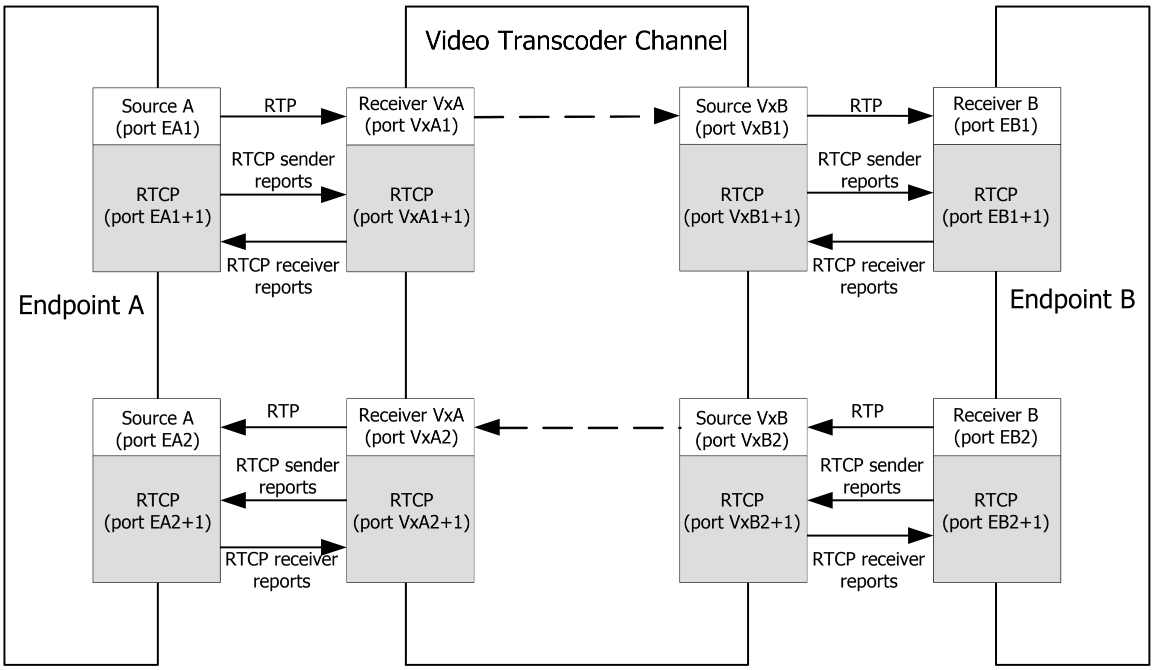


Configuring a simplex channel



Optional RTCP configuration

For each RTP data stream, there can also be an associated RTCP data stream. The following illustration shows the RTCP flows that may exist for a full-duplex transcoder channel. The top half of the illustration can be considered as a simplex channel with endpoint A sending RTCP sender reports and receiving receiver reports; while endpoint B receives sender reports and sends receiver reports:



The following table provides a description of the RTCP options:

Option	Description
rtcpReceiver	Operating mode for handling RTCP communication with the input endpoint: TRC_RTCP_DEFAULT Handle RTCP as specified by video transcoder platform-level default configuration (rtcpMode). TRC_RTCP_DISABLED Do not listen for RTCP. TRC_RTCP_ENABLED Listen for receive of RTCP from endpoint and send RTCP to remote as appropriate.
rtcpRxTimeout	Maximum amount of time (in milliseconds) that can pass without receiving RTP or RTCP before considering input endpoint timed out (0 = no timeout [DEFAULT]). Note: This field is only valid when rtcpReceiver is not set to TRC_RTCP_DEFAULT.

Option	Description
rtcpTransmitter	Operating mode for handling RTCP communication with the output endpoint: TRC_RTCP_DEFAULT Handle RTCP as specified by video transcoder platform-level default configuration (rtcpMode). TRC_RTCP_DISABLED Do not send RTCP. TRC_RTCP_ENABLED Listen for receive of RTCP from endpoint and send RTCP to remote as appropriate (with all transmission triggered by receiving RTCP from the input endpoint).
rtcpTxTimeout	Maximum amount of time (in milliseconds) that can pass without receiving RTP or RTCP before considering output endpoint timed out (0 = no timeout [DEFAULT]). **Note:** This field is only valid when rtcpTransmitter is not set to TRC_RTCP_DEFAULT.

The simplest way to activate RTCP for all channels is to set the video transcoder platform-level configuration field rtcpMode to ENABLED. This causes all video transcoder channels to act as RTCP translators. Use the video transcoder platform-level configuration fields rtcpInTimeout and rtcpOutTimeout to control whether inactivity is monitored by default and, if so, what time period to use.

You can also allow each channel to indicate whether RTCP should be registered for. In this case, the application sets the configuration elements, taking over RTCP configuration from any video transcoder platform-level defaults.

The TRC API can also manage video transcoder platforms that support the RTCP feature while also managing some that do not. To guarantee that a video transcoder platform is selected with RTCP capability, the application must specify the TRC_CH_RTCP option as part of the channel type provided to the **trcCreateVideoChannel** function.

Examples

The following example shows a full-duplex channel between MPEG-4 and H.263:

```
 /* start a full-duplex channel between MPEG-4 endpoint (A) and H.263 endpoint (B)   */

memset( &cfg, TRC_CONFIG_DEFAULT, sizeof(cfg) );      /* start from all defaults     */
cfg.endpointA.vidType = TRC_VIDTYPE_MPEG4;
cfg.endpointA.profile = TRC_PROFILE_SIMPLE;
cfg.endpointA.level = TRC_MPEG4_LEVEL_0;
cfg.endpointA.dataRate = 43; /* kbits/sec */
cfg.endpointA.frameRate = 7; /* frames/sec */
cfg.endpointA.packetizeMode = TRC_PACKETIZE_3016;
cfg.endpointA.chanIn.jitterMode = TRC_JITTER_NONE;    /* no jitter buffer used       */
strcpy( cfg.endpointA.chanOut.ipAddr, "192.68.2.1" );
cfg.endpointA.chanOut.rtpPort = 1000;
cfg.endpointA.chanOut.payloadID = 100;
cfg.endpointA.chanOut.tos = 0;

cfg.endpointB.vidType = TRC_VIDTYPE_H263;
cfg.endpointB.profile = TRC_PROFILE_BASELINE;
cfg.endpointB.level = TRC_H263_LEVEL_10;
cfg.endpointB.dataRate = 43;                          /* kbits/sec                   */
```

```
cfg.endpointB.frameRate = 7;                          /* frames/sec                    */
cfg.endpointB.packetizeMode = TRC_PACKETIZE_2429;
cfg.endpointB.chanIn.jitterMode = TRC_JITTER_STATIC;  /* use jitter                    */
cfg.endpointB.chanIn.jitterLatency = 400;             /* milliseconds                  */
strcpy( cfg.endpointB.chanOut.ipAddr, "192.68.2.2" );
cfg.endpointB.chanOut.rtpPort = 2000;
cfg.endpointB.chanOut.payloadID = 97;
cfg.endpointB.chanOut.tos = 0;

/* configure the MPEG-4 decoder that is transcoding from MPEG-4 endpoint A
to H.263 endpoint B                                                                    */
cfg.decoder[TRC_DIR_FDX1].optData = mpeg4DecoderCfg;
cfg.decoder[TRC_DIR_FDX1].optSize = sizeof(mpeg4DecoderCfg);

/* start the full-duplex channel                                                       */
result = trcStartVideoChannel( trcChHandle, &cfg );
if (result == TRC_SUCCESS)
{
   printf( "Start full-duplex channel request in progress\n" );
}
    else
{
    printf( "Start full-duplex channel request failed [%s]\n",
    trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when the start completes:

```
/*****************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*****************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
    printf( "TRC event [%s]: result [%s]\n', eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_START_CHANNEL_DONE:
            printf( "Channel start done [userKey 0x%X]\n",
                    pMsg->userKey );
            break;
    }
return( 0 ); /* always return 0 (successfully received event)         */
}
```

The following example shows a simplex channel from H.263 to MPEG-4:

```
/* start a simplex channel from H.263 endpoint (A) to MPEG-4 endpoint (B)            */
cfg.endpointA.vidType = TRC_VIDTYPE_H263;
cfg.endpointA.profile = TRC_PROFILE_BASELINE;
cfg.endpointA.level = TRC_H263_LEVEL_10;
cfg.endpointA.dataRate = 43;                         /* kbits/sec                    */
cfg.endpointA.frameRate = 7;                         /* frames/sec                   */
cfg.endpointA.packetizeMode = TRC_PACKETIZE_2429;
cfg.endpointA.chanIn.jitterMode = TRC_JITTER_STATIC;  /* use jitter                  */
cfg.endpointA.chanIn.jitterLatency = 400;            /* milliseconds                 */
/* endpoint A channel output configuration not required for simplex channel          */

cfg.endpointB.vidType = TRC_VIDTYPE_MPEG4;
cfg.endpointB.profile = TRC_PROFILE_SIMPLE;
cfg.endpointB.level = TRC_MPEG4_LEVEL_0;
cfg.endpointB.dataRate = 43;                         /* kbits/sec                    */
cfg.endpointB.frameRate = 7;                         /* frames/sec                   */
cfg.endpointB.packetizeMode = TRC_PACKETIZE_2429;
/* endpoint B channel input configuration not required for simplex channel           */

strcpy( cfg.endpointB.chanOut.ipAddr, "192.68.2.2" );
cfg.endpointB.chanOut.rtpPort = 2000;
cfg.endpointB.chanOut.payloadID = 100;
cfg.endpointB.chanOut.tos = 0;

/* start the simplex channel                                                          */
result = trcStartVideoChannel( trcChHandle, &cfg );
if (result == TRC_SUCCESS)
    {
        printf( "Start simplex channel request in progress\n" );
    }
    else
    {
      printf( "Start simplex channel request failed [%s]\n",
              trcValueName( TRCVALUE_RESULT, result ) );
    }

/* NOTE: See full-duplex example for trc_callback handling TRCEVN_START_CHANNEL_DONE   */
```

trcStopOverlay

Stops an active overlay.

Prototype

U32 **trcStopOverlay** (TRC_HANDLE ***trcChHandle***, TRC_OVL_HANDLE ***ovlHandle***)

Argument	Description
trcChHandle	Handle to a video transcoding channel created by **trcCreateVideoChannel**.
ovlHandle	Overlay handle created by **trcCreateOverlay**.

Return values

Return value	Description
TRCERR_INVALID_CHANNEL_HANDLE	Channel handle is not valid.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_OUT_OF_MEMORY	Cannot allocate enough memory to send request.
TRCERR_RING_FULL	Ring buffer is full.
TRC_SUCCESS	Overlay stop request was successfully issued. The application receives a TRCEVN_STOP_CHANNEL_OVL_DONE event when the request completes.
TRC_OVLEVT_TRCP_INVALID_OVL_HANDLE	Overlay handle is not valid.

Events

Event	Description
TRCEVN_STOP_OVL_DONE	Indicates the overlay is no longer being displayed. data[TRCDATA_OVERLAY_USERKEY] Contains the overlay user key of the destroyed overlay.

Details

All related resources remain available so the overlay can later be restarted using **trcStartOverlay**.

Examples

The following example shows how to stop an active overlay:

```
result = trcStopOverlay( chHandle, ovlHandle);
if (result == TRC_SUCCESS)
```

```
{
    printf( "trcStopOverlay() request in progress\n" );
}
else
{
    printf( "Unexpected result from trcStopOverlay() = 0x%08x [%s]\n",
            result,
            trcValueName(TRCVALUE_RESULT, result));
}
return( result );
```

The following example shows how to handle the callback event that occurs when the stop overlay completes:

```
/******************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
******************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
    S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
    S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );

    printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
    switch( pMsg->event )
    {
        case TRCEVN_STOP_OVL_DONE:
            if (pMsg->result == TRC_SUCCESS)
            {
                printf( "Success: Overlay userKey=%p \n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY] );
                printf( "Stopped on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
            else
            {
                printf( "Failure: Overlay userKey=%p\n",
                        pMsg->data[TRCDATA_OVERLAY_USERKEY] );
                printf( "could not be stopped on channel userKey=%p handle=%p\n",
                        pMsg->userKey,
                        pMsg->trcChHandle );
            }
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcStopVideoChannel

Stops transcoding on a specified video channel.

Prototype

U32 **trcStopVideoChannel** (TRC_HANDLE ***trcChHandle***)

Argument	Description
trcChHandle	Valid channel handle returned from **trcCreateVideoChannel**.

Return values

Return value	Description
TRC_SUCCESS	Stop request was successfully issued. The application receives a TRCEVN_STOP_CHANNEL_DONE event when the request completes.

Return value	Description
TRCERR_INVALID_CHANNEL_HANDLE	Identified channel does not exist.
TRCERR_INVALID_CHANNEL_STATE	Channel is not started.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. **trcShutdown** may have been called.
TRCERR_SOCKET_FAILURE	Unable to send the request over communication socket to the video transcoder platform (connection error).

Events

Event	Description
TRCEVN_STOP_CHANNEL_DONE	Channel stop request is complete. The event results are: TRC_SUCCESS Channel was successfully stopped. TRCERR_INVALID_CHANNEL_STATE Channel state on the video transcoder platform is not in a valid state for a channel stop. Call **trcDestroyVideoChannel** to destroy the channel.

Details

Use **trcStopVideoChannel** to stop all transcoding over the specified channel. For full-duplex channels, transcoding is stopped in both directions.

This function sends a stop channel message to the transcoder and returns to the application without waiting for an acknowledgement from the transcoder. When the TRC receives the acknowledgement, it calls the callback function provided in **trcInitialize** with the TRCEVN_STOP_CHANNEL_DONE event. The result field indicates the success or failure of the channel stop.

Once a channel is stopped, the channel can be used for a new transcoder channel of the same channel type (simplex or full-duplex) and requiring the same optional features (overlay, RTCP support, or both). Always reinitialize the channel configuration structure to all defaults before defining a new configuration.

See also

trcStartVideoChannel

Examples

The following example shows stopping a started channel:

```
result = trcStopVideoChannel( trcChHandle );
if (result == TRC_SUCCESS)
{
    printf( "Channel stop in progress\n" );
}
else
```

```
{
    printf( "Channel stop failed [%s]\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

The following example shows how to handle the callback event that occurs when the stop completes:

```
/*****************************************************************************
* trc_callback - upcall used by TRC thread to pass asynchronous events
*
* WARNING: This function is called as part of the TRC thread.
*          A thread-safe mechanism must be used when handling events.
*
* inputs: pMsg - pointer to message being received from TRC
*         size - byte length of message
*
* output: always 0
*****************************************************************************/
U32 trc_callback( tTrcMessage *pMsg, U32 size )
{
   S8 *eventName = trcValueName( TRCVALUE_EVENT, pMsg->event );
   S8 *resultName = trcValueName( TRCVALUE_RESULT, pMsg->result );
   printf( "TRC event [%s]: result [%s]\n", eventName, resultName );
   switch( pMsg->event )
   {
        case TRCEVN_STOP_CHANNEL_DONE:
            printf( "Channel stop done [userKey 0x%X]\n",
                    pMsg->userKey );
            break;
    }
    return( 0 ); /* always return 0 (successfully received event) */
}
```

trcUsage

Obtains information from the TRC regarding overall channel usage.

Prototype

U32 **trcUsage** (tTrcUsage ***_usage_**)

Argument	Description
usage	Pointer to the tTrcUsage structure that provides usage summary information. For more information, refer to *tTrcUsage* on page 252.

Return values

Return value	Description
TRC_SUCCESS	Usage information was provided successfully.
TRCERR_INVALID_CHANNEL_PARAM	Invalid address was provided as the return structure address.
TRCERR_LIB_NOT_INITIALIZED	TRC library was not initialized. Call **trcInitialize** first.

Events

None

Details

The controlling application can call **trcUsage** at any time after calling **trcInitialize** to obtain current channel usage information. This information is not required for any TRC control but is provided so that the application can monitor overall resource usage, if desired.

Example

```
result = trcUsage( &trcUsage );
if (result == TRC_SUCCESS)
{
       printf( "%d licenses available\n", trcUsage.licensesAvail );
       printf( "Local application controlling %d simplex and %d full-duplex channels\n",
               trcUsage.simplexLocal, trcUsage.fdxLocal );
       printf( "Total channels in use: %d simplex and %d full-duplex channels\n",
                trcUsage.simplexTotal, trcUsage.fdxTotal );
}
else
{
       printf( "Error [%s] while requesting channel usage information\n",
               trcValueName( TRCVALUE_RESULT, result ) );
}
```

trcValueName

Provides the application with an ASCII string that corresponds to a variety of numeric values used by the TRC.

Prototype

S8 ***trcValueName*** (U32 ***valueType***, U32 ***value***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>valueType</td><td>Type of value for which to provide an ASCII string. Valid values include:
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>TRCVALUE_BIT</td><td>ASCII name of a bit value (TRC_TRUE or TRC_FALSE).</td></tr>
<tr><td>TRCVALUE_CHSTATE</td><td>ASCII name of the channel state value.</td></tr>
<tr><td>TRCVALUE_DIRECTION</td><td>ASCII name of direction indicators used by a variety of TRC functions.</td></tr>
<tr><td>TRCVALUE_EVENT</td><td>ASCII name of any TRC event code.</td></tr>
<tr><td>TRCVALUE_FRAMERES</td><td>ASCII name of the frame resolution (CIF or QCIF).</td></tr>
<tr><td>TRCVALUE_JITTER</td><td>ASCII name of the jitter mode (NONE or STATIC).</td></tr>
<tr><td>TRCVALUE_LEVEL</td><td>Profile level represented as ASCII string.</td></tr>
<tr><td>TRCVALUE_PACKETIZE</td><td>ASCII name of packetization mode (3016, 2190, 2429).</td></tr>
<tr><td>TRCVALUE_PROFILE</td><td>ASCII name of profile type (SIMPLE or BASELINE).</td></tr>
<tr><td>TRCVALUE_RESULT</td><td>ASCII name of any TRC result code, including TRC_SUCCESS and all TRCERR_xxx values.</td></tr>
<tr><td>TRCVALUE_TRACE</td><td>ASCII name of the trace type bit provided in value.</td></tr>
<tr><td>TRCVALUE_TYPE</td><td>ASCII name of the channel type (SIMPLEX or FDX).</td></tr>
<tr><td>TRCVALUE_VTPSTATE</td><td>ASCII name of the video transcoder platform state value.</td></tr>
<tr><td>TRCVALUE_VIDTYPE</td><td>ASCII name of the video type (MPEG-4 or H.263).</td></tr>
</table></td></tr>
<tr><td>value</td><td>Numeric value for which to provide an ASCII string equivalent.</td></tr>
</table>

Return values

The function always returns an ASCII string:

- In the case where an invalid ***valueType*** is provided, the function returns the string UNKNOWN VALUE TYPE [0***xXXXXXXXX***] where ***XXXXXXXX*** is the hexadecimal representation of the ***valueType*** provided.
- In the case where the ***valueType*** is valid but the value is out of range for the given type, the function returns the string INVALID VALUE [0***xXXXXXXXX***] FOR ***sssss*** where ***XXXXXXXX*** is the hexadecimal representation of the value provided and ***sssss*** is the string representation of the ***valueType***.

The TRC uses a single global area to format error indication strings. A subsequent call to **trcValueName** can overwrite the previous error string text if another invalid ***valueType***/***value*** is specified.

Events

None.

Details

The controlling application can call **trcValueName** to obtain an ASCII string representation of the desired value. This function is provided as an aid to creating diagnostic messages.

Example

```
valueName = trcValueName( TRCVALUE_RESULT, result );

printf( "TRC result code [0x%08X] = %s\n", result, valueName );
```

trcVTPStatus

Allows the application to obtain information from the TRC regarding the overall status of all video transcoder platform connections.

Prototype

U32 **trcVTPStatus**(tTrcVtpAll ****vtpStatus***)

Argument	Description
vtpStatus	Pointer to the tTrcVtpAll structure, which provides a summary view of all video transcoder platforms currently in use by the TRC.

Return values

Return value	Description
TRC_SUCCESS	Status information was provided successfully.
TRCERR_INVALID_CHANNEL_PARAM	Invalid address was provided as the return structure address.
TRCERR_LIB_NOT_INITIALIZED	TRC library was not initialized. Call **trcInitialize** first.

Events

None.

Details

The controlling application can call **trcVTPStatus** at any time after calling **trcInitialize** to obtain current video transcoder platform connection status information. This information is not required for any TRC control but is provided so that the application can monitor overall resource connectivity if desired.

After a successful call to **trcVTPStatus**, the TRC returns VTP connection status information in the tTrcVtpAll structure and its substructures. For more information, refer to *tTrcVtpAll* on page 253.

Example

```
result = trcVTPStatus( &trcVTPStatus );
if (result == TRC_SUCCESS)
{
    printf( "%d VTPs defined\n", trcVTPStatus.vtpDefined );
    for (i = 0; i < trcVTPStatus.vtpDefined; i++)
    {
        printf( "VTP %d: state [%s]\n", trcVTPStatus.vtp[i].vtpId,
                trcValueName( TRCVALUE_VTPSTATE, trcVTPStatus.vtp[i].state ) );
    }
}
else
{
    printf( "Error [%s] while requesting VTP status information\n",
            trcValueName( TRCVALUE_RESULT, result ) );
}
```

9. Management functions

Using the management function reference

This section provides an alphabetical reference to the management interface functions. A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. If a function argument is a structure, the complete structure is shown.
Return values	The return value for a function is either VS_SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; a subsequent event indicates the completion status of the operation. Refer to the *Management error summary* on page 311 for a list of errors that the management interface functions return.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. Additional information such as reason codes and return values appears in the value field of the event. If there are no events listed, the function is synchronous. For more information, refer to *Management events* on page 315.

Standard mode versus raw mode

Management functions can be run in standard or raw mode. They are most often run in standard mode. The following table provides a description of these modes:

Mode	Description
Standard	Uses all of the VTMNG API functions. The application calls the various functions that format and then issue requests. The VTMNG API handles all message formatting and all UDP port handling. This mode is recommended because many of the jobs related to message-based communication are handled by the VTMNG API.

Mode	Description
Raw	The customer application is responsible for: Attaching to UDP ports. Formatting requests. Sending requests. Receiving all responses and notification messages. This mode only uses the two VTMNG API message conversion functions (**vtMngMsg2Host** and **vtMngMsg2Network**) and the function that provides the byte length of a given message (**vtMngMsgSize**). Use this mode to integrate sending and receiving of management messages into any management application that is in operation at your site. The raw mode allows you to develop a management application that is triggered by operator actions other than keyboard input.

vtMngEventApp

Issues a request to perform an application-level event.

This function is defined but no application-level events exist at this time. This function should be considered reserved for future use.

Prototype

U32 **vtMngEventApp**(VTMNG_VTPADDR ****vtpAddr***, U32 ***appUnique***, U32 ***appEvent***)

Argument	Description
vtpAddr	Video transcoder platform address.
appUnique	Application ID value identifying the application control entity to be evented. The application ID is unique across all applications connected to the video transcoder platform. All valid application ID values can be obtained using the **vtMngGetAppList** function.
appEvent	Application-level event to be issued.

Return values

Return value	Description
VTMNG_ERR_NOT_SUPPORTED	There are no supported application-level events defined at this time.

Events

Event	Description
vtEventAppRsp(upcall)	When the VTMNG API receives the response to the event application request, the API will upcall the owner's event application response function (if an upcall was provided to the **vtMngInit** function).

Details

Reserved for future use.

Example

Reserved for future use.

vtMngEventChn

Issues a request to perform a channel-level event.

Prototype

U32 **vtMngEventChn**(VTMNG_VTPADDR ****vtpAddr***, U32 ***chnUnique***, U32 ***chnEvent***)

Argument	Description
vtpAddr	Video transcoder platform address.
chnUnique	Channel ID value identifying the channel to be evented. The channel ID is unique across all channels connected to the video transcoder platform. All valid channel ID values can be obtained using the **vtMngGetChnList** function.
chnEvent	Channel-level event to be issued: VTMNG_CHN_E_ABORT Causes the channel to be aborted. The channel is immediately stopped and destroyed with resources returned to the free pool.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Channel identified by *chnUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtEventChnRsp(upcall)	When the VTMNG API receives the response to the event channel request, the API will upcall the owner's event channel response function (if an upcall was provided to the **vtMngInit** function).

Details

Use this function to terminate a channel from outside of the controlling application.

Example

```
        U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngEventChn( &dest, chnUnique, VTMNG_CHN_E_ABORT );
. . .
>>> VTMNG API receives response and upcalls the application's vtEventChnRsp function:
void myMngEventChnRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                       VT_MNG_MSG *msg, U32 result,
                       U32 chnUnique, U32 chnEvent,
                       VTMNG_CHN_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "Channel ID 0x%08X evented successfully\n", chnUnique );
    }
    else
    {
        printf( "Error 0x%08X while eventing channel ID 0x%08X\n", result, chnUnique );
    }
}
```

vtMngEventMon

Issue a request to perform a process monitor event.

Prototype

U32 **vtMngEventMon**(VTMNG_VTPADDR ****vtpAddr***, U32 ***monUnique***, U32 ***monEvent***)

Argument	Description
vtpAddr	Video transcoder platform address.
monUnique	Monitored process ID value identifying the process to be evented. The monitor ID is unique across all monitored processes on the video transcoder platform. All valid monitored process ID values can be obtained using the **vtMngGetMonList** function.
monEvent	Monitored process event to be issued: VTMNG_MON_E_TERMINATE Terminate the process, allowing auto-recovery to recover the process. VTMNG_MON_E_STOP Cause the process to stop executing and to not automatically restart the process. VTMNG_MON_E_START Start up a process that was stopped.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Process identified by *monUnique* does not exist.

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtEventMonRsp(upcall)	When the VTMNG API receives the response to the event Monitor Process request, the API will upcall the owner's event monitored process response function, if an upcall was provided to the **vtMngInit** function.

Details

Use the **vtMngEventMon** function to affect a specific monitored transcoder process. For example, if you suspect a process is corrupted, you can issue a TERMINATE event to trigger automatic recovery by the *vtmon* process monitor.

To replace a process executable, you must stop the process without triggering automatic recovery. Use the STOP event to stop a process without restarting it. This should normally only be performed after a video transcoder platform is disabled so that the stop process does not affect service. Once the process executable is updated, use the START event to restore the process back into service. You can then enable the video transcoder platform to bring the system back into service.

Example

```
    U32               result;
    VTMNG_VTPADDR   dest;            /* destination addressing information */
    U32               myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngEventMon( &dest, monUnique, VTMNG_MON_E_STOP );
. . .
>>> VTMNG API receives response and upcalls the application's vtEventMonRsp function:
void myMngEventMonRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                       VT_MNG_MSG *msg, U32 result,
                       U32 monUnique, U32 monEvent,
                       VTMNG_MON_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "Monitored Process ID 0x%08X evented successfully\n", monUnique );
    }
    else
    {
        printf( "Error 0x%08X while eventing Monitored Process ID 0x%08X\n",
                result, monUnique );
    }
}
```

vtMngEventVtp

Issues a request to perform a video transcoder platform-level event.

Prototype

U32 **vtMngEventVtp**(VTMNG_VTPADDR ****vtpAddr***, U32 ***vtpEvent***)

Argument	Description
vtpAddr	Video transcoder platform address.
vtpEvent	Video transcoder platform level event to be issued. VTMNG_VTP_E_ENABLE Enable assignment of channels to this video transcoder platform. VTMNG_VTP_E_DISABLE Do not allow new channel assignments. Once idle, go to a disabled state. VTMNG_VTP_E_ABORT Terminate all channels. Once all channels are aborted, go to a disabled state. VTMNG_VTP_E_RESTART Stop all transcoder processes and then restart all processes. VTMNG_VTP_E_.REBOOT Cause the video transcoder platform to reboot.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.

Return value	Description
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtEventVtpRsp(upcall)	When the VTMNG API receives the response to the event video transcoder platform request, the API will upcall the owner's event video transcoder platform response function (if an upcall was provided to vtMngInit).

Details

Use the **vtMngEventVtp** function to change the overall state of an entire video transcoder platform. Normally, each video transcoder platform operates in the ENABLED state allowing TRC APIs to request video transcoder resources. Under certain instances, it may be necessary to cause all transcoder processes running on a given video transcoder platform to stop and then be restarted. Issue the RESTART event to cause a warm start of the video transcoder platform. Issue the REBOOT event to cause the entire video transcoder platform to go through a cold start (complete reboot).

You can also disable a video transcoder platform. This means that the video transcoder platform will allow all current channels to continue but no new channels will be assigned to the video transcoder platform. Once the last channel is destroyed, the video transcoder platform enters the disabled state (allowing maintenance operations to be performed only after the video transcoder platform is taken out of service). Issue the ENABLE event to cause a disabled video transcoder platform to return to service. To cause a video transcoder platform to abort all current channels and immediately become disabled, issue the ABORT event.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngEventVtp( &dest, VTMNG_VTP_E_RESTART );
. . .
>>> VTMNG API receives response and upcalls the application's vtEventVtpRsp function:
void myMngEventVtpRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                       VT_MNG_MSG *msg, U32 result,
                       U32 vtpEvent,
                       VTMNG_VTP_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "VTP evented successfully\n" );
    }
    else
    {
        printf( "Error 0x%08X while eventing VTP\n", result );
    }
}
```

vtMngGetApp

Issues a request for details of a particular application.

Prototype

U32 **vtMngGetApp**(VTMNG_VTPADDR ****vtpAddr***, U32 ***appUnique***)

Argument	Description
vtpAddr	Video transcoder platform address.
appUnique	Application ID value identifying the application to be queried. The application ID is unique across all applications connected to the video transcoder platform. All valid application ID values can be obtained using the **vtMngGetAppList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Application identified by *appUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.

Return value	Description
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetAppRsp(upcall)	When the VTMNG API receives the response to the get application request, the API will upcall the owner's get application response function (if an upcall was provided to the **vtMngInit** function).

Details

Use this request to view information maintained by the video transcoder in relation to a particular control application.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;           /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetApp( &dest, appUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetAppRsp function:
void myMngGetAppRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     U32 appUnique,
                     VTMNG_APP_CFG *cfg,
                     VTMNG_APP_STATUS *status,
                     VTMNG_APP_STATS *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "Application ID 0x%08X information obtained successfully\n",
                appUnique );
    }
    else
    {
        printf( "Error 0x%08X while querying application ID 0x%08X\n",
                result, appUnique );
    }
}
```

vtMngGetAppList

Issues a request for the list of connected applications.

Prototype

U32 **vtMngGetAppList**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetAppListRsp (upcall)	When the VTMNG API receives the response to the get application list request, the API will upcall the owner's get application list response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetAppList** function to obtain a list of all controlling applications that are currently connected to the given video transcoder platform. Once the list of applications has been obtained, perform other requests to any given unique application ID (appUnique) listed in the response.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    U32             i;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetAppList( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetAppListRsp function:
void myMngGetAppListRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                         VT_MNG_MSG *msg, U32 result,
                         VTMNG_ENT_ID *app,
                         U32 appCount )
{
    if (result == VS_SUCCESS)
    {
        printf( "List of %u current applications obtained successfully\n", appCount );
        for (i = 0; i < appCount; i++)
        {
            printf( "  Application ID 0x%08X - name=%s\n",
                    app->entObj.entUnique, app->entName );
            app++;
        }
    }
    else
    {
        printf( "Error %s while querying application list\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngGetChn

Issues a request for details about a particular channel.

Prototype

U32 **vtMngGetChn**(VTMNG_VTPADDR ***vtpAddr***, U32 ***chnUnique***)

Argument	Description
vtpAddr	Video transcoder platform address.
chnUnique	Channel ID value identifying the channel to be queried. The channel ID is unique across all channels connected to the video transcoder platform. All valid channel ID values can be obtained using the **vtMngGetChnList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Channel identified by *chnUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetChnRsp (upcall)	When the VTMNG API receives the response to the get channel request, the API will upcall the owner's get channel response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetChn** function to obtain configuration, status, and statistics information for a given video transcoder channel.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetChn( &dest, chnUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetChnRsp function:
void myMngGetChnRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     U32 chnUnique,
                     VTMNG_CHN_CFG *cfg,
                     VTMNG_CHN_STATUS *status,
                     VTMNG_CHN_STATS *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "channel ID 0x%08X information obtained successfully\n",
                chnUnique );
    }
    else
    {
        printf( "Error %s while querying channel ID 0x%08X\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), chnUnique );
    }
}
```

vtMngGetChnList

Issues a request for the list of defined channels.

Prototype

U32 **vtMngGetChnList**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetChnListRsp(upcall)	When the VTMNG API receives the response to the get channel list request, the API will upcall the owner's get channel list response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetChnList** function to obtain a list of all video transcoder channels that are currently defined on the given video transcoder platform. Once the list of channels is obtained, perform other requests to any given unique channel ID (chnUnique) listed in the response.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    U32              i;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetChnList( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetChnListRsp function:
void myMngGetChnListRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     VTMNG_ENT_ID *chn,
                     U32 chnCount )
{
    if (result == VS_SUCCESS)
    {
        printf( "List of %u channels obtained successfully\n", chnCount );
        for (i = 0; i < chnCount; i++)
        {
            printf( "  Channel ID 0x%08X - name=%s\n",
                    chn->entObj.entUnique, chn->entName );
            chn++;
        }
    }
    else
    {
        printf( "Error %s while querying channel list\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngGetHistPerHHr

Issues a request for a per half-hour historical statistics record.

Prototype

U32 **vtMngGetHistPerHHr**(VTMNG_VTPADDR ****vtpAddr***, U32 ***halfHoursAgo***)

Argument	Description
vtpAddr	Video transcoder platform address.
halfHoursAgo	Number of half-hour intervals into the past history.

Return values

Return value	Description
VTMNG_ERR_INVALID_INDEX	Value provided as *halfHoursAgo* is beyond valid range (1..48).
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetHistPerHHrRsp (upcall)	When the VTMNG API receives the response to the get per-half-hour history request, the API will upcall the owner's get per-half-hour history response function (if an upcall was provided to **vtMngInit** function).

Details

Use the **vtMngGetHistPerHHr** function to obtain summary statistics for a particular half-hour interval over the span of the last 24 hour period. Statistical summary information is provided as a set of 48 per-half-hour buckets so that historical statistical analysis can be performed.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;           /* destination addressing information */
    U32              myKey = 12345;
    U32              i;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    for (i = 0; i < VTMNG_MAX_HHR_BUCKET; i++)
    {   /* request historgram entry for every half-hour bucket of last day */
        result = vtMngGetHistPerHHr( &dest, i );
        if (result == VS_SUCCESS)
        {
            globalExpectedRspCount++;
        }
        else
        {
            printf( "Error %s requesting histogram stats from %u half-hours ago\n",
                    vtMngValueName( VTMNG_VALUE_RESULT, result ), i );
        }
    }
. . .
>>> VTMNG API receives response and upcalls the application's vtGetHistPerHHrRsp:
void myMngGetHistPerHHrRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                            VT_MNG_MSG *msg, U32 result,
                            U32 halfHoursAgo,
                            VTMNG_ST_ENTRY *stats )
{
    if (result == VS_SUCCESS)
    {
        /* store away this response and wait until all have been received */
    }
    else
    {
        printf( "Error %s while querying stats from %u half-hours ago\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), halfHoursAgo );
    }
   globalExpectedRspCount--;
    if (globalExpectedRspCount == 0)
    {   /* have received all per-half-hour histogram entries */
        /* here, use the stored histogram buckets to produce reports, etc. */
        printf( "All per-half-hour histogram buckets have been collected.\n" );
    }
}
```

vtMngGetHistPerMin

Issues a request for a per-minute historical statistics record.

Prototype

U32 **vtMngGetHistPerMin**(VTMNG_VTPADDR ****vtpAddr***, U32 ***minutesAgo***)

Argument	Description
vtpAddr	Video transcoder platform address.
minutesAgo	Number of minutes into the past history.

Return values

Return value	Description
VTMNG_ERR_INVALID_INDEX	Value provided as *minutesAgo* is beyond valid range (1..60).

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetHistPerMinRsp (upcall)	When the VTMNG API receives the response to the get per-minute history request, the API will upcall the owner's get per-minute history response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetHistPerMin** function to obtain summary statistics for a particular minute interval over the span of the last one hour period. Statistical summary information is provided as a set of 60 per-minute buckets so that historical statistical analysis can be performed.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    U32             i;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    for (i = 0; i < VTMNG_MAX_MIN_BUCKET; i++)
    {   /* request historgram entry for every per-minute bucket of last hour */
        result = vtMngGetHistPerMin( &dest, i );
        if (result == VS_SUCCESS)
        {
            globalExpectedRspCount++;
        }
        else
        {
            printf( "Error %s requesting histogram stats from %u minutes ago\n",
                    vtMngValueName( VTMNG_VALUE_RESULT, result ), minutesAgo );
        }
    }
. . .
>>> VTMNG API receives response and upcalls the application's vtGetHistPerMinRsp:
void myMngGetHistPerMinRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                            VT_MNG_MSG *msg, U32 result,
                            U32 minutesAgo,
                            VTMNG_ST_ENTRY *stats )
{
    if (result == VS_SUCCESS)
    {
        /* store away this response and wait until all have been received */
    }
    else
    {
        printf( "Error 0x%08X while querying stats from %u minutes ago\n",
                result, minutesAgo );
    }
    globalExpectedRspCount--;
    if (globalExpectedRspCount == 0)
    {   /* have received all per-minute histogram entries */
        /* here, use the stored histogram buckets to produce reports, etc. */
        printf( "All per-minute histogram buckets have been collected.\n" );
    }
}
```

vtMngGetMon

Issues a request for details about a particular monitored process.

Prototype

U32 **vtMngGetMon**(VTMNG_VTPADDR ****vtpAddr***, U32 ***monUnique***);

Argument	Description
vtpAddr	Video transcoder platform address.
monUnique	Monitored process ID value identifying the process to be queried. The monitor ID is unique across all monitored processes on the video transcoder platform. All valid monitored process ID values can be obtained using the **vtMngGetMonList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Process identified by *monUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetmonRsp (upcall)	When the VTMNG API receives the response to the get monitored process request, the API will upcall the owner's get monitored process response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetMon** function to obtain configuration, status and statistics information for a given monitored process.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetMon( &dest, monUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetMonRsp function:
void myMngGetMonRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     U32 monUnique,
                     VTMNG_MON_CFG *cfg,
                     VTMNG_MON_STATUS *status,
                     VTMNG_MON_STATS *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "Monitored Process ID %s information obtained successfully\n",
                monUnique );
    }
    else
    {
        printf( "Error 0x%08X while querying monitored process ID 0x%08X\n",
                result, monUnique );
    }
}
```

vtMngGetMonList

Issues a request for the list of monitored processes.

Prototype

U32 **vtMngGetMonList**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetMonListRsp(upcall)	When the VTMNG API receives the response to the get monitored process list request, the API will upcall the owner's get monitored process list response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetMonList** function to obtain a list of all transcoder processes that are being monitored on the given video transcoder platform. Once the list of monitored processes is obtained, perform other requests to any given unique monitored process ID (***monUnique***) listed in the response.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;             /* destination addressing information */
    U32              myKey = 12345;
    U32              i;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetMonList( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetMonListRsp function:
void myMngGetMonListRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                         VT_MNG_MSG *msg, U32 result,
                         VTMNG_ENT_ID *mon,
                         U32 monCount )
{
    if (result == VS_SUCCESS)
    {
        printf( "List of %u monitored processes obtained successfully\n", monCount );
        for (i = 0; i < monCount; i++)
        {
            printf( "  Monitored Process ID 0x%08X - name=%s\n",
                    mon->entObj.entUnique, mon->entName );
            mon++;
        }
    }
    else
    {
        printf( "Error %s while querying monitored process list\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngGetStCurrMin

Issues a statistics request for the current minute.

Prototype

U32 **vtMngGetStCurrMin**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetHistCurrMinRsp(upcall)	When the VTMNG API receives the response to the get current minute statistics request, the API will upcall the owner's get current minute statistics response function (if an upcall was provided to the vtMngInit function).

Details

Use the **vtMngGetStCurrMin** function to request the value of all summary-level statistics maintained for the current one-minute interval. Current minute statistics are normally not useful on their own since these values are automatically zeroed at each one-minute interval and only include values for channels that have been stopped during the current one-minute interval.

The values that are accumulated in the CurrMin statistics are moved into the proper per-minute bucket as the one-minute timer expires. It is through this mechanism that all histogram statistics information is maintained.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetStCurrMin( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetStCurrMinRsp:
void myMngGetStCurrMinRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                            VT_MNG_MSG *msg, U32 result,
                            VTMNG_ST_ENTRY *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "current minute statistics successfully obtained\n" );
    }
    else
    {
        printf( "Error %s while querying current minute statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngGetStTotal

Issues a request for the total (overall) statistics.

Prototype

U32 **vtMngGetStTotal**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetStTotalRsp(upcall)	When the VTMNG API receives the response to the get total statistics request, the API will upcall the owner's get total statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngGetStTotal** function to request the total of all summary-level statistics maintained by the given video transcoder platform. Total statistics are used to view accumulated values since the last time that the total statistics were zeroed.

Examples

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetStTotal( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetStTotalRsp:
void myMngGetStTotalRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                         VT_MNG_MSG *msg, U32 result,
                         VTMNG_ST_ENTRY *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "total statistics successfully obtained\n" );
    }
    else
    {
        printf( "Error %s while querying total statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngGetVtp

Issues a request for all video transcoder platform-level information.

Prototype

U32 **vtMngGetVtp**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtGetVtpRsp(upcall)	When the VTMNG API receives the response to the get VTP request, the API will upcall the owner's get VTP response function (if an upcall was provided to the vtMngInit function).

Details

Use the **vtMngGetVtp** function to obtain all video transcoder platform-level configuration, status and statistics information.

Examples

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngGetVtp( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtGetVtpRsp function:
void myMngGetVtpRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     VTMNG_VTP_CFG *cfg,
                     VTMNG_VTP_STATUS *status,
                     VTMNG_VTP_STATS *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "VTP-level information obtained successfully\n" );
    }
    else
    {
        printf( "Error %s while querying VTP-level information\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngInit

Initializes and activates the management communication interface.

Prototype

U32 **vtMngInit**(void ****userkey***, VTMNG_UPCALLS ****upcalls***, U32 ***reqPort***, U32 ***trapPort***, U32 ***dbgMask***, S8 ****initdest***, S8 ****eventlog***)

Argument	Description
userkey	User-controlled key to be provided on all notifications.
upcalls	Set of upcall functions (received responses|traps [*keyboard*]).
reqPort	UDP port number to use for issuing requests and receiving responses. VTMNG_PORT_DISABLED Do not register for any UDP port for response handling. VTMNG_PORT_SELECT Allow the operating system to select any available UDP port. This is the default for *reqPort*. else Register for the port number specified.

Argument	Description
trapPort	UDP port number to listen on for receiving traps. VTMNG_PORT_DISABLED Do not register for any UDP port for trap handling. This is the default for *trapPort*. else Register for the port number specified.
dbgMask	Mask of active trace bits. The following bits can be set to assist in management application development: BIT 0: Trace any errors encountered. BIT 1: Trace any warnings encountered. BIT 16: Trace asynchronous events to console. BIT 23: Trace management message send/receive to debug log.
initdest	Optional. Name of the initial destination. Used for internally generated management requests.
eventlog	Optional. File name used to log asynchronous events to disk.

Return values

Return value	Description
VTMNG_ERR_INVALID_STATE	VTMNG API in invalid state for initialization, for example, the VTMNG API has already been initialized. Perform a **vtMngShutdown** and then attempt the initialization again.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create VTMNG API control context.
VTMNG_ERR_NO_RESOURCE	Unable to obtain the required UDP resource or resources. This error usually indicates that another application is already listening on one or both of the UDP ports indicated.

Events

Once the VTMNG API is initialized, the calling application can be upcalled whenever the VTMNG API receives a management response message. If the trapPort is set to a non-zero value, then the VTMNG API also registers to receive asynchronous events (traps) and will upcall the management application whenever an asynchronous event occurs.

All of the upcall functions for the calling application are provided in the structure for the upcall. Any upcall function pointer that is set to NULL is handled by the internal handler code of the VTMNG API. Any non-NULL upcall address implies a call from the VTMNG API to the handler function of the calling application.

The following table provides a quick breakdown of each type of upcall event that could occur:

Event (upcall)	Description
vtGetAppListRsp	Handle receipt of response to get application list.
vtGetMonListRsp	Handle receipt of response to get monitored process list.
vtGetChnListRsp	Handle receipt of response to get channel list.
vtGetVtpRsp	Handle receipt of response to get video transcoder platform-level information.
vtGetAppRsp	Handle receipt of response to get information about a particular controlling application connection.
vtGetMonRsp	Handle receipt of response to get information about a particular monitored process.
vtGetChnRsp	Handle receipt of response to get information about a particular channel.
vtGetStTotalRsp	Handle receipt of response to get total statistics.
vtGetStCurrMinRsp	Handle receipt of response to get statistics for the current one-minute period.
vtGetHistPerMinRsp	Handle receipt of response to get a particular per-minute histogram statistics record.
vtGetHistPerHHrRsp	Handle receipt of response to get a particular per-half-hour histogram statistics record.
vtSetVtpRsp	Handle receipt of response to setting video transcoder platform-level information.
vtEventVtpRsp	Handle receipt of response to a video transcoder platform-level event request.
vtSetAppRsp	Handle receipt of response to setting the configuration for a particular application connection.
vtEventAppRsp	Handle receipt of response to an application-level event request.
vtSetMonRsp	Handle receipt of response to setting the configuration for a particular monitored process.
vtEventMonRsp	Handle receipt of response to a monitored process event request.
vtSetChnRsp	Handle receipt of response to set channel-level configuration.

Event (upcall)	Description
vtEventChnRsp	Handle receipt of response to a channel-level event request.
vtZeroVtpRsp	Handle receipt of response to zero video transcoder platform-level statistics.
vtZeroAppRsp	Handle receipt of response to zero the statistics for a particular application connection.
vtZeroMonRsp	Handle receipt of response to zero the statistics for a particular monitored process.
vtZeroChnRsp	Handle receipt of response to zero a particular channel's statistics.
vtZeroTotalRsp	Handle receipt of response to zero total statistics.
vtVtpLevelTrap	Handle receipt of asynchronous event (trap) indicating that a VTP-level threshold was crossed.
vtVtpErrorTrap	Handle receipt of asynchronous event (trap) indicating that a VTP-level error was detected.
vtAppConnTrap	Handle receipt of asynchronous event (trap) indicating that an application connection state has changed.
vtMonProcTrap	Handle receipt of asynchronous event (trap) indicating that a monitored process was either lost or recovered.
vtChnCreateTrap	Handle receipt of asynchronous event (trap) indicating that a new channel was created.
vtChnStartTrap	Handle receipt of asynchronous event (trap) indicating that a channel was started.
vtChnStopTrap	Handle receipt of asynchronous event (trap) indicating that a channel was stopped.
vtChnDeadTrap	Handle receipt of asynchronous event (trap) indicating that a channel was destroyed.
vtChnErrorTrap	Handle receipt of asynchronous event (trap) indicating that a channel-level error was detected.
vtCmdNotif	Handle indication that an operator command was received through the keyboard.

Details

Use the **vtMngInit** function to initialize the VTMNG API, providing a set of upcall functions that are used by the VTMNG API to handle all received management messages. Each upcall function can be provided by the calling application or set to NULL. The NULL setting indicates that the VTMNG API should handle the given response (or trap) internally.

Example

```
/* start with all upcalls handled internally (by VTMNG API) */
 memset( (void *)&upcalls, 0, sizeof(VTMNG_UPCALLS) );
 if (regResponses)
 {   /* register to handle all received management responses */
     upcalls.vtGetAppListRsp     = vtMgrGetAppListRsp;
     upcalls.vtGetMonListRsp     = vtMgrGetMonListRsp;
     upcalls.vtGetChnListRsp     = vtMgrGetChnListRsp;
     upcalls.vtGetVtpRsp         = vtMgrGetVtpRsp;
     upcalls.vtGetAppRsp         = vtMgrGetAppRsp;
     upcalls.vtGetMonRsp         = vtMgrGetMonRsp;
     upcalls.vtGetChnRsp         = vtMgrGetChnRsp;
     upcalls.vtGetStTotalRsp     = vtMgrGetStTotalRsp;
     upcalls.vtGetStCurrMinRsp   = vtMgrGetStCurrMinRsp;
     upcalls.vtGetHistPerMinRsp  = vtMgrGetHistPerMinRsp;
     upcalls.vtGetHistPerHHrRsp  = vtMgrGetHistPerHHrRsp;
     upcalls.vtSetVtpRsp         = vtMgrSetVtpRsp;
     upcalls.vtEventVtpRsp       = vtMgrEventVtpRsp;
     upcalls.vtSetAppRsp         = vtMgrSetAppRsp;
     upcalls.vtEventAppRsp       = vtMgrEventAppRsp;
     upcalls.vtSetMonRsp         = vtMgrSetMonRsp;
     upcalls.vtEventMonRsp       = vtMgrEventMonRsp;
     upcalls.vtSetChnRsp         = vtMgrSetChnRsp;
     upcalls.vtEventChnRsp       = vtMgrEventChnRsp;
     upcalls.vtZeroVtpRsp        = vtMgrZeroVtpRsp;
     upcalls.vtZeroAppRsp        = vtMgrZeroAppRsp;
     upcalls.vtZeroMonRsp        = vtMgrZeroMonRsp;
     upcalls.vtZeroChnRsp        = vtMgrZeroChnRsp;
     upcalls.vtZeroTotalRsp      = vtMgrZeroTotalRsp;
 }
 if (regTraps)
 {   /* register to handle all received management TRAPs */
     upcalls.vtVtpLevelTrap      = vtMgrVtpLevelTrap;
     upcalls.vtVtpErrorTrap      = vtMgrVtpErrorTrap;
     upcalls.vtAppConnTrap       = vtMgrAppConnTrap;
     upcalls.vtMonProcTrap       = vtMgrMonProcTrap;
     upcalls.vtChnCreateTrap     = vtMgrChnCreateTrap;
     upcalls.vtChnStartTrap      = vtMgrChnStartTrap;
     upcalls.vtChnStopTrap       = vtMgrChnStopTrap;
     upcalls.vtChnDeadTrap       = vtMgrChnDeadTrap;
     upcalls.vtChnErrorTrap      = vtMgrChnErrorTrap;
 }
 if (regKeyboard)
 {   /* register to handle all received operator commands (via keyboard) */
     upcalls.vtCmdNotif          = vtMgrCmdNotif;
 }
 result = vtMngInit( (void *)&myInfo,    /* user-controlled key provided on notifs */
                     &upcalls,           /* set of upcall functions */
                     reqPort,            /* UDP port for requests and responses */
                     trapPort,           /* UDP port to listen on for traps */
                     dbgMask,            /* mask of active trace bits */
                     vtpAddress,         /* OPTIONAL name of initial destination */
                     eventLog );         /* OPTIONAL filename to log events to */
 if (result != VS_SUCCESS)
 {
     printf( "!!! ERROR: %s returned from vtMngInit.\n",
             vtMngValueName( VTMNG_VALUE_RESULT, result ) );
 }
```

vtMngMsg2Host

Converts message from network-byte order to host-byte order.

Prototype

U32 **vtMngMsg2Host**(VT_MNG_MSG ****msg***, U32 ***len***)

Argument	Description
msg	Message to be converted.
len	Total byte length of message to be converted.

Return values

Return value	Description
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

None.

Details

The **vtMngMsg2Host** function is used internally by the VTMNG API to perform all message field swapping required to convert a message from network byte order to local host byte order. This function is normally not called directly by the calling application.

If an application is operating in raw mode (formatting all management messages and handling receipt of all responses and traps directly), then the **vtMngMsg2Host** function is called when any management message is received. This frees the application from having to perform any field-by-field numeric representation conversions.

Raw VTMNG API functions can be called even when the VTMNG API has not been initialized. This is because in raw mode, the controlling application manages all UDP communications.

Example

```
/* any time a raw UDP packet (management message) is received from a VTP */
result = vtMngMsg2Host( vtmsg, msgLen );
if (result != VS_SUCCESS)
{
    printf( "Error %s converting VTMNG message to local host representation.\n",
            vtMngValueName( VTMNG_VALUE_RESULT, result ) );
}
```

vtMngMsg2Network

Converts message from host-byte order to network-byte order.

Prototype

U32 **vtMngMsg2Network**(VT_MNG_MSG ***msg**, U32 ***len***)

Argument	Description
msg	Message to be converted
len	Total byte length of message to be converted.

Return values

Return value	Description
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

None.

Details

The **vtMngMsg2Network** function is used internally by the VTMNG API as needed to perform all message field swapping required to convert a message from local host byte order to network byte order. This function is normally not called directly by the calling application.

If an application is operating in raw mode (formatting all management messages and handling receipt of all responses and traps directly) then the **vtMngMsg2Network** function is called when any management request message is to be sent. This frees the application from having to perform any field-by-field numeric representation conversions.

Raw VTMNG API functions can be called even when the VTMNG API has not been initialized. This is because in raw mode, the controlling application manages all UDP communications.

Examples

```
/* any time a raw UDP packet (management message) is to be sent to a VTP */
result = vtMngMsg2Network( vtmsg, msgLen );
if (result != VS_SUCCESS)
{
    printf( "Error %s converting VTMNG message to network representation.\n",
            vtMngValueName( VTMNG_VALUE_RESULT, result ) );
}
```

vtMngMsgSize

Returns the total byte size of a message.

Prototype

U32 **vtMngMsgSize**(U8 ***msgOp***, U8 ***msgTypeId***, U8 ***msgCategory***, U32 ***eventId***, U32 ***count***)

Argument	Description
msgOp	Message operation (VTMNG_OP_*xxx*).
msgTypeId	Message type ID (VTMNG_*xxx*_ID).
msgCategory	Message category (VTMNG_CATEG_*xxx*).
eventId	Event [asynchronous indications only]; else use 0.
count	Number of elements [lists only]; else use 0.

Return values

Return value	Description
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

None.

Details

The **vtMngMsgSize** function is used internally by the VTMNG API to determine the required byte length of any given management message. This function is normally not called directly by the calling application.

If an application is operating in raw mode (formatting all management messages and handling receipt of all responses and traps directly), then the **vtMngMsgSize** function can be used to easily determine the buffer size of any VTMNG message.

Raw VTMNG API functions can be called even when the VTMNG API has not been initialized. This is because in raw mode, the controlling application manages all UDP communications.

Example

```
VT_MNG_MSG *vtmsg = NULL;
U32         msgSize;
msgSize = vtMngMsgSize( msgOp, msgTypeId, msgCategory, eventId, count );
if (msgSize != 0)
{   /* valid VTMNG API message length provided */
    vtmsg = (VT_MNG_MSG *)malloc( msgSize );
    if (vtmsg != NULL)
    {
        /* zero-fill the message and then initialize based on message type */
    }
    else
    {
        printf( unable to allocate memory for management message.\n" );
    }
}
```

vtMngPollLoop

Enters a polling loop and calls the appropriate management application upcall function on receipt of any management response (or asynchronous trap event). Optionally, the poll loop can also detect keyboard input and upcall the keyboard input handler function of the management application.

Prototype

U32 **vtMngPollLoop**(U8 ***keyboard***, U32 ***options***)

Argument	Description
keyboard	Flag specifying whether to register for keyboard input (operator commands).
options	Seconds of idle time before exiting loop. 0 = infinite Note: This field is reserved for future use. For the current release, *options* must be set to zero.

Return values

Return value	Description
VTMNG_ERR_NOT_INITIALIZED	VTMNG API has not been initialized. Call vtMngInit before calling this function.
VS_SUCCESS	Operator specified quit.
VTMNG_ERR_NOT_IMPLEMENTED	Requested capability not currently supported. This error is returned if non-zero *options* are specified.

Events

Event	Description
ALL	Any upcall function can be called from within the polling loop. For a complete breakdown of all supported upcalls, refer to *vtMngInit* on page 191.

Details

Use the **vtMngPollLoop** function after initializing the VTMNG API to enter into the main polling loop. Once a management application enters the polling loop, all program execution control is performed from within the **vtMngPollLoop** function. Any time a management message is received by the VTMNG API, the appropriate upcall is made back into the code space of the calling application.

Examples

```
 . . . initialize the VTMNG API by calling vtMngInit . . .
result = vtMngPollLoop( TRUE, 0 ); /* stay in polling loop forever */
if (result != VS_SUCCESS)
{   /* exitd the polling loop for reason other than operator specifying quit */
    printf( "Error %s returned from vtMngPollLoop.\n",
            vtMngValueName( VTMNG_VALUE_RESULT, result ) );
}
```

vtMngSetApp

Issues a request to modify the configuration of a particular application.

This function is defined but no application-level configurable elements exist. This function should be considered as reserved for future use.

Prototype

U32 **vtMngSetApp**(VTMNG_VTPADDR ****vtpAddr***, U32 ***appUnique***, VTMNG_APP_CFG ****cfg***, VTMNG_APP_CFG ****mask***)

Argument	Description
vtpAddr	Video transcoder platform address.
appUnique	Application ID value identifying the application control entity to be modified. The application ID is unique across all applications connected to the video transcoder platform. Use **vtMngGetAppList** to obtain all valid application ID values.
cfg	Set of configuration changes.
mask	Indication of which fields are being altered. Initialize this structure to VTMNG_NO_CHANGE, then set any fields being altered to VTMNG_CHANGE.

Return values

Return value	Description
VTMNG_ERR_NOT_SUPPORTED	There are no supported application-level configurable elements defined at this time.

Events

Event	Description
vtSetAppRsp(upcall)	When the VTMNG API receives the response to the set application request, the API will upcall the owner's set application response function (if an upcall was provided to the **vtMngInit** function).

Details

Reserved for future use.

Example

Reserved for future use.

vtMngSetChn

Issues a request to modify the configuration of a particular channel.

Prototype

U32 ***vtMngSetChn***(VTMNG_VTPADDR ****vtpAddr***, U32 ***chnUnique***, VTMNG_CHN_CFG ****cfg***, VTMNG_CHN_CFG ****mask***)

Argument	Description
vtpAddr	Video transcoder platform address.
chnUnique	Channel ID value that is unique across all channels connected to the video transcoder platform.
cfg	Set of configuration changes.
mask	Indication of which fields are being altered. Initialize this structure to VTMNG_NO_CHANGE, then set any fields being altered to VTMNG_CHANGE.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Channel identified by *chnUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.

Return value	Description
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtSetChnRsp(upcall)	When the VTMNG API receives the response to the set channel request, the API will upcall the owner's set channel response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngSetChn** function to change channel-specific configuration. This function should be considered as reserved for future use since there are no channel-level configuration fields that can be altered by a management application. For the current release, all channel-specific configuration is under the sole control of the controlling application.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    VTMNG_CHN_CFG   cfg;
    VTMNG_CHN_CFG   mask;
    memset( (void *)&cfg, 0, sizeof(VTMNG_CHN_CFG) );
    memset( (void *)&mask, VTMNG_NO_CHANGE, sizeof(VTMNG_CHN_CFG) );
    /* make any desired changes to the channel's configuration */
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngSetChn( &dest, chnUnique, &cfg, &mask );
. . .
>>> VTMNG API receives response and upcalls the application's vtSetChnRsp function:
void myMngSetChnRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     U32 chnUnique,
                     VTMNG_CHN_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "channel ID 0x%08X configuration set successfully\n",
                chnUnique );
    }
    else
    {
        printf( "Error %s while setting channel ID 0x%08X configuration\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), chnUnique );
    }
}
```

vtMngSetMon

Issues a request to modify a particular monitored process configuration.

Prototype

U32 **vtMngSetMon**(VTMNG_VTPADDR ***vtpAddr***, U32 ***monUnique***, VTMNG_MON_CFG ***cfg***, VTMNG_MON_CFG ***mask***);

Argument	Description
vtpAddr	Video transcoder platform address.
monUnique	Monitored process ID value identifying the process to be modified. The monitor ID is unique across all monitored processes on the video transcoder platform. All valid monitored process ID values can be obtained using the **vtMngGetMonList** function.
cfg	Set of configuration changes.
mask	Indication of which fields are being altered. Initialize this structure to VTMNG_NO_CHANGE, then set any fields being altered to VTMNG_CHANGE.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Process identified by *monUnique* does not exist.

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtSetMonRsp(upcall)	When the VTMNG API receives the response to the set monitored process request, the API will upcall the owner’s set monitored process response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngSetMon** function to modify the configuration of any monitored process. All monitored process configuration is automatically stored to a configuration file maintained on the video transcoder platform (*vtmon.cfg*) This allows changes to remain across any power failure or video transcoder platform server reboot.

For more information, refer to *VTMNG_MON_CFG* on page 282.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    VTMNG_MON_CFG   cfg;
    VTMNG_MON_CFG   mask;
    memset( (void *)&cfg, 0, sizeof(VTMNG_MON_CFG) );
    memset( (void *)&mask, VTMNG_NO_CHANGE, sizeof(VTMNG_MON_CFG) );
    /* make any desired changes to the monitored process's configuration */
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngSetMon( &dest, monUnique, &cfg, &mask );
. . .
>>> VTMNG API receives response and upcalls the application's vtSetMonRsp function:
void myMngSetMonRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     U32 monUnique,
                     VTMNG_MON_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "monitored process ID 0x%08X configuration set successfully\n",
                monUnique );
    }
    else
    {
        printf( "Error %s while setting monitored process ID 0x%08X configuration\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), monUnique );
    }
}
```

vtMngSetVtp

Issues a request to modify the video transcoder platform-level configuration.

Prototype

U32 **vtMngSetVtp**(VTMNG_VTPADDR ****vtpAddr***, VTMNG_VTP_CFG ****cfg***, VTMNG_VTP_CFG ****mask***)

Argument	Description
vtpAddr	Video transcoder platform address.
cfg	Set of configuration changes.
mask	Indication of which fields are being altered. Initialize this structure to VTMNG_NO_CHANGE, then set any fields being altered to VTMNG_CHANGE.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtSetVtpRsp(upcall)	When the VTMNG API receives the response to the set video transcoder platform request, the API will upcall the owner's set video transcoder platform response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngSetVtp** function to modify any of the video transcoder platform-level configuration fields. All video transcoder platform-level configuration is automatically stored to a configuration file maintained on the video transcoder platform (*vtp.cfg*). This allows changes to remain across any power failure or video transcoder platform server reboot.

For more information, refer to *VTMNG_VTP_CFG* on page 299.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    VTMNG_VTP_CFG    cfg;
    VTMNG_VTP_CFG    mask;
    memset( (void *)&cfg, 0, sizeof(VTMNG_VTP_CFG) );
    memset( (void *)&mask, VTMNG_NO_CHANGE, sizeof(VTMNG_VTP_CFG) );
    /* make any desired changes to the VTP-level configuration */
    strcpy( cfg.vtpName, "my VTP name" );
    mask.vtpName[0] = VTMNG_CHANGE;     /* show that this field has been changed */
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngSetVtp( &dest, &cfg, &mask );
. . .
>>> VTMNG API receives response and upcalls the application's vtSetVtpRsp function:
void myMngSetVtpRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                     VT_MNG_MSG *msg, U32 result,
                     VTMNG_VTP_CFG *cfg )
{
    if (result == VS_SUCCESS)
    {
        printf( "VTP-level configuration set successfully\n" );
    }
    else
    {
        printf( "Error %s while setting VTP-level configuration\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngShutdown

Shuts down all use of the video transcoder management interface.

Prototype

U32 **vtMngShutdown**( )

This function has no arguments.

Return values

result of shutdown

Return value	Description
VS_SUCCESS	VTMNG API shut down successfully.

Events

None.

Details

Call **vtMngShutdown** any time after a successful initialization to cleanly terminate the VTMNG API. This allows the VTMNG API to close any UDP sockets it is currently listening on.

Examples

```
    result = vtMngShutdown( );
    if (result != VS_SUCCESS)
    {
        printf( "Error %s returned from vtMngShutdown.\n",
```

```
            vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
```

vtMngValueName

Provides the application with an ASCII string that corresponds to the management value provided.

Prototype

S8 ***vtMngValueName*** (U32 ***valueType***, U32 ***value***)

Argument	Description
valueType	Type of value for which to provide an ASCII string. Valid values: VTMNG_VALUE_RESULT Result code provided by any VTMNG API function. VTMNG_VALUE_LASTERR Any value reported in a management statistic named lastError. Used to provide lower-layer error information as reported by the decoder (receive statistics) or encoder (transmit statistics).
value	Numeric value for which to provide an ASCII string equivalent.

Return values

The function always returns an ASCII string:

- In the case where an invalid ***valueType*** is provided, the function returns the string UNKNOWN VALUE TYPE [0x***XXXXXXXX***] where ***XXXXXXXX*** is the hexadecimal representation of the ***valueType*** provided.
- In the case where the ***valueType*** is valid but the value is out of range for the given type, the function returns the string INVALID VALUE [0x***XXXXXXXX***] FOR ***sssss*** where ***XXXXXXXX*** is the hexadecimal representation of the value provided and ***sssss*** is the string representation of the ***valueType***.

The VTMNG API uses a single global area to format error indication strings. A subsequent call to **vtMngValueName** can overwrite the previous error string text if another invalid ***valueType*** or ***value*** is specified.

Events

None.

Details

The management application can call **vtMngValueName** to obtain an ASCII string representation of the desired value. This function is provided as an aid to creating diagnostic messages.

Use a ***valueType*** of VTMNG_VALUE_RESULT to obtain the ASCII name for any error code returned by a VTMNG API function or for any error code provided in a VTMNG API response or event (trap) upcall.

Use a ***valueType*** of VTMNG_VALUE_LASTERR to obtain the ASCII name for any error code reported in a lastError management statistic.

Example

```
VTMNG_CHN_STATS    *chstats = &vtmsg->rsp.getEnt.u.chn.stats;
printf( "Last reported errors from channel video-type specific statistics:\n" );
printf( "  endpoint A RX Last Error: %s\n",
        vtMngValueName( VTMNG_VALUE_LASTERR,
                        chnStats->endpointA.vtype.rx.vtcomm.lastError ) );
printf( "  endpoint B TX Last Error: %s\n",
        vtMngValueName( VTMNG_VALUE_LASTERR,
                        chnStats->endpointB.vtype.tx.vtcomm.lastError ) );
/* NOTE: for full-duplex, should show endpoint B RX and endpoint A TX */
```

vtMngZeroApp

Issues a request to get and then zero an application's statistics.

Prototype

U32 **vtMngZeroApp**(VTMNG_VTPADDR ***vtpAddr***, U32 ***appUnique***)

Argument	Description
vtpAddr	Video transcoder platform address.
appUnique	Application ID value identifying the application control entity to be zeroed. The application ID is unique across all applications connected to the video transcoder platform. All valid application ID values can be obtained using the **vtMngGetAppList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Application identified by *appUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtZeroAppRsp(upcall)	When the VTMNG API receives the response to the zero application statistics request, the API will upcall the owner's zero application statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngZeroApp** function to request the current statistics for a given application with these statistics being zeroed to create a clean base for future statistics accumulation.

Example

```
    U32               result;
    VTMNG_VTPADDR     dest;            /* destination addressing information */
    U32               myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngZeroApp( &dest, appUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtZeroAppRsp function:
void myMngZeroAppRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                      VT_MNG_MSG *msg, U32 result,
                      U32 appUnique,
                      VTMNG_APP_STATS *appStats )
{
    if (result == VS_SUCCESS)
    {
        printf( "application ID 0x%08X statistics successfully zeroed\n",
                appUnique );
        /* NOTE: appStats holds statistics just before stats were zero'd */
    }
    else
    {
        printf( "Error %s while zeroing application ID 0x%08X statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), appUnique );
    }
}
```

vtMngZeroChn

Issues a request to get and then zero a channel's statistics.

Prototype

U32 **vtMngZeroChn**(VTMNG_VTPADDR ***_vtpAddr_**, U32 **_chnUnique_**)

Argument	Description
vtpAddr	Video transcoder platform address.
chnUnique	Channel ID value identifying the channel to be zeroed. The channel ID is unique across all channels connected to the video transcoder platform. All valid channel ID values can be obtained using the **vtMngGetChnList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Channel identified by *chnUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtZeroChnRsp(upcall)	When the VTMNG API receives the response to the zero channel statistics request, the API will upcall the owner's zero channel statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngZeroChn** function to request the current statistics for a given transcoder channel with these statistics being zeroed to create a clean base for future statistics accumulation.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;            /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngZeroChn( &dest, chnUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtZeroChnRsp function:
void myMngZeroChnRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                      VT_MNG_MSG *msg, U32 result,
                      U32 chnUnique,
                      VTMNG_CHN_STATS *chnStats )
{
    if (result == VS_SUCCESS)
    {
        printf( "channel ID 0x%08X statistics successfully zeroed\n",
                chnUnique );
        /* NOTE: chnStats holds statistics just before stats were zero'd */
    }
    else
    {
        printf( "Error %s while zeroing channel ID 0x%08X statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), chnUnique );
    }
}
```

vtMngZeroMon

Issues a request to get and then zero the statistics for a monitored process.

Prototype

U32 vtMngZeroMon(VTMNG_VTPADDR ****vtpAddr***, U32 ***monUnique***)

Argument	Description
vtpAddr	Video transcoder platform address.
monUnique	Monitored process ID value identifying the process to be zeroed. The monitor ID is unique across all monitored processes on the video transcoder platform. All valid monitored process ID values can be obtained using the **vtMngGetMonList** function.

Return values

Return value	Description
VTMNG_ERR_DOES_NOT_EXIST	Process identified by *monUnique* does not exist.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtZeroMonRsp(upcall)	When the VTMNG API receives the response to the zero monitored process statistics request, the API will upcall the owner's zero monitored process statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngZeroMon** function to request the current statistics for a given monitored process with these statistics being zeroed to create a clean base for future statistics accumulation.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngZeroMon( &dest, monUnique );
. . .
>>> VTMNG API receives response and upcalls the application's vtZeroMonRsp function:
void myMngZeroMonRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                      VT_MNG_MSG *msg, U32 result,
                      U32 monUnique,
                      VTMNG_APP_STATS *monStats )
{
    if (result == VS_SUCCESS)
    {
        printf( "monitored process ID 0x%08X statistics successfully zeroed\n",
                monUnique );
        /* NOTE: monStats holds statistics just before stats were zero'd */
    }
    else
    {
        printf( "Error %s while zeroing monitored process ID 0x%08X statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ), monUnique );
    }
}
```

vtMngZeroTotal

Issues a request to get and then zero the total statistics.

Prototype

U32 **vtMngZeroTotal**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtZeroTotalRsp(upcall)	When the VTMNG API receives the response to the zero total statistics request, the API will upcall the owner's zero total statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngZeroTotal** function to request the current total statistics for a given video transcoder platform with these statistics being zeroed to create a clean base for future statistics accumulation.

Example

```
    U32              result;
    VTMNG_VTPADDR    dest;           /* destination addressing information */
    U32              myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngZeroTotal( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtZeroTotalRsp function:
void myMngZeroTotalRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                        VT_MNG_MSG *msg, U32 result,
                        VTMNG_ST_ENTRY *stats )
{
    if (result == VS_SUCCESS)
    {
        printf( "Total statistics successfully zeroed\n" );
        /* NOTE: stats holds statistics just before stats were zero'd */
    }
    else
    {
        printf( "Error %s while zeroing total statistics\n",
               vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

vtMngZeroVtp

Issues a request to get and then zero the video transcoder platform-level statistics.

Prototype

U32 **vtMngZeroVtp**(VTMNG_VTPADDR ****vtpAddr***)

Argument	Description
vtpAddr	Video transcoder platform address.

Return values

Return value	Description
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory available. Unable to create request message.
VTMNG_ERR_INVALID_SIZE	Message size-related error encountered.
VTMNG_ERR_UNKNOWN_TYPE	Unknown or unsupported value encountered.

Events

Event	Description
vtZeroVtpRsp(upcall)	When the VTMNG API receives the response to the zero video transcoder platform-level statistics request, the API will upcall the owner's zero video transcoder platform-level statistics response function (if an upcall was provided to the **vtMngInit** function).

Details

Use the **vtMngZeroVtp** function to request the current video transcoder platform-level statistics with these statistics being zeroed to create a clean base for future statistics accumulation.

Example

```
    U32             result;
    VTMNG_VTPADDR   dest;           /* destination addressing information */
    U32             myKey = 12345;
    dest.ipv4Addr = inet_addr( "127.0.0.1" );
    dest.sendkey = myKey;
    result = vtMngZeroVtp( &dest );
. . .
>>> VTMNG API receives response and upcalls the application's vtZeroVtpRsp function:
void myMngZeroVtpRsp( void *userkey, VTMNG_VTPADDR *vtpAddr,
                      VT_MNG_MSG *msg, U32 result,
                      VTMNG_VTP_STATS *vtpStats )
{
    if (result == VS_SUCCESS)
    {
        printf( "VTP-level statistics successfully zeroed\n" );
        /* NOTE: vtpStats holds statistics just before stats were zero'd */
    }
    else
    {
        printf( "Error %s while zeroing VTP-level statistics\n",
                vtMngValueName( VTMNG_VALUE_RESULT, result ) );
    }
}
```

10. Transcoder resource controller structures

TRC structures overview

This section provides an alphabetical reference to the TRC structures. It defines each structure and provides a description of the fields in the structure.

Note: All structures have space reserved at the end of the structure. To allow for future compatibility, you must zero-fill all structures.

The following table lists the TRC structures and the functions that are used by each structure:

Structure	Functions that use this structure
tTrcCfgValue	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcChAll	**trcChannelStatus**
tTrcChConfig	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcChInfo	**trcInfoVideoChannel**
tTrcChOptions	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcChStatus	**trcChannelStatus** **trcInfoVideoChannel**
tTrcChSummary	**trcChannelStatus** **trcInfoVideoChannel**
tTrcEndInput	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcEndOutput	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcEndpoint	**trcInfoVideoChannel** **trcStartVideoChannel**
tTrcError	**trcChannelStatus** **trcInfoVideoChannel** **trcVTPStatus**

tTrcErrorDesc	**trcChannelStatus** **trcInfoVideoChannel** **trcVTPStatus**
tTrcRes	**trcInfoVideoChannel**
tTrcMessage	**trcInitialize** (provided on all TRC message [event] upcalls)
tTrcOvlConfig	**trcCreateOverlay**
tTrcOvlContent	**trcCreateOverlay**
tTrcOvlScroll	**trcCreateOverlay**
tTrcUsage	**trcUsage**
tTrcVtpAll	**trcVTPStatus**
tTrcVtpSummary	**trcVTPStatus**
tTrcVtpUsage	**trcVTPStatus**

TRC structure relationships

Some TRC structures have substructures that provide support to the main structure. The following table lists those TRC structures and their substructures:

Primary structure	Substructures		
tTrcChAll	tTrcChSummary tTrcChStatus tTrcErrorDesc tTrcError		
tTrcChConfig	tTrcEndpoint	tTrcEndInput	
		tTrcEndOutput	tTrcCfgValue
	tTrcChOptions		
tTrcChInfo	tTrcChSummary	tTrcChStatus tTrcErrorDesc tTrcError	
	tTrcRes		

	tTrcChConfig	tTrcEndInput tTrcEndOutput	tTrcCfgValue
tTrcVtpAll	tTrcVtpSummary tTrcVtpUsage tTrcErrorDesc tTrcError		
tTrcOvlBorder	tTrcOvlColor		
tTrcOvlConfig	tTrcOvlCoordinates tTrcOvlColor tTrcOvlBorder tTrcOvlFont		
tTrcOvlContent	tTrcOvlScroll		
tTrcOvlFont	tTrcOvlColor		
tTrcOvlScroll	tTrcOvlCoordinates		

tTrcCfgValue

Sets output configuration values for the encoder. This structure is used with **trcStartVideoChannel** and **trcInfoVideoChannel** as part of the tTrcEndOutput structure.

Definition

```
typedef struct
{
    U8          isSet;          /* TRC_FALSE = value not set by application (use default
                                   behavior)
                                 * TRC_TRUE  = value has been set            (use value) */
    U8          avail[3];       /* avaliable for future use */
    U32         value;          /* value (when isSet = TRC_TRUE)
                                   [for bit values use TRC_TRUE or TRC_FALSE] */
} tTrcCfgValue;
```

Field listing

Field	Type	Description
isSet	U8	Indicates whether the application has set the value of the specified output configuration field for the encoder. Valid values are: TRC_FALSE - (Default). Application has not set the value of the specified output configuration field. Transcoder uses the default value. TRC_TRUE - Application has set the value of the specified output configuration field.

Field	Type	Description
avail	U8	Available for future use.
value	U32	Value of the output configuration field when the isSet field = TRC_TRUE. For BIT options, the value must be TRC_TRUE or TRC_FALSE. For other options, the value is a number.

Example

The following example enables the application to control the behavior of the duplicateInitialI field in the tTrcEndOutput structure. In this example, the application directs the encoder to duplicate the initial I-frame:

```
endpoint.chanOut.duplicateInitialI.isSet = TRC_TRUE;
endpoint.chanOut.duplicateInitialI.value = TRC_TRUE;
```

tTrcChAll

Provides a summary view of all channels currently in use by the calling application. It is used with **trcChannelStatus**.

Definition

```
typedef struct
{
    U16             chanDefined;            /* total number of channels that have been
                                               defined for this instance of the TRC */
    tTrcChSummary   chan[TRC_MAX_CHANNELS]; /* summary-level status information
                                               for each defined channel */
    U8              reserved[32];           /* reserved for future use */
} tTrcChAll;
```

Field listing

Field	Type	Description
chanDefined	U16	Total number of channels that are defined for this instance of the TRC. Defined channels include all created channels that are not yet destroyed.
chan	Structure	Array of tTrcChSummary structures, which provides status information for each defined channel. Supports a maximum of 512 channels. Refer to *tTrcChSummary* on page 227.
reserved	U8	Reserved for future use.

tTrcChConfig

Describes the configuration of the two endpoints in the video channel with optional direction-specific decoder and encoder configurations. This structure configures the transcoding features that are in use when a channel is started by calling **trcStartVideoChannel**. An active channel's configuration can be determined by calling **trcInfoVideoChannel**, which provides the tTrcChConfig structure as part of the tTrcChInfo structure.

When providing this structure to **trcStartVideoChannel**, the structure must first be set to indicate default values for all parameters, as follows:

```
memset( &cfg, TRC_CONFIG_DEFAULT, sizeof(cfg) ); /* start from all defaults */
```

Definition

```
typedef struct
{
    tTrcEndpoint    endpointA;      /* description of endpoint A configuration */
    tTrcEndpoint    endpointB;      /* description of endpoint B configuration */
    tTrcChOptions   decoder[TRC_DIR_COUNT]; /* optional decoder config for each
                                               direction */
    tTrcChOptions   encoder[TRC_DIR_COUNT]; /* optional encoder config for each
                                               direction */
    U8              reserved[32];   /* reserved for future use */
} tTrcChConfig;
```

Field listing

Field	Type	Description
endpointA	Structure	Defines the configuration for endpoint A. Specified in *tTrcEndpoint* on page 235.
endpointB	Structure	Defines the configuration of endpoint B. Specified in *tTrcEndpoint* on page 235.
decoder	Structure	(MPEG-4 only) Optional decoder configuration information (DCI) for each transcoding direction specified in *tTrcChOptions* on page 225. An array index is used to identify the leg to which the decoder configuration applies. Valid values are: TRC_DIR_SIMPLEX Command applies to the only connection leg in a simplex channel that transcodes from endpoint A to endpoint B. TRC_DIR_FDX1 Command applies to the full-duplex connection leg that transcodes from endpoint A to endpoint B. TRC_DIR_FDX2 Command applies to the full-duplex connection leg that transcodes from endpoint B to endpoint A.

Field	Type	Description
encoder	Structure	(MPEG-4 only) Optional encoder configuration information for each transcoding direction specified in *tTrcChOptions* on page 225. An array index is used to identify the leg to which the optional encoder DCI configuration applies. Valid values are: TRC_DIR_SIMPLEX Command applies to the only connection leg in a simplex channel that transcodes from endpoint A to endpoint B. TRC_DIR_FDX1 Command applies to the full-duplex connection leg that transcodes from endpoint A to endpoint B. TRC_DIR_FDX2 Command applies to the full-duplex connection leg that transcodes from endpoint B to endpoint A.
reserved	U8	Reserved for future use.

tTrcChInfo

Describes detailed channel information for a specified channel. This structure is used with **trcInfoVideoChannel**.

Definition

```
typedef struct
{
    tTrcChSummary   summary;              /* summary-level status information */
    tTrcRes         res;                  /* assigned transcoder resource addressing
                                             information */
    tTrcChConfig    config;               /* channel configuration information
                                             (provided when channel started) */
    S8              name[TRC_CHNAME_LEN]; /* ASCII name of the channel (set via
                                             trcNameVideoChannel) */
    U8              reserved[2760];       /* reserved for future use */
} tTrcChInfo;
```

Field listing

Field	Type	Description
summary	Structure	Current summary-level status information for the channel. Specified in *tTrcChSummary* on page 227.
res	Structure	Information on the transcoder resources assigned to the channel. Specified in *tTrcRes* on page 251.
config	Structure	Configuration information for the channel. Specified in *tTrcChConfig* on page 223.
name[n]	S8	ASCII name of channel assigned through *trcNameVideoChannel* on page 135.
reserved	U8	Reserved for future use.

tTrcChOptions

Sets up optional encoder and decoder configurations. It is used with **trcStartVideoChannel** as part of the tTrcChConfig structure.

Definition

```
typedef struct
{
    void           *optData;        /* pointer to options data record (ex: DCI for
                                       MPEG-4) */
    U32             optSize;        /* size of options data (in bytes) */

    U8              reserved[32];   /* reserved for future use */
} tTrcChOptions;
```

Field listing

Field	Type	Description
optData	void*	Pointer to the options data record (for example, DCI for MPEG-4). Allows you to optionally configure the transcoder decoder, encoder, or both for each channel direction. Use NULL to indicate no optional configuration record.
optSize	U32	Byte length of the optional data record when optData is non-NULL.
reserved	U8	Reserved for future use.

tTrcChStatus

Describes current channel status information for a specified channel. It is used with **trcChannelStatus** as part of the tTrcChSummary structure, and with **trcInfoVideoChannel** as part of the tTrcChInfo structure.

Definition

```
typedef struct
{
    U16             state;          /* current overall state of the channel
                                       (TRC_STATE_xxx) */
    U16             type;           /* type of channel (TRC_CH_xxx) */
    tTrcErrorDesc   errorDesc;      /* description of any channel-level
                                       errors encountered */

    U8              reserved[32];   /* reserved for future use */
} tTrcChStatus;
```

Field listing

Field	Type	Description
state	U16	Current state of the channel. Valid state values are: TRC_STATE_UNAVAIL Channel is not available because it is not licensed. TRC_STATE_AVAILABLE Channel is available for use. TRC_STATE_CREATING Channel create is in progress. TRC_STATE_STOPPED Channel was created, but is not actively transcoding. Resources are reserved. TRC_STATE_STARTING Channel is starting. TRC_STATE_ACTIVE Channel was successfully started and is actively transcoding. TRC_STATE_STOPPING Channel is stopping. TRC_STATE_DESTROYING Channel destroy is in progress.
type	U16	Type of channel. Valid values are: TRC_CH_SIMPLEX TRC_CH_FDX TRC_CH_UNUSED - Channel has not been created. (Ignore this entry when processing **trcChannelStatus** results.) **Note:** TRC_CH_SIMPLEX and TRC_CH_FDX can be combined with other channel type options including TRC_CH_OVERLAY and TRC_CH_RTCP.
errorDesc	Structure	Summary of the error types detected by the channel. Specified in *tTrcErrorDesc* on page 239.
reserved	U8	Reserved for future use.

tTrcChSummary

Provides high-level information for each defined channel. This structure is used with **trcChannelStatus**, and is part of the tTrcChAll structure.

Definition

```
typedef struct
{
    TRC_HANDLE       trcHandle;    /* TRC_defined handle associated with the channel */
    void            *userKey;      /* user-provided key associated with the channel */
    U32              vtpId;        /* identifies the VTP that the channel has been
```

```
                                  assigned to */
    tTrcChStatus    status;       /* top-level channel status information */
    U32             vtpChanId;    /* unique channel ID (assigned by the VTP) */

    U8              reserved[28]; /* reserved for future use */
} tTrcChSummary;
```

Field listing

Field	Type	Description
trcHandle	Structure	TRC-defined handle associated with the channel. For more information, refer to *trcCreateVideoChannel* on page 116.
userKey	void*	User-provided key associated with the channel.
vtpId	U32	ID of the video transcoder platform to which the channel is assigned.
status	Structure	High-level status information for the channel. Specified in *tTrcChStatus* on page 226.
vtpChanId	U32	Unique channel ID assigned by the video transcoder platform.
reserved	U8	Reserved for future use.

tTrcEndInput

Defines the specific endpoint characteristics required for endpoints from which the transcoder receives input. Endpoint configuration is specified for **trcStartVideoChannel** and can be queried using **trcInfoVideoChannel**.

For a simplex channel, only the tTrcEndInput configuration for endpoint A is required. The transcoder receives a video bit stream from endpoint A only.

For a full-duplex channel, the transcoder receives a video bit stream from both endpoints. The tTrcEndInput structure must be configured for both endpoint A and endpoint B.

Definition

```
typedef struct
{
    U16             jitterMode;     /* mode of operation for receive jitter buffer
                                       control */
#define TRC_JITTER_NONE     1       /* no jitter buffer in use */
#define TRC_JITTER_STATIC   2       /* constant jitter buffer offset */

    U32             jitterLatency;  /* jitter buffer latency (in milliseconds): amount of
                                       time that received packets are delayed to allow
                                       for jitter buffer reordering */
    U8              rtcpReceiver;   /* mode for handling RTCP communication with input
                                       endpoint (TRC_RTCP_xxx) */
    U8              avail[3];
    U32             rtcpRxTimeout;  /* max time (in msecs) that can pass without
                                       receiving RTP or RTCP before considering endpoint
                                       timed out (0 = no timeout)
                                       [only valid if rtcpReceiver != 0] */
    U8              reserved[24];   /* reserved for future use */
} tTrcEndInput;
```

Field listing

Field	Type	Description
jitterMode	U16	Mode of operation for receive jitter buffer control. Valid values are: TRC_JITTER_NONE - (Default). No jitter buffer in use. Video data goes directly to the transcoder. TRC_JITTER_STATIC - Constant jitter buffer offset. Use TRC_JITTER_STATIC and set the jitter latency if the transcoder is receiving from an endpoint that is connected over a network that can introduce out-of-order or bursty packet delivery.
jitterLatency	U32	The jitter buffer latency in milliseconds. This defines the amount of time that received packets are delayed to allow for jitter buffer reordering. This field is applicable only when the jitter buffer is in use.
rtcpReceiver	U8	Mode for handling RTCP communication with input endpoint: TRC_RTCP_DEFAULT– Handle RTCP as specified by video transcoder platform-level default configuration (rtcpMode). TRC_RTCP_DISABLED – Do not listen for RTCP. TRC_RTCP_ENABLED – Listen for receive of RTCP from remote endpoint and send RTCP to remote as appropriate.
avail	U8	Space available for future use.
rtcpRxTimeout	U32	Maximum time in milliseconds that can pass without receiving RTP or RTCP before considering endpoint timed out. Set this field to 0 for no timeout. This is only valid if rtcpReceiver = TRC_RTCP_ENABLED.

Field	Type	Description
reserved	U8	Reserved for future use.

tTrcEndOutput

Defines the specific endpoint characteristics required for endpoints to which the transcoder sends output. Endpoint configuration is specified for **trcStartVideoChannel** and can be queried using **trcInfoVideoChannel**.

The transcoder sends video bit streams as follows:

- For a simplex channel, the transcoder sends a video bit stream to endpoint B only. The tTrcEndOutput structure must be configured only for endpoint B.
- For a full-duplex channel, the transcoder sends a video bit stream to both endpoints. The tTrcEndOutput structure must be configured for both endpoint A and endpoint B.

The tTrcEndOutput structure contains optional output configuration fields for the encoder that are defined by *tTrcCfgValue* on page 221. Each of these optional output configuration fields has a default behavior that is controlled by configuration files that are resident on every video transcoder platform. There is a separate configuration file for H.263 (*encodeh263.cfg*) and MPEG-4 (*encodempeg.cfg*) endpoints.

Definition

```
typedef struct
{
    S8            ipAddr[TRC_IPADDR_LEN]; /* IP address of endpoint that transcoder
                                             outputs RTP video bitstream to */
   U16            rtpPort;           /* UDP port number of endpoint that transcoder
                                        outputs RTP video bitstream to */
   U32            payloadId;         /* payload ID used in outbound RTP packets issued
                                         by the transcoder */
   U8             tos;               /* type of Service used in outbound RTP packets
                                        issued by the transcoder */

   tTrcCfgValue   duplicateInitialI; /* BIT: TRC_TRUE = issue duplicate of initial
                                        I-frame */
   tTrcCfgValue   dropEarlyFrames;   /* BIT: TRC_TRUE = encoder will drop early
                                        frames */
   tTrcCfgValue   dropLowQualFrames; /* BIT: TRC_TRUE = encoder will drop low quality
                                        frames */
   tTrcCfgValue   timeResolution;    /* (MPEG4 only) time resolution */
   tTrcCfgValue   partitioned;       /* (MPEG4 only) BIT: TRC_TRUE = enable
                                     *  partitioning, TRC_FALSE = disable partitioning */
   tTrcCfgValue   framesPerI;        /* when the encoder generates an I-frame in
                                      * relation to P-frames
                                      *   0    - Default. Encoder generates only one
                                      *          I-frame. The other frames are P-frames
                                      *   else - Frame interval for generating
                                      *          I-frames. For example, a value of 10
                                      *          means that the encoder generates one
                                      *          I-frame and nine P-frames for every
                                     *          ten frames */
   tTrcCfgValue   numMbRefresh;      /* encoder number: macroblock refresh for
                                        intra-coding of P-frames (only valid when
                                        framesPerI = 0) */
   tTrcCfgValue   packetSize;        /* (MPEG4 only) encoder packet size */
   tTrcCfgValue   timePeriod;        /* whether the encoder uses a fixed time increment
                                     *  for transmitting frames
                                     *   0    - Default. Encoder transmits frames using
                                     *          the time interval associated with the
```

```
                                    *          corresponding decoded input frames
                                    *   else - count of ticks [as specified by
                                    *          timeResolution] Encoder transmits
                                    *          frames using the specified time
                                    *          interval */
   tTrcCfgValue   acPrediction;      /* (MPEG4 only) BIT: TRC_TRUE = enable AC
                                        prediction */
   tTrcCfgValue   useType2Mb;        /* (MPEG4 only) BIT: TRC_TRUE = enable use of
                                        type 2 macroblocks */

   U8             rtcpTransmitter;   /* mode for handling RTCP communication with
                                        output endpoint (TRC_RTCP_xxx) */
   U8             avail[3];
   U32            rtcpTxTimeout;     /* max time (in msecs) that can pass without
                                        receiving RTCP before considering endpoint
                                        timed out (0 = no timeout) [only valid if
                                        rtcpTransmitter != 0] */
   U8             reserved[24];      /* reserved for future use */
} tTrcEndOutput;
```

Field listing

Field	Type	Description
ipAddr[*n*]	S8	IP address of the endpoint, as an ASCII string representation. The transcoder outputs the RTP video bit stream to this address. The value for ipAddr[*n*] cannot be defaulted by using the TRC_CONFIG_DEFAULT literal.
rtpPort	U16	UDP port number of the endpoint. The transcoder outputs the RTP video bit stream to this RTP port. The value for rtpPort cannot be defaulted by using the TRC_CONFIG_DEFAULT literal.
playloadID	U32	Payload ID used in outbound RTP packets issued by the transcoder. Set the payload ID values as follows: For H.263 endpoints using packetization according to RFC 2190, use 34 (the default). For H.263 endpoints using packetization according to RFC 2429, choose from the dynamic range 96 - 127. The default is 96. For MPEG-4 endpoints, choose from the dynamic range 96 - 127. The default is 100.
tos	U8	Type of service used in outbound RTP packets issued by the transcoder. The default is 0 (TRC_CONFIG_DEFAULT), which specifies standard quality of service.
duplicateInitialI	Structure	(BIT) Indicates whether the encoder duplicates the initial I-frame. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - Encoder does not duplicate the initial I-frame. TRC_TRUE - Encoder duplicates the initial I-frame.
dropEarlyFrames	Structure	(BIT) Indicates whether the encoder drops early frames. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - (Default). Encoder does not drop early frames. TRC_TRUE - Encoder drops early frames.

Field	Type	Description
dropLowQualFrames	Structure	(BIT) Indicates whether the encoder drops low quality frames. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - (Default). Encoder does not drop low quality frames. TRC_TRUE - Encoder drops low quality frames.
timeResolution	Structure	(MPEG-4 only) Time resolution of the outbound video stream, in ticks per second specified in *tTrcCfgValue* on page 221. Zero is not a valid value.
partitioned	Structure	(BIT, MPEG-4 only) Indicates whether the encoder enables or disables partitioning. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - (Default). Disable partitioning. TRC_TRUE - Enable partitioning.
framesPerI	Structure	Specifies when the encoder generates an I-frame in relation to P-frames. Specified in *tTrcCfgValue* on page 221. Valid values are: 0 - (Default). Encoder generates only one I-frame. The other frames are P-frames. Non-zero integer - Frame interval for generating I-frames. For example, a value of 10 means that the encoder generates one I-frame and nine P-frames for every ten frames.
numMbRefresh	Structure	Intra macroblock refresh rate specified in *tTrcCfgValue* on page 221. This is the number of macroblocks coded as I-blocks in a P-frame. The position of the I-blocks rotates through the image on successive P-frames. The numMbRefresh field applies only when framesPerI = 0. The default is 3.
packetSize	Structure	(MPEG-4 only) Encoder packet size, in bytes, specified in *tTrcCfgValue* on page 221.

Field	Type	Description
timePeriod	Structure	Indicates whether the encoder uses a fixed time increment for transmitting frames. Specified in *tTrcCfgValue* on page 221. Valid values are: 0 - (Default). Encoder transmits frames using the time interval associated with the corresponding decoded input frames. A non-zero integer count of ticks as specified by timeResolution - Encoder transmits frames using the specified time interval.
acPrediction	Structure	(BIT, MPEG-4 only) Indicates whether to enable AC prediction for the encoder. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - (Default). Disable AC prediction. TRC_TRUE - Enable AC prediction.
useType2Mb	Structure	(BIT) Indicates whether to enable the use of type 2 macroblocks for the encoder. Specified in *tTrcCfgValue* on page 221. Valid values are: TRC_FALSE - (Default). Disable the use of type 2 macroblocks. TRC_TRUE - Enable the use of type 2 macroblocks.
rtcpTransmitter	U8	Mode for handling RTCP communication with the output endpoint: TRC_RTCP_DEFAULT – Handle RTCP as specified by VTP-level default configuration (rtcpMode). TRC_RTCP_DISABLED – Do not send RTCP. TRC_RTCP_ENABLED – Forward any RTCP received from the remote input endpoint to the output endpoint.
avail	U8	Space available for future use.
rtcpTxTimeout	U32	Maximum time in milliseconds that can pass without receiving RTCP before considering the endpoint timed out. Set this field to 0 for no timeout. This is only valid if rtcpTransmitterr = TRC_RTCP_ENABLED.
reserved	U8	Reserved for future use.

tTrcEndpoint

Defines each endpoint involved in a transcoder channel. It is used with **trcStartVideoChannel** and **trcInfoVideoChannel**, as part of the tTrcChConfig structure.

Definition

```
typedef struct
{
    U16             vidType;        /* identifies the type of video format used by the
                                       endpoint */
#define TRC_VIDTYPE_MPEG4       1   /* MPEG4 encoded bit stream */
#define TRC_VIDTYPE_H263        2   /* H.263 encoded bit stream */
#define TRC_VIDTYPE_PIXEL       3   /* pixel encoded bit stream */

    U16             profile;        /* type of profile in use by endpoint */
#define TRC_PROFILE_SIMPLE      1   /* MPEG-4: simple profile in use */
#define TRC_PROFILE_BASELINE    2   /* H.263: baseline profile in use */

    U16             level;          /* profile level in use by endpoint */
#define TRC_MPEG4_LEVEL_0       1   /* MPEG-4 profile level 0 */
#define TRC_MPEG4_LEVEL_1       2   /* MPEG-4 profile level 1 */
#define TRC_MPEG4_LEVEL_2       3   /* MPEG-4 profile level 2 */
#define TRC_MPEG4_LEVEL_3       4   /* MPEG-4 profile level 3 */
#define TRC_H263_LEVEL_10       5   /* H.263 profile level 10 */
#define TRC_H263_LEVEL_20       6   /* H.263 profile level 20 */
#define TRC_H263_LEVEL_30       7   /* H.263 profile level 30 */

    U16             dataRate;       /* video bit stream data rate in use by the
                                       endpoint expressed in kilobits/second */
    U16             frameRate;      /* video bit stream frame rate in use by the
                                       endpoint expressed in frames/second */
    U8              frameRes;       /* video frame resolution */
#define TRC_FRAME_RES_QCIF      1   /* Quarter Common Interchange Format (176 x 144) */
#define TRC_FRAME_RES_CIF       2   /* Common Interchange Format (352 x 288) */
#define TRC_FRAME_RES_SUBQCIF   3   /* Sub-
Quarter Common Interchange Format (128 x 96) */

    U8              packetizeMode;  /* packetization mode */
#define TRC_PACKETIZE_2190      1   /* endpoint uses packetization mode defined in
                                       RFC 2190 */
#define TRC_PACKETIZE_2429      2   /* endpoint uses packetization mode defined in
                                       RFC 2429 */
#define TRC_PACKETIZE_3016      3   /* endpoint uses packetization mode defined in
                                       RFC 3016 */

    tTrcEndInput    chanIn;         /* characteristics of endpoint when transcoder
                                       channel is receiving input from the endpoint */
    tTrcEndOutput   chanOut;        /* characteristics of endpoint when transcoder
                                       channel is transmitting output to the endpoint */

    U8              reserved[32];   /* reserved for future use */
} tTrcEndpoint;
```

Field listing

Field	Type	Description
vidType	U16	Type of video used by the endpoint. Valid values are: TRC_VIDTYPE_MPEG4 (Default) TRC_VIDTYPE_H263
profile	U16	Type of profile used by endpoint. Valid values are: TRC_PROFILE_SIMPLE (for MPEG-4 endpoints) TRC_PROFILE_BASELINE (for H.263 endpoints) The default is TRC_PROFILE_SIMPLE if the video type in the vidType field is TRC_VIDTYPE_MPEG4. The default is TRC_PROFILE_BASELINE if the video type in the vidType field is TRC_VIDTYPE_H263. Currently, the transcoder verifies whether the profile value is valid, but does not use this value to make transcoding decisions. Future releases may alter transcoding behavior based on the specific profile and levels indicated.
level	U16	Profile level used by endpoint. Valid values are: TRC_MPEG4_LEVEL_0 TRC_MPEG4_LEVEL_1 TRC_MPEG4_LEVEL_2 TRC_MPEG4_LEVEL_3 TRC_H263_LEVEL_10 TRC_H263_LEVEL_20 TRC_H263_LEVEL_30 The default is the most basic level supported by the video type specified in the vidType field. Currently, the transcoder verifies whether the profile level value is valid, but does not use this value to make transcoding decisions. Future releases may alter transcoding behavior based on the specific profile and levels indicated.
dataRate	U16	Video bit stream data rate in use by the endpoint. The default is 43 kbit/s. For more information, refer to *Specifying a data rate* on page 237.

Field	Type	Description
frameRate	U16	The number of video frames per second being output by or that are expected to be received by the endpoint. The default is 7 fps. For recommended values, refer to *Specifying a data rate* on page 237.
frameRes	U8	The pixel resolution of the video frame in use by the endpoint. Valid values are: TRC_FRAME_RES_QCIF - (Default). Quarter Common Interchange Format (176 x 144) TRC_FRAME_RES_CIF - Common Interchange Format (352 x 288)
packetizeMode	U8	The method that the endpoint uses for encapsulating video frame data into RTP packets. The packetization mode is specific to the video format in use by the endpoint: For an H.263 endpoint, specify either TRC_PACKETIZE_2190 (default) or TRC_PACKETIZE_2429. For an MPEG-4 endpoint, specify TRC_PACKETIZE_3016. A detailed description of each packetization mode can be found in the RFC of the corresponding number. If the default value is specified (TRC_CONFIG_DEFAULT), the packetization mode is set based on the type of endpoint. H.263 endpoints default to TRC_PACKETIZE_2190. MPEG-4 endpoints default to TRC_PACKETIZE_3016.
chanIn	Structure	Defines the endpoint characteristics that are specific to endpoints from which the transcoder is receiving input. Specified in *tTrcEndInput* on page 228.
chanOut	Structure	Defines the endpoint characteristics that are specific to endpoints to which the transcoder is transmitting output. Specified in *tTrcEndOutput* on page 230.
reserved	U8	Reserved for future use.

Specifying a data rate

For a receive endpoint, specifying the data rate is optional. The actual receive data rate is determined based on the actual receive bit stream.

For a transmit endpoint, the data rate represents the target rate of the overall outbound data bit stream. Instantaneous bit rates vary considerably, because the bit rate is a factor of the video input data. Bit rate increases according to video stream resolution, scene complexity, scene changes, and increased motion content.

For 3G-324M wireless endpoints using MPEG-4 simple profile level 0, a bit rate of 43 kbit/s is recommended. This rate allows enough room in the target 64 kbit/s channel for control information and AMR audio coded data encoded at 12.2 kbit/s.

For endpoints that do not have the bandwidth restrictions imposed by wireless connections, such as H.263 endpoints connecting over a LAN, the data rate can be extended to a value between 100 and 150 kbit/s for baseline profile level 10. An output data rate up to 384 kbit/s can accommodate the data rates required when a larger video stream resolution is in use, such as H.263 baseline profile level 30.

Recommend values

The following table lists the recommended values based on terminal type:

Terminal	Recommended values
3G-324M terminal or Video mail system destination terminal	FrameRes = TRC_FRAME_RES_QCIF outDataRate = 43 kbit/s outFrameRate = 7 fps
CIF terminal	FrameRes = TRC_FRAME_RES_CIF outDataRate = 215 kbit/s outFrameRate = 15 fps

tTrcError

Describes any error encountered by the TRC. It is part of the tTrcErrorDesc structure.

Definition

```
typedef struct
{
    U32          trcError;       /* last error encountered expressed as a TRCERR_xxx
                                    value */
    U32          osError;        /* operating system specific error code that was
                                    mapped to TRC error code */
    S8           errorInfo[TRC_ERROR_INFO_LEN];  /* ASCII description providing
                                    detailed information about the error */

    U8           reserved[32];   /* reserved for future use */
} tTrcError;
```

Field listing

Field	Type	Description
trcError	U32	Error code identifying the encountered error, expressed as a TRCERR_*xxx* value. Applicable errors are listed with their associated functions.
osError	U32	Operating system-specific error code that is mapped to the TRC error code, when applicable. A value of 0 (zero) indicates that the error is not an operating system-specific error.

Field	Type	Description
errorInfo[n]	S8	Detailed description of the error condition expressed as an ASCII string.
reserved	U8	Reserved for future use.

tTrcErrorDesc

Provides an overview of all errors detected by any channel or video transcoder platform resource being managed by the TRC. It is part of the tTrcChStatus structure and the tTrcVtpUsage structure.

Definition

```
typedef struct
{
    U32             numErrors;      /* total number of errors detected */
    tTrcError       firstError;     /* description of the first error encountered */
    tTrcError       lastError;      /* description of the last (most recent) error
                                       encountered */

    U8              reserved[32];   /* reserved for future use */
} tTrcErrorDesc;
```

Field listing

Field	Type	Description
numErrors	U32	Total number of errors detected.
firstError	Structure	Describes the first error encountered by the channel or video transcoder platform resource. Specified in *tTrcError* on page 238.
lastError	Structure	Describes the most recent error encountered by the channel or video transcoder platform resource. Specified in *tTrcError* on page 238.
reserved	U8	Reserved for future use.

tTrcMessage

Provides a TRC asynchronous event message to the call back function.

Definition

```
typedef struct
{
    U32          event;                 /* event code (TRCEVN_xxx) */
    U32          result;                /* result of request or reason for unsolicited
                                           indication (TRC_SUCCESS or TRCERR_xxx) */
    TRC_HANDLE   trcChHandle;           /* TRC channel handle associated with the event
                                           (only applicable to channel events) */
    void        *userKey;               /* user-supplied key associated with the channel
                                           (only applicable to channel events) */
    U32          data[TRCDATA_COUNT];   /* event-specific information (indexes of
                                           TRCDATA_xxx) */
```

```
    U8           reserved[32];          /* reserved for future use */
} tTrcMessage;
```

Field listing

The tTrcMessage structure contains the following fields:

Field	Type	Description
event	U32	Event code. For more information on the TRCEVN_*xxx* event codes, refer to *Transcoder resource controller events* on page 312.
result	U32	Result of the request or the reason for an unsolicited indication. Valid values are: TRC_SUCCESS TRCERR_*xxx* (when failure is indicated)
trcChHandle	Structure	TRC channel handle associated with the event. This field does not apply to non-channel events. Specified as TRC_HANDLE. For more information, refer to *trcCreateVideoChannel* on page 116.
userKey	void*	User-supplied key associated with the channel. This field does not apply to non-channel events.

Field	Type	Description
data[]	array of U32 entries	Event-specific information. The data array provides up to 16 values associated with the given event. Indexes into the data array are defined as TRCDATA_***xxx***. Valid values include: TRCEVN_IFRAME_CHANNEL_DONE event: TRCDATA_IFRAME_CHANNEL_DIRECTION - Direction that the IFRAME was generated over (TRC_DIR_SIMPLEX, TRC_DIR_FDX1 or TRC_DIR_FDX2). TRCEVN_*_OVL_DONE event (overlay done event set): TRCDATA_OVERLAY_OVLHANDLE - Contains the overlay handle. TRCDATA_OVERLAY_USERKEY - Contains the application's user key. TRCEVN_RESOURCE_CHANGE event: TRCDATA_RESOURCE_CHANGE_AVAILABLE - Number of port licenses currently available. TRCDATA_RESOURCE_CHANGE_PREVIOUS - Previous number of licenses that were available. TRCDATA_RESOURCE_OVL_LICENSE - Number of video transcoder platforms licensed for overlays. TRCDATA_RESOURCE_OVL_EXCLUSIVE - Number of video transcoder platforms that only allow channels requiring overlays. TRCEVN_CHANNEL_RTCP_BYE event: TRCDATA_CHANNEL_RTCP_ENDPOINT - Endpoint that BYE was received over (TRC_END_A or TRC_END_B). TRCDATA_CHANNEL_RTCP_DIRECTION - Direction over which BYE was received (TRC_DIR_SIMPLEX, TRC_DIR_FDX1 or TRC_DIR_FDX2). TRCDATA_CHANNEL_RTCP_REASON - Optional reason that was provided with BYE (ASCII string). TRCEVN_CHANNEL_RTCP_TIMEOUT event: TRCDATA_CHANNEL_RTCP_ENDPOINT - Endpoint that timeout condition has occurred for (TRC_END_A or TRC_END_B). TRCDATA_CHANNEL_RTCP_DIRECTION - Direction associated with endpoint connection that has timed out (TRC_DIR_SIMPLEX, TRC_DIR_FDX1 or TRC_DIR_FDX2). TRCDATA_CHANNEL_RTCP_PROTOCOL - Protocol that timeout was detected for (TRC_PROT_RTP or

Field	Type	Description
reserved	U8	Reserved for future use.

tTrcOvlBorder

Defines the overlay border.

Definition

```
typedef struct
{
    U16                 width;                      /* Border width in pixels */
    tTrcOvlColor        color;                      /* Border color*/
} tTrcOvlBorder;
```

Field listing

Field	Type	Description
width	U16	Border width in pixels.
color	Structure	Border color specified in *tTrcOvlColor* on page 242.

tTrcOvlColor

Defines the overlay color type.

Definition

```
typedef U32 tTrcOvlColor;
```

Field listing

Field	Type	Description
NA	U32	Type used to define overlay colors. The default is 0x00000000. Four byte value where each byte represents a color component: Red Green Blue Alpha (level of transparency) For more information, refer to *Customizing colors* on page 78.

tTrcOvlConfig

Describes the overlay layout. The same structure is used for text and graphic overlays.

Definition

```
typedef struct
{
   U16                  type;        /* Overlay type (See: TRC_OVL_TYPE_...)*/
```

```
   U16                 layer;       /* Overlay display order*/
   U16                 initState;   /* Initial state of the overlay
                                       (See: TRC_OVL_STATE_...)*/
   tTrcOvlCoordinates  size;        /* Overlay size */
   tTrcOvlCoordinates  position;    /* Overlay position */
   tTrcOvlColor        fgColor;     /* Default overlay foreground color */
   tTrcOvlColor        bgColor;     /* Default overlay background color */
   tTrcOvlBorder       border;      /* Overlay area border */
   tTrcOvlFont         font;        /* Default font information (only used for text
                                       overlays)*/
   U8                  reserved[32];/* Reserved area */
} tTrcOvlConfig;
```

Field listing

Field	Type	Description
type	U16	Indicates the overlay type: TRC_OVL_TYPE_MULTILINE_TEXT Multi-line text overlay. TRC_OVL_TYPE_SINGLELINE_TEXT Single line text overlay. TRC_OVL_TYPE_IMAGE Graphic overlay.
layer	U16	A positive integer used to indicate the depth at which the overlay is displayed. Overlapping overlays with a higher layer number are displayed over overlays with a lower layer number. Multiple overlays can be displayed at the same layer, but it is the application's responsibility to ensure that they do not overlap. The result of having overlapping overlays with the same layer value is unpredictable but will not generate an error.
initState	U16	Indicated the initial state of the overlay. The initial state can be either: TRC_OVL_INITSTATE_STARTED Overlay is displayed on creation (0) TRC_OVL_INITSTATE_STOPPED Overlay not displayed on creation (1)
size	Structure	Provides the height and width of the overlay. The height and width can be expressed in pixels, as a percentage of the background video frame, or based on the rendered content size. The height and width can be expressed in different units. For example, a multi-line text overlay can be defined with a width that is a percentage of the video frame width and a height that represents a number of pixels. Height and width are specified in *tTrcOvlCoordinates* on page 247.

Field	Type	Description
position	Structure	Determines the upper left hand corner of the overlay area. The horizontal and vertical position can be expressed in pixels or as a percentage of the background video frame. The height and width can be expressed in different units. For example, an overlay can be defined to be displayed at 20 percent from the top of the video frame vertically and 10 pixels from the left of the video frame horizontally. Vertical and horizontal positions are specified in *tTrcOvlCoordinates* on page 247.
fgColor and bgColor	Structure	Specifies the foreground and background color of the overlay area. Specified in *tTrcOvlColor* on page 242. Default for fgColor is white, bgColor is transparent. For more information, refer to *Predefined color values* on page 78.
border	Structure	Describes the border to be drawn around the overlay area. It is defined as a width, in pixels and a color. If the overlay is activated without any content, it is displayed as a rectangle of the background color surrounded by a border as defined by the border field. Specified in *tTrcOvlBorder* on page 242.
font	Structure	Describes the default font characteristics. The font can use a different foreground and background color than the overlay area. Use TRC_OVL_COLOR_DEFAULT to have the text use the same colors as the overlay area. Specified in *tTrcOvlFont* on page 249.
reserved	U8	Reserved for future use.

tTrcOvlContent

Specifies the type of content to display in overlays.

Definition

```
typedef struct

{

    U16                  type;            /* Content type: TRC_OVL_CONT_TYPE_... */
    tTrcOvlScroll        scroll;          /* Scrolling information*/
    U32                  options;         /* see TRC_OVL_TEXT_CONT_... and
                                             TRC_OVL_GRAPH_CONT_...*/
    tTrcOvlCoordinates   size;            /* Size of rendered content */
    S8                   *ovlData         /* Pointer to text content or image file
                                             URL string. */

} tTrcOvlContent;
```

Field listing

Field	Type	Description
type	U16	Identifies the content type. Valid values include: TRC_OVL_CONT_TYPE_TEXT Content is text. ovlData points to a text string to use as content. TRC_OVL_CONT_TYPE_GRAPHIC Content is graphic. ovlData points to a text string that represents a file name or URL of the file that must be used as content.
scroll	Structure	Scrolling information specified in *tTrcOvlScroll* on page 249. For example, type, speed, and direction.
options	U32	A bit field containing content type specific options for text content and graphic content. For a list of valid values, refer to *Options field valid values* on page 246.
size	Structure	Content dimensions specified in *tTrcOvlCoordinates* on page 247. By default, the content size is the same as the overlay area's size. Specifying a content size different from the overlay area size is useful when scrolling content. In this case, the size component associated with the direction of the scroll can be specified larger than the overlay area so that content larger than the overlay area can be scrolled through. The size of the overlay content can also be derived from the actual size of the rendered content by using the TRC_OVL_COORDINATE_UNIT_UNUSED unit type. It is also possible to have only one dimension derived from the rendered content size while setting the remaining dimension to a predefined value.

Field	Type	Description
ovlData	S8	For text content, pointer to the text string to be rendered. Carriage return and line feed characters both cause text to continue on the next line. The tab character is replaced by 8 spaces. Any other control characters or invalid character is replaced by a period (.). For graphic content, the pointer to a string that represents a URL to the image file to display in the overlay area. This URL can be in the following formats: http://<ip address of http server>/<path>/<file name.ext> file://<path>/<file name.ext> local://<**image name**> <path>/<file name.ext> If *http://* or *file://* is not specified, *file://* is assumed. The following examples show image file URLs: http://10.10.20.10/images/logo.png file://opt/nms/video/images/logo.jpg /opt/nms/video/images/logo.jpg *local://**NMS_N_32x32*** where ***NMS_N_32x32*** is the name assigned to a cached image in *trcr.cfg*.

Options field valid values

Use the following values to specify a value for the options field:

Value	Description
TRC_OVL_TEXT_CONT_WRAP_WORD	Only applies to multi-line text overlays. Causes words to be displayed on a following line if they cannot fit on the current line. If this option is not specified, words are clipped at the end of the line.
TRC_OVL_TEXT_CONT_ALIGN_LEFT	Text is aligned left in overlay area.
TRC_OVL_TEXT_CONT_ALIGN_RIGHT	Text is aligned right in overlay area.
TRC_OVL_TEXT_CONT_ALIGN_CENTER	Text is centered in overlay area.

Value	Description
TRC_OVL_GRAPH_CONT_FIT_HORIZONTAL	Resize large image to fit overlay horizontally. Resize is always proportional, so the image will also be resized vertically. Resulting image is clipped vertically if it is larger that the overlay area.
TRC_OVL_GRAPH_CONT_FIT_VERTICAL	Resize large image to fit overlay vertically. Resize is always proportional, so the image will also be resized horizontally. Resulting image is clipped horizontally if it is wider that the overlay area. An application can ensure that an image can be completely displayed in the overlay area by requesting that it be resized both horizontally and vertically.
TRC_OVL_GRAPH_CONT_FIT	Resize large image to fit overlay. Either the horizontal or vertical resize will be used, which ever allows the full image to display.
TRC_OVL_GRAPH_CONT_CENTER	Center image in overlay area without any resizing. Any image part exceeding the overlay area will be clipped. This option must be used with cached images because they cannot be resized.

tTrcOvlCoordinates

Defines a flexible coordinates structure used with overlays. This structure is used for any values that require a horizontal coordinate, a vertical coordinate, or both (for example, position, size, scroll direction).

Definition

```
typedef struct
{
    U16                 xUnit;   /* Horizontal coordinate unit
                                    (See: TRC_OVL_COORDINATE_UNIT_...)*/
    U16                 yUnit;   /* Vertical coordinate unit
                                    (See: TRC_OVL_COORDINATE_UNIT_...)*/
    S32                 x;       /* Horizontal coordinate */
    S32                 y;       /* Vertical coordinate */
} tTrcOvlCoordinates;
```

Field listing

Field	Type	Description
xUnit	U16	Horizontal coordinate unit. For a list of valid values, refer to the Overlay coordinate units table.
yUnit	U16	Vertical coordinate unit. For a list of valid values, refer to the Overlay coordinate units table.
x	S32	Horizontal coordinate.
y	S32	Vertical coordinate.

Overlay coordinate units

Coordinate unit	Value	Description
TRC_OVL_COORDINATE_UNIT_UNUSED	0	The coordinate is unused or must be determined by the transcoder. For example, this unit type is used to indicate that an overlay's size must be based on its associated content size.
TRC_OVL_COORDINATE_UNIT_PIXEL	1	Coordinate is given in pixels.
TRC_OVL_COORDINATE_UNIT_PERCENT	2	Coordinate represents a percentage of the video frame's resolution or of the overlay area.
TRC_OVL_COORDINATE_UNIT_LINECOL	3	Reserved for future use.
TRC_OVL_COORDINATE_UNIT_MS_PER_PIXEL	4	Coordinate represents the number of milliseconds per pixel.

tTrcOvlFont

Defines the font used in an overlay.

Definition

```
typedef struct
{
    S8                  name[TRC_OVL_FONTNAME_MAX]; /* Name of the font */
    U16                 style;                      /* Normal, Bold, Italic, etc.
                                                       (See: TRC_OVL_FONTSTYLE_...) */
    U16                 size;                       /* Font size */
    tTrcOvlColor        fgColor;                    /* Foreground color*/
    tTrcOvlColor        bgColor;                    /* Background color*/
} tTrcOvlFont;
```

Field listing

Field	Type	Description
name	S8	Name of the font.
style	U16	Font style. See TRC_OVL_FONTSTYLE. For more information, refer to *Customizing the font style* on page 79.
size	U16	Font size.
fgColor	Structure	Foreground color. Specified in *tTrcOvlColor* on page 242.
bgColor	Structure	Background color. Specified in *tTrcOvlColor* on page 242.

tTrcOvlScroll

Defines the overlay content scrolling.

Definition

```
typedef struct

{

    U16                 type;           /* Scrolling type (See: TRC_OVL_SCROLL_...)*/
    tTrcOvlCoordinates  speedDir;       /* Speed and direction of scrolling */

} tTrcOvlScroll;
```

Field listing

Field	Type	Description
type	U16	Determines the type of scrolling to apply to the content. Valid values include: TRC_OVL_SCROLL_TYPE_NONE No scrolling. The content that can fit the overlay area is displayed and the rest is clipped. TRC_OVL_SCROLL_TYPE_CONTENT Content is scrolled up to the end of the available content at a rate indicated by speedDir and stops. Scrolling continues when new content is added. TRC_OVL_SCROLL_TYPE_CONTINOUS Content is scrolled at a rate indicated by speedDir until the last line of available content disappears at which point scrolling starts over from the beginning of the content.
speedDir	Structure	Speed and direction of the scrolling defined in *tTrcOvlCoordinates* on page 247. The scrolling direction is described by using either the x or y (setting the according *x*Unit/*y*Unit to a value other than TRC_OVL_COORDINATE_UNIT_UNUSED). Using the x coordinate indicates horizontal scrolling, with positive values representing left to right and negative values representing right to left scrolling. Using the y coordinate indicates vertical scrolling, with positive values representing bottom to top and negative values representing top to bottom scrolling. For example, to create a marquee style overlay that scrolls the text horizontally from right to left, use the following tTrcOvlCoordinates definition: x = -100 xUnit = TRC_OVL_COORDINATE_UNIT_MS_PER_PIXEL y = 0· yUnit = TRC_OVL_COORDINATE_UNIT_UNUSED The text for this definition scrolls the text right to left at a rate of 10 pixels per second.

Using overlay coordinate units for scrolling

The following table describes how to use the coordinate units (***x***Unit and ***y***Unit) for scrolling:

Unit	Description
TRC_OVL_COORDINATE_UNIT_UNUSED	Indicates that there is no scrolling in the direction associated with the coordinate (*x*=horizontal, *y*=vertical). With content-based or continuous scrolling one of the coordinates must be set to this unit type.
TRC_OVL_COORDINATE_UNIT_PIXEL	This unit type is not used for scrolling.
TRC_OVL_COORDINATE_UNIT_PERCENT	This unit type is not used for scrolling.
TRC_OVL_COORDINATE_UNIT_MS_PER_PIXEL	Defines the scrolling rate as a number of milliseconds per pixel. For example, using a y coordinate of 1500 ms/pixel has the overlay scrolling up by one pixel every 1.5 seconds.

tTrcRes

Describes assigned resource information for a channel. It is used with **trcInfoVideoChannel**, as part of the tTrcChInfo structure.

Definition

```
typedef struct
{
    S8          ipAddr[TRC_IPADDR_LEN]; /* IP address of VTP providing transcoder
                                           resources */
    U16         rxPort[TRC_DIR_COUNT];  /* assigned transcoder RTP receive port(s) */

    U8          reserved[32];   /* reserved for future use */
} tTrcRes;
```

Field listing

Field	Type	Description
ipAddr	S8	Assigned transcoder address expressed as an ASCII string.

Field	Type	Description
rxPort[]	array of U16 values	Assigned transcoder RTP receive ports. Valid index values are: TRC_DIR_SIMPLEX - Only port used for simplex channels. TRC_DIR_FDX1 - RTP port on which the transcoder receives a video bit stream being output from the first endpoint of a full-duplex connection (endpoint A). TRC_DIR_FDX2 - RTP port on which the transcoder receives a video bit stream being output from the second endpoint of a full-duplex connection (endpoint B).
reserved	U8	Reserved for future use.

tTrcUsage

Describes usage information for the current channel. It is used with **trcUsage**.

Definition

```
typedef struct
{
    U16      licensesAvail;      /* total number of transcoder port licenses
                                    available */
    U16      vtpDefined;         /* total number of VTPs defined for this instance
                                    of the TRC */
    U16      vtpAvail;           /* total number of VTPs that are currently
                                   available to the TRC */
    U16      simplexLocal;       /* number of simplex channels currently in use by
                                    the calling application */
    U16      fdxLocal;           /* number of full-duplex channels currently in use
                                    by the calling application */
    U16      simplexTotal;       /* number of simplex channels currently in use by
                                    all  applications sharing the same VTP resources */
    U16      fdxTotal;           /* number of full-duplex channels currently in use
                                    by all applications sharing the same VTP resources */
    U16      ovlLicense;         /* number of vtps with an overlay license */
    U16      ovlExclusive;       /* number of vtps that only accept channel that
                                    request the overlay option */
    U8       reserved[28];       /* reserved for future use */
} tTrcUsage;
```

Field listing

Field	Type	Description
licensesAvail	U16	Total number of transcoder port licenses that are available to the application and all other applications sharing the same video transcoder platform resources. Each created simplex channel uses one port license while each full-duplex channel uses two licenses.
vtpDefined	U16	Total number of video transcoder platforms defined for this instance of the TRC.

Field	Type	Description
vtpAvail	U16	Total number of video transcoder platforms that are currently available to the TRC for transcoder resource assignments. A video transcoder platform is considered available once the TRC has completely connected to the video transcoder platform. The video transcoder platform is considered available even in the case in which all transcoder resources are in use.
simplexLocal	U16	Number of simplex channels currently in use by the calling application.
fdxLocal	U16	Number of full-duplex channels currently in use by the calling application.
simplexTotal	U16	Total number of simplex channels currently in use by all applications sharing the same video transcoder platform resources.
fdxTotal	U16	Total number of full-duplex channels currently in use by all applications sharing the same video transcoder platform resources.
ovlLicense	U16	Number of video transcoder platforms with an overlay license.
ovlExclusive	U16	Number of video transcoder platforms that are dedicated to channels with the overlay option.
reserved	U8	Reserved for future use.

tTrcVtpAll

Provides a summary view of all video transcoder platforms currently in use by the TRC. This structure is used with **trcVTPStatus**.

Definition

```
typedef struct
{
    U16             vtpDefined;             /* total number of VTPs that have been
                                               defined for this instance of the TRC */
    tTrcVtpSummary  vtp[TRC_MAX_VTPS];      /* summary-level status information for
                                               each defined VTP */
    U8              reserved[32];           /* reserved for future use */
} tTrcVtpAll;
```

Field listing

Field	Type	Description
vtpDefined	U16	Total number of video transcoder platforms that are defined for this instance of the TRC.

Field	Type	Description
vtp[n]	Structure	Array that provides status information for each defined video transcoder platform. Specified in *tTrcVtpSummary* on page 254.
reserved	U8	Reserved for future use.

tTrcVtpSummary

Provides summary information for each defined video transcoder platform. It is used with **trcVTPStatus** as part of the tTrcVtpAll structure.

Definition

```
typedef struct
{
    U32             vtpId;           /* identifies the VTP that the channel has been
                                        assigned to */
    U16             state;           /* current state of the connection to the VTP
                                        (TRCVTP_STATE_xxx) */
    U16             simplexLocal;    /* total number of simplex channels (in use by
                                        calling application) assigned to the given VTP */
    U16             fdxLocal;        /* total number of full duplex channels (in use by
                                        calling application) assigned to the given VTP */
    tTrcVtpUsage    usage;           /* total channel usage information and error
                                        information for the given VTP */
    U8              reserved[32];    /* reserved for future use */
} tTrcVtpSummary;
```

Field listing

Field	Type	Description
vtpId	U32	TRC-defined unique identifier for the given video transcoder platform.

Field	Type	Description
state	U16	Current state of the connection to the video transcoder platform. Valid state values are: TRCVTP_STATE_CONFIGURING Passing configuration information to the video transcoder platform. TRCVTP_STATE_CONNECT_LOST Connection to video transcoder platform was lost. The connection had been established previously, but a connection problem has occurred. The TRC is attempting to re-connect. TRCVTP_STATE_CONNECTING Attempting to establish TCP/IP connection. TRCVTP_STATE_DISABLED Connected to the video transcoder platform, which is disabled and not allowing new channels. TRCVTP_STATE_ENABLED Connected to the video transcoder platform, which is ready to create channels (and may already have channels). TRCVTP_STATE_RECONNECTING Connection was lost and has now been re-established. The TRC is communicating with the video transcoder platform to synchronize channel states and any configuration information. RCVTP_STATE_SHUTTING_DOWN TRC is shutting down this video transcoder platform connection due to a **trcShutdown** call.
simplexLocal	U16	Total number of simplex channels in use by the calling application that are assigned to the given video transcoder platform.
fdxLocal	U16	Total number of full-duplex channels in use by the calling application that are assigned to the given video transcoder platform.
usage	Structure	Describes total channel usage and error information for the given video transcoder platform. Specified in *tTrcVtpUsage* on page 256.
reserved	U8	Reserved for future use.

tTrcVtpUsage

Describes usage information for a particular video transcoder platform in use by the TRC. It is used with **trcVTPStatus**, as part of the tTrcVtpSummary structure.

Definition

```
typedef struct
{
    U16             licensesAvail;  /* total transcoder port licenses available */
    U16             simplexInUse;   /* total simplex channels currently in use by all
                                       applications  sharing the same VTP */
    U16             fdxInUse;       /* total full duplex channels currently in use by
                                       all applications sharing the same VTP */
    tTrcErrorDesc   errorDesc;      /* description of any VTP-level errors encountered */
    U8              reserved[32];   /* reserved for future use */
} tTrcVtpUsage;
```

Field listing

Field	Type	Description
licensesAvail	U16	Total number of transcoder port licenses that are available to the application through the given video transcoder platform resource. Each created simplex channel uses one port license, while each full-duplex channel uses two licenses.
simplexInUse	U16	Total number of simplex channels in use by all applications sharing the same video transcoder platform.
fdxInUse	U16	Total number of full-duplex channels in use by all applications sharing the same video transcoder platform.
errorDesc	Structure	Information related to any errors encountered by the given video transcoder platform. Specified in *tTrcErrorDesc* on page 239.
reserved	U8	Reserved for future use.

11. Management structures

Management structures overview

This section provides an alphabetical reference to the management structures. It defines each structure and provides a description of the fields in the structure.

Note: Fields marked available or reserved are for future modifications and should be zero-filled.

Requests

Management requests are always issued to a particular video transcoder platform as specified in the VTMNG_VTPADDR structure. The only other structures involved in issuing management requests are the entity configuration structures that are used to modify an entity:

- VTMNG_VTP_CFG
- VTMNG_APP_CFG
- VTMNG_MON_CFG
- VTMNG_CHN_CFG

For more information, refer to the **vtMngSet*xxx*** functions.

Modifications

When modifying a configuration, two copies of the given structure are provided. The first copy holds the values to be set and the second copy provides a mask identifying which specific fields are being set.

Responses

All management responses provide the:

- Addressing information about the video transcoder platform that the request was received from. This information is included in the VTMNG_VTPADDR structure.
- Complete management response message in the VT_MNG_MSG structure.

Each management response also provides other structures that have a direct relationship to the type of request that was issued:

- For the **vtMngGet*xxx*List** functions, the VTMNG_ENT_ID structure points to the first entity in the response set.
- The **vtMngGet*xxx*** functions retrieve a specific entity. The configuration, status, and statistics structures are provided for that entity.
- The **vtMngGet*xxx*** functions that request statistics are passed the VTMNG_ST_ENTRY structure.
- The VTMNG_ST_ENTRY structure is provided in response to the **vtMngZeroTotal** request. All other responses to zero statistics are provided with the given entity's statistics structure: VTMNG_VTP_STATS, VTMNG_APP_STATS, VTMNG_MON_STATS or VTMNG_CHN_STATS.

Notifications

All management asynchronous notifications (traps) provide:

- Addressing information for the video transcoder platform that issues the trap. This information is provided in the VTMNG_VTPADDR structure.
- Information that is common to all trap types is provided in the VTMNG_NOTIF_INFO structure.
- Information that is common to all management messages. All management messages are defined by the VTMNG_MNG_MSG structure.

Each management response also provides a structure representing the information that is specific to the type of trap:

Structure	Type of trap
VTMNG_VTPLVL_NOTIF	Video transcoder platform-level traps.
VTMNG_APPCON_NOTIF	Application-related traps.
VTMNG_MONPROC_NOTIF	Monitored process-related traps.
VTMNG_CHN_<*type*>_NOTIF	Channel-specific trap notifications.

Management structures summary

The following tables list the management structures and the functions (or higher-level structures) that use each structure:

Message routing control structures

Structures	Description	Components that use this structure
VTMNG_VTPADDR	Provides video transcoder platform addressing information. For requests, this is the video transcoder platform destination address. For responses and traps, this is the video transcoder platform source address.	All request functions. All responses. All trap notifications.
VTMNG_UPCALLS	Defines the upcall function set used by the VTMNG API to route received management messages to the proper handling function.	vtMngInit

Lists of entities and specific entity identification structures

Structures	Description	Components that use this structure
VTMNG_OBJ_ID	Minimal information to uniquely identify any managed object (entity).	Part of every request, response, and notification message.
VTMNG_ENT_ID	Fully identifies a given entity.	All list responses.

Video transcoder platform-level information structures

Structures	Description	Components that use this structure
VTMNG_VTP_CFG	Contains all video transcoder platform-level configuration.	vtMngSetVtp vtMngSetVtpRsp vtMngGetVtpRsp vtMngEventVtpRsp
VTMNG_VTP_STATUS	Current video transcoder platform status information.	vtMngGetVtpRsp
VTMNG_VTP_STATS	Set of statistics maintained at the video transcoder platform-level.	vtMngGetVtpRsp vtMngZeroVtpRsp
VTMNG_VTP_ENTITY	Contains all video transcoder platform-level information.	Used in VT_MNG_MSG.

Controlling application-specific information structures

Structures	Description	Components that use this structure
VTMNG_APP_CFG	Configuration information for a given application.	vtMngSetApp vtMngSetAppRsp vtMngGetAppRsp vtMngEventAppRsp
VTMNG_APP_STATUS	Current status of a given application.	vtMngGetAppRsp
VTMNG_APP_STATS	Statistics maintained for a given application.	vtMngGetAppRsp vtMngZeroAppRsp
VTMNG_APP_ENTITY	Contains all information for a given application.	Used in VT_MNG_MSG.

Monitored process-specific information structures

Structures	Description	Components that use this structure
VTMNG_MON_VAR	Holds a monitored process variable definition.	Used in VTMNG_MON_CFG.
VTMNG_MON_CFG	Configuration information for a given monitored process.	vtMngSetMon vtMngSetMonRsp vtMngGetMonRsp vtMngEventMonRsp
VTMNG_MON_STATUS	Current status of a given monitored process.	vtMngGetMonRsp
VTMNG_MON_STATS	Statistics maintained for a given monitored process.	vtMngGetMonRsp vtMngZeroMonRsp
VTMNG_MON_ENTITY	Contains all information for a given monitored process.	Used in VT_MNG_MSG.

Current, total, and histogram statistics structures

Structures	Description	Components that use this structure
VTMNG_ST_COMM	Statistics substructure containing data communication related statistics.	Used in VTMNG_ST_DIR and VTMNG_CHN_RTP_RXTX.
VTMNG_ST_VTYPE	Statistics substructure containing video-type related statistics.	Used in VTMNG_ST_DIR and VTMNG_CHN_VTYPE_RXTX.
VTMNG_ST_DIR	Statistics substructure that holds a set of statistics being maintained per direction of data flow.	Used in the receive and transmit portion of VTMNG_ST_ENTRY.
VTMNG_ST_ENTRY	Common statistics record providing video transcoder platform-level statistics as well as summary statistics for receive and transmit related information.	vtMngGetStTotalRsp vtMngGetStCurrMinRsp vtMngGetHistPerMinRsp vtMngGetHistPerHHrRsp vtMngZeroTotalRsp

Remote video endpoint information structures

Structures	Description	Components that use this structure
VTMNG_RTCP_INFO	RTCP-specific addressing information provided for any endpoint participating in an RTCP communication session.	Used as a sub-section of VTMNG_ADDR_INFO.
VTMNG_ADDR_INFO	Full addressing information provided as part of the status information for each remote receive and transmit endpoint involved in a channel.	Used in VTMNG_CHN_STATUS.

Video transcoder channel-specific information structures

Structures	Description	Components that use this structure
VTMNG_CHN_CFG	Channel configuration information.	vtMngSetChn vtMngSetChnRsp vtMngGetChnRsp vtMngEventChnRsp
VTMNG_CHN_STATUS	Current channel status information.	vtMngGetChnRsp
VTMNG_CHN_RTP_RXTX	RTP-related statistics that are maintained for both receive and transmit directions.	Used in VTMNG_CHN_RTP.
VTMNG_CHN_RTP	Overall RTP information.	Used in VTMNG_CHN_END.
VTMNG_CHN_RTCP_RXTX	RTCP-related statistics that are maintained for both receive and transmit directions.	Used in VTMNG_CHN_RTCPE.
VTMNG_CHN_RTCPE	RTCP information maintained for both the input half and output half of each RTCP Translator endpoint.	Used in VTMNG_CHN_END.
VTMNG_CHN_VTYPE_RXTX	Video type-specific statistics that are maintained for both receive and transmit directions.	Used in VTMNG_CHN_VTYPE.
VTMNG_CHN_VTYPE	Video type-specific information maintained for both receive and transmit directions.	Used in VTMNG_CHN_END.
VTMNG_CHN_END	Set of all statistics maintained for each channel endpoint (A and B).	Used in VTMNG_CHN_STATS.
VTMNG_CHN_STATS	Complete statistical information for a given channel.	vtMngGetChnRsp vtMngZeroChnRsp
VTMNG_CHN_ENTITY	Contains all information for a given channel.	Used in VT_MNG_MSG.

Request-specific structures

Structure	Description	Components that use this structure
VTMNG_SETCFG_REQ	Set of all requests that perform configuration changes.	Used in VTMNG_REQ_SET.

Response-specific structures

Structures	Description	Components that use this structure
VTMNG_RSP_INFO	Common response information.	Used in all response messages.
VTMNG_GETLIST_RSP	Common response to any request for a list of entities.	Used in VTMNG_RSP_SET.
VTMNG_GETENT_RSP	Set of all responses that provide complete information for a specific entity.	Used in VTMNG_RSP_SET.
VTMNG_SETCFG_RSP	Set of all responses to configuration modification requests.	Used in VTMNG_RSP_SET.
VTMNG_ZEROSTATS_RSP	Set of all responses to requests to zero current statistics.	Used in VTMNG_RSP_SET.

Asynchronous notifications (traps) structures

Structures	Description	Components that use this structure
VTMNG_NOTIF_INFO	Common notification information.	Used in all notification messages.
VTMNG_VTPLVL_NOTIF	Information provided by notifications indicating video transcoder platform-level thresholds being crossed.	vtMngVtpLevelTrap
VTMNG_APPCON_NOTIF	Information provided by notifications indicating a change in the connection status of a given control application.	vtMngAppConnTrap

Structures	Description	Components that use this structure
VTMNG_MONPROC_NOTIF	Information provided by notifications indicating a change in state of a monitored process.	vtMngMonProcTrap
VTMNG_CHN_CREATE_NOTIF	Information known about a channel at the time it is created.	vtMngChnCreateTrap
VTMNG_CHN_START_NOTIF	Full channel configuration and status as channel is started.	vtMngChnStartTrap
VTMNG_CHN_STOP_NOTIF	Full channel configuration, status and final overall channel statistics as channel is stopped.	vtMngChnStopTrap
VTMNG_CHN_DEAD_NOTIF	Identification of channel that has been destroyed.	vtMngChnDeadTrap
VTMNG_CHN_ERROR_NOTIF	Information providing snapshot of channel statistics at the time an error was encountered.	vtMngChnErrorTrap

Common management message structures

Structures	**Description**	**Components that use this structure**
VTMNG_HDR	Header common to all messages.	Used in all message types.
VTMNG_REQ_MSG	Information common to all requests.	Used in all request messages.
VTMNG_RSP_MSG	Information common to all responses.	Used in all response messages.
VTMNG_NOTIF_MSG	Information common to all asynchronous notifications (traps).	Used in all notification messages.
VTMNG_REQ_SET	Set of all defined management request messages.	Used in VT_MNG_MSG.
VTMNG_RSP_SET	Set of all defined management response messages.	Used in VT_MNG_MSG.

Structures	Description	Components that use this structure
VTMNG_NOTIF_SET	Set of all defined management notifications messages.	Used in VT_MNG_MSG.
VT_MNG_MSG	Top-level management message that defines every type of message as a single union.	All responses. All trap notifications. **vtMngMsg2Network** **vtMngMsg2Host**

VTMNG_ADDR_INFO

Full addressing information provided as part of the status information for each remote receive and transmit endpoint involved in a channel.

Definition

```
typedef struct __vtMng_Addr_Info
{
    S8              ipAddr[VTMNG_NAME_SZ];          /* IP address */
    U32             udpPort;                        /* UDP port number */
    U32             rtpSsrc;                        /* RTP/RTCP synchronization source */
    VTMNG_RTCP_INFO rtcp;                           /* RTCP-specific information */
    U8              reserved[16];                   /* reserved for future use */
} VTMNG_ADDR_INFO;
```

Fields

Field	Type	Description
ipAddr	S8	IP address expressed as a NULL-terminated ASCII string. For example: 127.0.0.1
udpPort	U32	UDP port number.
rtpSsrc	U32	RTP/RTCP synchronization source.
rtcp	Structure	RTCP-specific information. Specified in *VTMNG_RTCP_INFO* on page 291.
reserved	U8	Reserved for future use.

VTMNG_APP_CFG

Configuration information for a given application.

Definition

```
typedef struct __vtMng_App_Cfg
{
    /* configuration elements that are set internally */
    U32         appUnique;                      /* unique value ID'ing this application
                                                   connection */
    S8          appName[VTMNG_NAME_SZ];         /* name of application */
    S8          appHost[VTMNG_LONGNAME_SZ];     /* name of host that application
                                                   executes on */
```

```
    U8          reserved[16];                   /* reserved for future use */
} VTMNG_APP_CFG;
```

Fields

Field	Type	Description
appUnique	U32	Unique value identifying this application connection.
appName	S8	Name of application.
appHost	S8	Name of host that application executes on.
reserved	U8	Reserved for future use.

VTMNG_APP_ENTITY

Contains all information for a given application.

Definition

```
typedef struct __vtMng_App_Entity
{
    VTMNG_APP_CFG       cfg;            /* application connection configuration */
    VTMNG_APP_STATUS    status;         /* current application connection status */
    VTMNG_APP_STATS     stats;          /* set of statistics maintained per
                                           application connection */
    U8                  reserved[16];   /* reserved for future use */
} VTMNG_APP_ENTITY;
```

Fields

Field	Type	Description
cfg	Structure	Application connection configuration. Specified in *VTMNG_APP_CFG* on page 265.
status	Structure	Current application connection status. Specified in *VTMNG_APP_STATUS* on page 267.
stats	Structure	Set of statistics maintained per application connection. Specified in *VTMNG_APP_STATS* on page 267.
reserved	U8	Reserved for future use.

VTMNG_APP_STATS

Statistics maintained for a given application.

Definition

```
typedef struct __vtMng_App_Stats
{
    U32             appSpxInUse;            /* current number of simplex channels
                                               in use by application */
    U32             appFdxInUse;            /* current number of full-duplex
                                               channels in use by application */
    U8              reserved[16];           /* reserved for future use */
} VTMNG_APP_STATS;
```

Fields

Field	Type	Description
appSpxInUse	U32	Current number of simplex channels in use by application.
appFdxInUse	U32	Current number of full-duplex channels in use by application.
reserved	U8	Reserved for future use.

VTMNG_APP_STATUS

Current status of a given application.

Definition

```
typedef struct __vtMng_App_Status
{
    U8              appState;           /* current application connection
                                           state (VTMNG_APP_S_xxx) */
    U8              avail[3];
    VSLOG_TIME      appStartTime;       /* time when application first connected */
    U8              reserved[16];       /* reserved for future use */
} VTMNG_APP_STATUS;
```

Fields

Field	Type	Description
appState	U8	Current application connection state (VTMNG_APP_S_*xxx*).
avail	U8	Available for future use.
appStartTime	Structure	Time when application first connected. Uses the VSLOG_TIME record to represent time stamps provided through the VTMNG.
reserved	U8	Reserved for future use.

VTMNG_APPCON_NOTIF

Information provided by notifications indicating a change in the connection status of a given control application.

Definition

```
typedef struct __vtMngAppCon_Notif
{
    VTMNG_NOTIF_MSG      common;         /* common portion */
    VTMNG_APP_ENTITY     app;            /* application information */
    U8                   reserved[16];   /* reserved for future use */

} VTMNG_APPCON_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
app	Structure	Application information. Specified in *VTMNG_APP_ENTITY* on page 266.
reserved	U8	Reserved for future use.

VTMNG_CHN_CFG

Channel configuration information.

Definition

```
typedef struct __vtMng_Chn_Cfg
{
    /* configuration elements that can be set by external manager */
    U8              chnEvent;               /* optional event to issue to the channel
                                               (VTMNG_CHN_E_xxx) */
    U8              avail[3];
    U32             chnDebugMask;           /* channel-specific debug log mask (set
                                               of VSLOG_xxx bits) */

    /* configuration elements that are set internally */
    U32             appUnique;              /* unique value ID'ing the owning
                                               application (same as APP's appUnique) */
    U32             chnUnique;              /* unique value ID'ing this channel */
    S8              chnName[VTMNG_NAME_SZ]; /* optional name applied to channel
                                               (assigned by controlling application) */
    U16             chnBasicType;           /* type of channel (specified when channel
                                               created) (TRC_CH_SIMPLEX|TRC_CH_FDX) */
    U16             chnRequire;             /* Channel feature requirements
                                               ( TRC_CH_OVERLAY, TRC_CH_RTCP) */
    tTrcEndpoint    endpointA;              /* description of endpoint A configuration */
    tTrcEndpoint    endpointB;              /* description of endpoint B configuration */
    U32             trcpUnique;             /* unique value ID'ing the transcoder
                                               process in use by this channel */
                                            /* NOTE: trcp-specific log filename =
                                               xc_log_<trcpUnique> (as %02d) */
    U32             trcpProcess;            /* process ID of the transcoder process
                                               in use by this channel */
    U32             trcpDemux;              /* value used to demux this channel's
                                               messages from trcp connection */
    U8              reserved[16];           /* reserved for future use */
} VTMNG_CHN_CFG;
```

Fields

Field	Type	Description
chnEvent	U8	Optional event to issue to the channel (VTMNG_CHN_E_*xxx*).
avail	U8	Available for future use.
chnDebugMask	U32	Channel-specific debug log mask. Note: This field is reserved for future use. No channel-specific debug logging currently exists. To control debug tracing for transcoder processes, use the video transcoder platform-level trcpLogMask.
appUnique	U32	Unique value identifying the owning application. This value is the same as the appUnique value for the application.
chnUnique	U32	Unique value identifying this channel.
chnName	S8	Optional name applied to channel that is assigned by the controlling application.

Field	Type	Description
chnBasicType	U16	Type of channel specified when the channel was created. Valid values: TRC_CH_SIMPLEX TRC_CH_FDX
chnRequire	U16	Channel feature requirements (TRC_CH_OVERLAY, TRC_CH_RTCP).
endpointA	Structure	Description of endpoint A configuration. Specified in *tTrcEndpoint* on page 235.
endpointB	Structure	Description of endpoint B configuration. Specified in *tTrcEndpoint* on page 235.
trcpUnique	U32	Unique value identifying the transcoder process in use by this channel. The trcp-specific log filename is equal to: xc.log_<***trcpUnique***> (as %02d).
trcpProcess	U32	Process ID of the transcoder process in use by this channel.
trcpDemux	U32	Value used to demux this channel's messages from the trcp connection.
reserved	U8	Reserved for future use.

VTMNG_CHN_CREATE_NOTIF

Information known about a channel at the time it is created.

Definition

```
typedef struct __vtMngChnCreate_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    U32                appUnique;      /* unique value ID'ing the owning application
                                          (same as APP's appUnique) */
    U32                chnUnique;      /* unique value ID'ing this channel */
    S8                 chnName[VTMNG_NAME_SZ]; /* optional name applied to channel
                                                  (assigned by controlling application) */
    U16                chnBasicType;   /* basic channel type
                                          (TRC_CH_SIMPLEX|TRC_CH_FDX) */
    U16                chnRequire;     /* channel featre requirements
                                          (TRC_CH_OVERLAY, TRC_CH_RTCP) */
    U8                 reserved[16];   /* reserved for future use */
} VTMNG_CHN_CREATE_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion specified in *VTMNG_NOTIF_MSG* on page 288.
appUnique	U32	Unique value that identifies the owning application. This is the same as the appUnique for the owning application.
chnUnique	U32	Unique value that identifies this channel.
chnName	S8	Optional name applied to this channel. Assigned by the controlling application.
chnBasicType	U16	Basic channel type. Valid values: TRC_CH_SIMPLEX TRC_CH_FDX
chnRequire	U16	Channel feature requirements: TRC_CH_OVERLAY TRC_CH_RTCP
reserved	U8	Reserved for future use.

VTMNG_CHN_DEAD_NOTIF

Identification of a channel that has been destroyed.

Definition

```
typedef struct __vtMngChnDead_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    U8                  reserved[16];   /* reserved for future use */
} VTMNG_CHN_DEAD_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
reserved	U8	Reserved for future use.

VTMNG_CHN_END

Set of all statistics maintained for each channel endpoint (A and B).

Definition

```
typedef struct __vtMng_Chn_End
{
    VTMNG_CHN_RTP    rtp;              /* UDP-level stats */
```

```
    VTMNG_CHN_VTYPE vtype;              /* video type-specific stats */
    VTMNG_CHN_RTCPE rtcpInput;          /* RTCP stats for input half of RTCP Translator */
    VTMNG_CHN_RTCPE rtcpOutput;         /* RTCP stats for output half of RTCP Translator */
    U8              reserved[16];       /* reserved for future use */
} VTMNG_CHN_END;
```

Fields

Field	Type	Description
rtp	Structure	UDP-level statistics. Specified in *VTMNG_CHN_RTP* on page 275.
vtype	Structure	Video type-specific statistics. Specified in *VTMNG_CHN_VTYPE* on page 279.
rtcpInput	Structure	RTCP statistics for input half of RTCP translator. Specified in *VTMNG_CHN_RTCPE* on page 275.
rtcpOutput	Structure	RTCP statistics for output half of RTCP translator. Specified in *VTMNG_CHN_RTCPE* on page 275.
reserved	U8	Reserved for future use.

VTMNG_CHN_ENTITY

Contains all information for a given channel.

Definition

```
typedef struct __vtMng_Chn_Entity
{
    VTMNG_CHN_CFG       cfg;        /* channel configuration */
    VTMNG_CHN_STATUS    status;     /* current channel status information */
    VTMNG_CHN_STATS     stats;      /* set of statistics maintained by channel */
    U8                  reserved[16];   /* reserved for future use */
} VTMNG_CHN_ENTITY;
```

Fields

Field	Type	Description
cfg	Structure	Channel configuration. Specified in *VTMNG_CHN_CFG* on page 269.
status	Structure	Current channel status information. Specified in *VTMNG_CHN_STATUS* on page 277.
stats	Structure	Set of statistics maintained by channel. Specified in *VTMNG_CHN_STATS* on page 276.
reserved	U8	Reserved for future use.

VTMNG_CHN_ERROR_NOTIF

Information providing snapshot of channel statistics at the time an error was encountered.

Definition

```
typedef struct __vtMngChnError_Notif
{
    VTMNG_NOTIF_MSG     common;          /* common portion */

    VTMNG_CHN_STATS     stats;           /* set of statistics maintained by channel */
} VTMNG_CHN_ERROR_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
stats	Structure	Set of statistics maintained by channel at the time the error occurred. Specified in *VTMNG_CHN_STATS* on page 276.

VTMNG_CHN_RTCP_RXTX

RTCP-related statistics that are maintained for both receive and transmit directions.

Definition

```
typedef struct __vtMng_Chn_Rtcp_RxTx
{
    /* sender info */
    U32             ntpMostSig;         /* most significant 32-bits of NTP timestamp */
    U32             ntpLeastSig;        /* least significant 32-bits of NTP timestamp */
    U32             rtpTimestamp;       /* RTP timestamp associated with NTP timestamp */
    U32             senderPktCount;     /* total number of packets sent (as reported by
                                           the sender) */
    U32             senderOctetCount;   /* total number of payload bytes sent
                                           [no headers/padding] (as reported by the
                                           sender) */

    /* reception report block */
    U8              fractionLost;       /* fraction of RTP data packets lost since
                                           previous SR or RR was sent */
    U8              avail1[3];
    U32             cumulativePktsLost; /* total number of RTP data packets that
                                           have been lost */
    U32             extendedHighSeqNo;  /* low 16-bits = highest sequence number
                                           received; high 16-bits = count of cycles */
    U32             interarrivalJitter; /* estimate of statistical variance of RTP
                                           data packet interarrival time */
    U32             lastSr;             /* (LSR) middle 32-bits of NTP timestamp
                                           from most recently received SR */
    U32             delaySinceLastSr;   /* delay between receiving last SR and
                                           sending given reception report block */

    /* RTCP packet counters */
    U32             rtcpSr;             /* sender report packet count */
    U32             rtcpRr;             /* receiver report packet count */
    U32             rtcpSdes;           /* source descriptor packet count */
    U32             rtcpApp;            /* app-specific packet count */
    U32             rtcpBye;            /* bye packet count */

    /* error handling */
    U32             rtcpErrors;         /* number of errors encountered */
    U32             rtcpLastError;      /* last error encountered */
```

```
    U8              reserved[16];       /* reserved for future use */
} VTMNG_CHN_RTCP_RXTX;
```

Fields

Field	Type	Description
ntpMostSig	U32	Most significant 32-bits of NTP timestamp.
ntpLeastSig	U32	Least significant 32-bits of NTP timestamp.
rtpTimestamp	U32	RTP timestamp associated with NTP timestamp.
senderPktCount	U32	Total number of packets sent as reported by the sender.
senderOctetCount	U32	Total number of payload bytes sent as reported by the sender. Does not include headers or padding.
fractionLost	U8	Fraction of RTP data packets lost since previous sender report or receiver report was sent.
avail1	U8	Available for future use. Should be zero-filled.
cumulativePktsLost	U32	Total number of RTP data packets that have been lost.
extendedHighSeqNo	U32	Low 16-bits = highest sequence number received. High 16-bits = count of cycles.
interarrivalJitter	U32	Estimate of statistical variance of RTP data packet inter-arrival time.
lastSr	U32	(LSR) Middle 32-bits of NTP timestamp from most recently received sender report.
delaySinceLastSr	U32	Delay between receiving last SR and sending given reception report block.
rtcpSr	U32	Sender report packet count.
rtcpRr	U32	Receiver report packet count.
rtcpSdes	U32	Source descriptor packet count.
rtcpApp	U32	Application-specific packet count.
rtcpBye	U32	Bye packet count.
rtcpErrors	U32	Number of errors encountered.
rtcpLastError	U32	Last error encountered.
reserved	U8	Reserved for future use.

VTMNG_CHN_RTCPE

RTCP information maintained for both the input half and output half of each RTCP Translator endpoint.

Definition

```
typedef struct __vtMng_Chn_Rctp
{
    VTMNG_CHN_RTCP_RXTX rx;              /* common receive stats */
    VTMNG_CHN_RTCP_RXTX tx;              /* common transmit stats */
    U8                  reserved[16];    /* reserved for future use */
} VTMNG_CHN_RTCPE;
```

Fields

Field	Type	Description
rx	Structure	Common receive statistics. Specified in *VTMNG_CHN_RTCP_RXTX* on page 273.
tx	Structure	Common transmit statistics. Specified in *VTMNG_CHN_RTCP_RXTX* on page 273.
reserved	U8	Reserved for future use.

VTMNG_CHN_RTP

Overall RTP information.

Definition

```
typedef struct __vtMng_Chn_Rtp
{
    VTMNG_CHN_RTP_RXTX  rx;              /* RTP receive stats */
    VTMNG_CHN_RTP_RXTX  tx;              /* RTP transmit stats */
    U8                  reserved[16];    /* reserved for future use */
} VTMNG_CHN_RTP;
```

Fields

Field	Type	Description
rx	Structure	RTP receive statistics. Specified in *VTMNG_CHN_RTP_RXTX* on page 275.
tx	Structure	RTP transmit statistics. Specified in *VTMNG_CHN_RTP_RXTX* on page 275.
reserved	U8	Reserved for future use.

VTMNG_CHN_RTP_RXTX

RTP-related statistics that are maintained for both receive and transmit directions.

Definition

```
typedef struct __vtMng_Chn_Rtp_RxTx
{
    VTMNG_ST_COMM       comm;                /* RTP data communication statistics */
```

```
    U8                  reserved[16];       /* reserved for future use */
} VTMNG_CHN_RTP_RXTX;
```

Fields

Field	Type	Description
comm	Structure	RTP data communication statistics. Specified in *VTMNG_ST_COMM* on page 293.
reserved	U8	Reserved for future use.

VTMNG_CHN_START_NOTIF

Full channel configuration and status as channel is started.

Definition

```
typedef struct __vtMngChnStart_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    VTMNG_CHN_CFG       cfg;            /* channel configuration */
    VTMNG_CHN_STATUS    status;         /* current status information
                                           [RTP addressing not yet known] */
    U8                  reserved[16];   /* reserved for future use */
} VTMNG_CHN_START_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
cfg	Structure	Channel configuration. Specified in *VTMNG_CHN_CFG* on page 269.
status	Structure	Current status information. The RTP addressing information will not be filled in because this information is not known at channel start time. Specified in *VTMNG_CHN_STATUS* on page 277.
reserved	U8	Reserved for future use.

VTMNG_CHN_STATS

Complete statistical information for a given channel.

Definition

```
typedef struct __vtMng_Chn_Stats
{
    VTMNG_CHN_END   endpointA;          /* statistics maintained for endpoint A */
    VTMNG_CHN_END   endpointB;          /* statistics maintained for endpoint B */
    U8              reserved[16];       /* reserved for future use */
} VTMNG_CHN_STATS;
```

Fields

Field	Type	Description
endpointA	Structure	Statistics maintained for endpoint A. Specified in *VTMNG_CHN_END* on page 271.
endpointB	Structure	Statistics maintained for endpoint B. Specified in *VTMNG_CHN_END* on page 271.
reserved	U8	Reserved for future use.

VTMNG_CHN_STATUS

Current channel status information.

Definition

```
typedef struct __vtMng_Chn_Status
{
    U8              chnState;         /* current state of channel (VTMNG_CHN_S_xxx) */
    U8              avail[3];
    VSLOG_TIME      chnStartTime;     /* time when channel was last started */
    VSLOG_TIME      chnStopTime;      /* time when channel was last stopped */
    VTMNG_ADDR_INFO remTxEndA;        /* information that is known about the
                                         remote transmitter (endpoint A) */
    VTMNG_ADDR_INFO remRxEndB;        /* information that is known about the
                                         remote receiver (endpoint B) */
    VTMNG_ADDR_INFO remTxEndB;        /* [full-duplex only] info known about
                                         remote transmitter (endpoint B) */
    VTMNG_ADDR_INFO remRxEndA;        /* [full duplex only] info known about
                                         remote receiver (endpoint A) */
    U32             locRxEndA;        /* UDP port number transcoder receives
                                         RTP video stream (from endpoint A) */
    U32             locTxEndB;        /* UDP port number transcoder transmits
                                         RTP video stream from (to endpoint B) */
    U32             locRxEndB;        /* [full-duplex only] UDP port number
                                         transcoder receives RTP video stream
                                         (from endpoint B) */
    U32             locTxEndA;        /* [full duplex only] UDP port number
                                         transcoder transmits RTP video stream from
                                         (to endpoint A) */
    S8              locIpAddr[VTMNG_NAME_SZ]; /* IP address of the local transcoder
                                         side of the RTP/RTCP sessions */
    U8              reserved[16];     /* reserved for future use */
} VTMNG_CHN_STATUS;
```

Fields

Field	Type	Description
chnState	U8	Current state of channel (VTMNG_CHN_S_*xxx*).
avail	U8	Available for future use.
chnStartTime	Structure	Time when channel was last started. Uses the VSLOG_TIME record to represent time stamps provided through the VTMNG.

Field	Type	Description
chnStopTime	Structure	Time when channel was last stopped. Uses the VSLOG_TIME record to represent time stamps provided through the VTMNG.
remTxEndA	Structure	Information that is known about the remote transmitter (endpoint A). Specified in *VTMNG_ADDR_INFO* on page 265.
remRxEndB	Structure	Information that is known about the remote receiver (endpoint B). Specified in *VTMNG_ADDR_INFO* on page 265.
remTxEndB	Structure	Information that is known about the remote transmitter (endpoint B). Specified in *VTMNG_ADDR_INFO* on page 265. Full-duplex only.
remRxEndA	Structure	Information that is about the remote receiver (endpoint A) specified in *VTMNG_ADDR_INFO* on page 265. Full-duplex only.
locRxEndA	U32	UDP port number that the transcoder receives RTP video stream from endpoint A.
locTxEndB	U32	UDP port number that the transcoder transmits RTP video stream to endpoint B.
locRxEndB	U32	UDP port number that the transcoder receives RTP video stream from endpoint B. Full-duplex only.
locTxEndA	U32	UDP port number that the transcoder transmits RTP video stream to endpoint A from. Full-duplex only.
locIpAddr	S8	IP address of the local transcoder side of the RTP/RTCP sessions.
reserved	U8	Reserved for future use.

VTMNG_CHN_STOP_NOTIF

Full channel configuration, status, and final overall channel statistics as channel is stopped.

Definition

```
typedef struct __vtMngChnStop_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    VTMNG_CHN_CFG       cfg;            /* channel configuration */
    VTMNG_CHN_STATUS    status;         /* status information at time when channel
                                           stopped */
    VTMNG_CHN_STATS     stats;          /* set of statistics maintained by channel */
    U8                  reserved[16];   /* reserved for future use */
} VTMNG_CHN_STOP_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
cfg	Structure	Channel configuration. Specified in *VTMNG_CHN_CFG* on page 269.
status	Structure	Status information at time when channel is stopped. Specified in *VTMNG_CHN_STATUS* on page 277.
stats	Structure	Set of statistics maintained by channel. Specified in *VTMNG_CHN_STATS* on page 276.
reserved	U8	Reserved for future use.

VTMNG_CHN_VTYPE

Video type-specific information maintained for both receive and transmit directions.

Definition

```
typedef struct __vtMng_Chn_VType
{
    VTMNG_CHN_VTYPE_RXTX    rx;         /* common receive stats */
    VTMNG_CHN_VTYPE_RXTX    tx;         /* common transmit stats */
    U8                      reserved[16];   /* reserved for future use */
} VTMNG_CHN_VTYPE;
```

Fields

Field	Type	Description
rx	Structure	Common receive statistics. Specified in *VTMNG_CHN_VTYPE_RXTX* on page 280.
tx	Structure	Common transmit statistics. Specified in *VTMNG_CHN_VTYPE_RXTX* on page 280.
reserved	U8	Reserved for future use.

VTMNG_CHN_VTYPE_RXTX

Video type-specific statistics maintained for both receive and transmit directions.

Definition

```
typedef struct __vtMng_Chn_VType_RxTx
{
    VTMNG_ST_VTYPE  vtcomm;              /* video type-specific communication stats */
    U8              reserved[16];        /* reserved for future use */
} VTMNG_CHN_VTYPE_RXTX;
```

Fields

Field	Type	Description
vtcomm	Structure	Video type-specific communication statistics. Specified in *VTMNG_ST_VTYPE* on page 296.
reserved	U8	Reserved for future use.

VTMNG_ENT_ID

An entity ID that fully identifies a given entity with a unique ID and ASCII name when available.

Definition

```
typedef struct __vtMngEnt_Id
{
    VTMNG_OBJ_ID    entObj;                   /* basic entity object identifier */
    S8              entName[VTMNG_NAME_SZ];  /* optional ASCII name of entity */

    U8              reserved[4];              /* reserved for future use */
} VTMNG_ENT_ID;
```

Fields

Field	Type	Description
entObj	Structure	Basic entity object identifier. Specified in *VTMNG_OBJ_ID* on page 289.
entName	S8	Optional ASCII name of entity.
reserved	U8	Reserved for future use.

VTMNG_GETENT_RSP

Set of all responses that provide complete information for a specific entity.

Definition

```
typedef struct __vtMngGetEnt_Rsp
{
    VTMNG_RSP_MSG       common;          /* common portion */

    union
    {
        VTMNG_VTP_ENTITY    vtp;         /* VTP top-level entity */
        VTMNG_APP_ENTITY    app;         /* application entity */
        VTMNG_MON_ENTITY    mon;         /* monitored process entity */
```

```
        VTMNG_CHN_ENTITY    chn;        /* channel entity */
        VTMNG_ST_ENTRY      stats;      /* statistics */
    } u;

} VTMNG_GETENT_RSP;
```

Field listing

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_RSP_MSG* on page 290.
u	Union	Area used to hold the complete entity response information. Select specific substructure based on the message category. Valid values: vtp: Video transcoder platform top-level entity. Specified in *VTMNG_VTP_ENTITY* on page 303. app: Application entity. Specified in *VTMNG_APP_ENTITY* on page 266. mon: Monitored process entity. Specified in *VTMNG_MON_ENTITY* on page 283. chn: Channel entity. Specified in *VTMNG_CHN_ENTITY* on page 272. stats: Statistics. Specified in *VTMNG_ST_ENTRY* on page 295.

VTMNG_GETLIST_RSP

Common response to any request for a list of entities.

Definition

```
typedef struct __vtMngGetList_Rsp
{
    VTMNG_RSP_MSG     common;       /* common portion */

    VTMNG_ENT_ID      gListEnt[1];  /* first entity in response list
                                       (actual count = resultCount) */
} VTMNG_GETLIST_RSP;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_RSP_MSG* on page 290.
gListEnt	Structure	First entity in response list (actual count = resultCount). Specified in *VTMNG_ENT_ID* on page 280.

VTMNG_HDR

Header common to all messages.

Definition

```
typedef struct __vtMngHdr
{
    U8         version;        /* VT management protocol version in use
```

```
                                 (VTMNG_VERSION) */
    U8        revision;       /* VT management protocol revision in use
                                 (VTMNG_REVISION) */
    U16       len;            /* byte length of entire message
                                 [sizeof(VTMNG_HDR) + payload] */
    U8        msgOp;          /* message operation (VTMNG_OP_xxx) */
    U8        msgTypeId;      /* message type ID (VTMNG_xxx_ID) */
    U8        msgCategory;    /* message category (VTMNG_CATEG_xxx) */
    U8        avail[1];       /* available for future use */
    U32       senderID;       /* ID set by sender (Req) and copied by responder (Rsp);
                                 set by sender (Ind) */
    U32       sequenceNum;    /* outbound sequence number [set by sender] (incremented
                                 each time a message is sent) */
    U8        reserved[16];   /* reserved for future use */
} VTMNG_HDR;
```

Fields

Field	Type	Description
version	U8	Video transcoder management protocol version in use (VTMNG_VERSION).
revision	U8	Video transcoder management protocol revision in use (VTMNG_REVISION).
len	U16	Byte length of entire message [size of (VTMNG_HDR) + payload].
msgOp	U8	Message operation (VTMNG_OP_*xxx*).
msgTypeId	U8	Message type ID (VTMNG_*xxx*_ID).
msgCategory	U8	Message category (VTMNG_CATEG_*xxx*).
avail	U8	Available for future use.
senderID	U32	ID set by the sender (Req) and copied by the responder (Rsp). Set by sender (Ind).
sequenceNum	U32	Outbound sequence number set by sender that is incremented each time a message is sent.
reserved	U8	Reserved for future use.

VTMNG_MON_CFG

Configuration information for a given monitored process.

Definition

```
typedef struct __vtMng_Mon_Cfg
{
    /* configuration elements that are set internally */
    U32         monUnique;                      /* unique value ID'ing this monitored
                                                   process */
    S8          monName[VTMNG_NAME_SZ];         /* name of monitored process */
    U8          monEvent;                       /* optional event to issue to the
                                                   process monitor (VTMNG_MON_E_xxx) */
    U8          monInitState;                   /* initial monitoring state for given
                                                   process (VTMNG_MON_I_xxx) */
```

```
    U8          monLostAction;                  /* action to be taken if process is
                                                   ever lost (VTMNG_MON_A_xxx) */
    S8          monExecutable[VTMNG_NAME_SZ];   /* name of executable file for this
                                                    process */
    S8          monCmdLine[VTMNG_LONGNAME_SZ];  /* command-line options provided on
                                                   process creation */
    VTMNG_MON_VAR   monVar[VTMNG_MON_VARS];     /* optional set variables that can be
                                                   stored by process monitor */
    U8          reserved[16];                   /* reserved for future use */
} VTMNG_MON_CFG;
```

Fields

Field	Type	Description
monUnique	U32	Unique value identifying this monitored process.
monName	S8	Name of monitored process.
monEvent	U8	Optional event to issue to the process monitor (VTMNG_MON_E_*xxx*).
monInitState	U8	Initial monitoring state for given process (VTMNG_MON_I_*xxx*).
monLostAction	U8	Action to be taken if process is ever lost (VTMNG_MON_A_*xxx*).
monExecutable	S8	Name of executable file for this process.
monCmdLine	S8	Command-line options provided on process creation.
monVar	Structure	Optional set of variables that can be stored by the process monitor. Specified in *VTMNG_MON_VAR* on page 285.
reserved	U8	Reserved for future use.

VTMNG_MON_ENTITY

Contains all information for a given monitored process.

Definition

```
typedef struct __vtMng_Mon_Entity
{
    VTMNG_MON_CFG       cfg;            /* monitored process configuration */
    VTMNG_MON_STATUS    status;         /* current status of monitored process */
    VTMNG_MON_STATS     stats;          /* set of statistics maintained per
                                           monitored process */
    U8                  reserved[16];   /* reserved for future use */
} VTMNG_MON_ENTITY;
```

Fields

Field	Type	Description
cfg	Structure	Monitored process configuration. Specified in *VTMNG_MON_CFG* on page 282.
status	Structure	Current status of monitored process. Specified in *VTMNG_MON_STATUS* on page 285.
stats	Structure	Set of statistics maintained per monitored process. Specified in *VTMNG_MON_STATS* on page 284.
reserved	U8	Reserved for future use.

VTMNG_MON_STATS

Statistics maintained for a given monitored process.

Definition

```
typedef struct __vtMng_Mon_Stats
{
    U32             monLost;            /* number of times that monitored process
                                           was lost */
    U32             monLostReason;      /* last reason that monitored process was
                                           considered lost */
    U8              reserved[16];       /* reserved for future use */
} VTMNG_MON_STATS;
```

Fields

Field	Type	Description
monLost	U32	Number of times that monitored process was lost.
monLostReason	U32	Last reason that monitored process was considered lost.
reserved	U8	Reserved for future use.

VTMNG_MON_STATUS

Current status of a given monitored process.

Definition

```
typedef struct __vtMng_Mon_Status
{
    U8          monState;               /* current state of the monitored process
                                           (VTMNG_MON_S_xxx) */
    U8          avail[3];
    U32         monParent;              /* non-zero value when process parent is
                                           also being monitored */
    VSLOG_TIME  monStartTime;           /* time when this process was last started
                                           (when under monitor control) */
    U8          reserved[16];           /* reserved for future use */
} VTMNG_MON_STATUS;
```

Fields

Field	Type	Description
monState	U8	Current state of the monitored process (VTMNG_MON_S_*xxx*).
avail	U8	Available for future use.
monParent	U32	Non-zero value when process parent is also being monitored.
monStartTime	Structure	Time when this process was last started (when operating in the default monitor mode [monitor on]). Uses the VSLOG_TIME record to represent time stamps provided through the VTMNG.
reserved	U8	Reserved for future use.

VTMNG_MON_VAR

Holds a monitored process variable definition.

Definition

```
typedef struct __vtMng_Mon_Var
{
    S8          varName[VTMNG_NAME_SZ];        /* name of optional variable */
    S8          varString[VTMNG_NAME_SZ];      /* string value of variable */
    U8          reserved[8];                   /* reserved for future use */
} VTMNG_MON_VAR;
```

Fields

Field	Type	Description
varName	S8	Name of optional variable.
varString	S8	String value of variable.
reserved	U8	Reserved for future use.

VTMNG_MONPROC_NOTIF

Information provided by notifications indicating a change in state of a monitored process.

Definition

```
typedef struct __vtMngMonProc_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    VTMNG_MON_ENTITY    mon;            /* monitored process information */
    U8                  reserved[16];   /* reserved for future use */

} VTMNG_MONPROC_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
mon	Structure	Monitored process information. Specified in *VTMNG_MON_ENTITY* on page 283.
reserved	U8	Reserved for future use.

VTMNG_NOTIF_INFO

Common notification information.

Definition

```
typedef struct __vtMngNotif_Info
{
    U32      eventId;                    /* unique value identifying event being
                                            notified (VTMNG_EVENT_xxx) */
    U32      errCode;                    /* 0 = successful event; else error code
                                                associated with event */
    U32      severity;                   /* severity level of notification
                                            (VSLOG_SEV_xxx) */
    U32      options;                    /* options associated with the event|alarm
                                            (VSLOG_EVT_xxx) */
    U32      alarmState;                 /* state associated with alarm condition
                                            (VSLOG_ALARM_xxx) */
    S8       locName[VTMNG_NAME_SZ];     /* OPTIONAL name of location where notif
                                            generated */
    S8       notifDesc[VTMNG_DESC_SZ];   /* OPTIONAL text description of notification */
    U8       reserved[20];               /* reserved for future use */
} VTMNG_NOTIF_INFO;
```

Fields

Field	Type	Description
eventId	U32	Unique value identifying the event being notified (VTMNG_EVENT_*xxx*). The event type determines which trap notification function is upcalled.

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td>errCode</td><td>U32</td><td>If 0, the event was successful. Otherwise, an error code is associated with the event.

The error code is intended to help developers quickly identify the source of errors within the code space. Use the notifDesc text to obtain additional information related to the error.</td></tr>
<tr><td>severity</td><td>U32</td><td>Severity level of notification (VSLOG_SEV_xxx):
<table>
<tr><th>Severity level</th><th>Description</th></tr>
<tr><td>VSLOG_SEV_EMERGENCY</td><td>System is unusable.</td></tr>
<tr><td>VSLOG_SEV_ALERT</td><td>Action must be taken immediately.</td></tr>
<tr><td>VSLOG_SEV_CRITICAL</td><td>Critical conditions.</td></tr>
<tr><td>VSLOG_SEV_MAJOR</td><td>Major affect on system operation.</td></tr>
<tr><td>VSLOG_SEV_ERROR</td><td>Error conditions.</td></tr>
<tr><td>VSLOG_SEV_MINOR</td><td>Minor affect on system operation.</td></tr>
<tr><td>VSLOG_SEV_WARNING</td><td>Warning conditions.</td></tr>
<tr><td>VSLOG_SEV_NOTICE</td><td>Normal but significant condition.</td></tr>
<tr><td>VSLOG_SEV_INFO</td><td>Informational message. No effect on system operation.</td></tr>
<tr><td>VSLOG_SEV_DEBUG</td><td>Debug-level messages.</td></tr>
<tr><td>SLOG_SEV_NONE</td><td>Special indicator when no severity provided.</td></tr>
</table></td></tr>
<tr><td>options</td><td>U32</td><td>Options associated with the event or alarm (VSLOG_EVT_xxx):
<table>
<tr><th>Event or alarm</th><th>Description</th></tr>
<tr><td>VSLOG_EVT_SVC_AFFECT</td><td>Event or alarm has an affect on the overall service.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Field</th><th>Type</th><th>Description</th></tr>
<tr><td>alarmState</td><td>U32</td><td>State associated with alarm condition (VSLOG_ALARM_xxx)
<table>
<tr><th>Alarm state</th><th>Value</th><th>Description</th></tr>
<tr><td>VSLOG_ALARM_NONE</td><td>0</td><td>No alarm associated with the event.</td></tr>
<tr><td>VSLOG_ALARM_INFO</td><td>1</td><td>Alarm providing generic information.</td></tr>
<tr><td>VSLOG_ALARM_IN_STATE</td><td>2</td><td>Entity has entered the state in which alarm condition exists (in alarm).</td></tr>
<tr><td>VSLOG_ALARM_OUT_STATE</td><td>3</td><td>Entity has left the state in which alarm condition existed (out of alarm).</td></tr>
<tr><td>VSLOG_ALARM_THRESH</td><td>4</td><td>A threshold has been crossed.</td></tr>
</table></td></tr>
<tr><td>locName</td><td>S8</td><td>Optional. Name of location where notification was generated.</td></tr>
<tr><td>notifDesc</td><td>S8</td><td>Optional. Text description of notification.</td></tr>
<tr><td>reserved</td><td>U8</td><td>Reserved for future use.</td></tr>
</table>

VTMNG_NOTIF_MSG

Information common to all asynchronous notifications (traps).

Definition

```
typedef struct __vtMngNotif_Msg
{
    VTMNG_HDR           mngHdr;     /* common message header */
    VTMNG_OBJ_ID        mngObj;     /* source object */

    VTMNG_NOTIF_INFO    info;       /* common notification information */

} VTMNG_NOTIF_MSG;.
```

Fields

Field	Type	Description
mngHdr	Structure	Common message header. Specified in *VTMNG_HDR* on page 281.
mngObj	Structure	Source object. Specified in *VTMNG_OBJ_ID* on page 289.

Field	Type	Description
info	Structure	Common notification information. Specified in *VTMNG_NOTIF_INFO* on page 286.

VTMNG_OBJ_ID

Minimal information to uniquely identify any managed object (entity).

Definition

```
typedef struct __vtMngObj_Id
{
    U32         entUnique;              /* unique value ID'ing this entity;
                                           unique across set of entities in a given list:
                                         *   VTP: NOT USED
                                         *   APP: entUnique = cfg.appUnique
                                         *   CHN: entUnique = cfg.chnUnique
                                         * STATS: entUnique = 1..VTMNG_MAX_MIN_BUCKET |
                                                              1..VTMNG_MAX_HHR_BUCKET
                                         * TRAPS: entUnique = RESERVED FOR FUTURE USE */
    U8          reserved[4];            /* reserved for future use */
} VTMNG_OBJ_ID;.
```

Fields

Field	Type	Description
entUnique	U32	Unique value used to identify this entity. This value is unique across a set of entities in a given list. Valid values: VTP: Not used. APP: entUnique = cfg.appUnique CHN: entUnique = cfg.chnUnique STATS: entUnique = 1..VTMNG_MAX_MIN_BUCKET | 1..VTMNG_MAX_HHR_BUCKET TRAPS: entUnique = Reserved for future use.
reserved	U8	Reserved for future use.

VTMNG_REQ_MSG

Information common to all requests.

Definition

```
typedef struct __vtMngReq_Msg
{
    VTMNG_HDR           mngHdr;     /* common message header */
    VTMNG_OBJ_ID        mngObj;     /* destination object */
} VTMNG_REQ_MSG;
```

Fields

Field	Type	Description
mngHdr	Structure	Common message header. Specified in *VTMNG_HDR* on page 281.

Field	Type	Description
mngObj	Structure	Destination object. Specified in *VTMNG_OBJ_ID* on page 289.

VTMNG_RSP_INFO

Common response information.

Definition

```
typedef struct __vtMngRsp_Info
{
    U32         result;              /* result of request */
    U32         resultCount;         /* number of elements provided in response
                                        (when result = 0) */
    U8          reserved[16];        /* reserved for future use */
} VTMNG_RSP_INFO;
```

Fields

Field	Type	Description
result	U32	Result of request.
resultCount	U32	Number of elements provided in response when the result is equal to 0.
reserved	U8	Reserved for future use.

VTMNG_RSP_MSG

Information common to all responses.

Definition

```
typedef struct __vtMngRsp_Msg
{
    VTMNG_HDR           mngHdr;      /* common message header */
    VTMNG_OBJ_ID        mngObj;      /* destination object from corresponding
                                        request */
    VTMNG_RSP_INFO      mngRsp;      /* common response information */
} VTMNG_RSP_MSG;
```

Fields

Field	Type	Description
mngHdr	Structure	Common message header. Specified in *VTMNG_HDR* on page 281.
mngObj	Structure	Destination object. Specified in *VTMNG_OBJ_ID* on page 289.
mngRsp	Structure	Common response information. Specified in *VTMNG_RSP_INFO* on page 290.

VTMNG_RTCP_INFO

RTCP-specific addressing information. As an RTCP translator, each transcoder channel records any of the optional RTCP source descriptors reported by the remote endpoints.

Definition

```
typedef struct __vtMng_Rtcp_Info
{
    S8              cname[VTMNG_NAME_SZ];    /* RTCP: CNAME */
    S8              name[VTMNG_NAME_SZ];     /* RTCP: NAME */
    S8              email[VTMNG_NAME_SZ];    /* RTCP: EMAIL */
    S8              phone[VTMNG_NAME_SZ];    /* RTCP: PHONE */
    S8              loc[VTMNG_NAME_SZ];      /* RTCP: LOC */
    S8              tool[VTMNG_NAME_SZ];     /* RTCP: TOOL */
    S8              note[VTMNG_NAME_SZ];     /* RTCP: NOTE */
    U8              reserved[16];            /* reserved for future use */
} VTMNG_RTCP_INFO;
```

Fields

Field	Type	Description
cname	S8	RTCP: CNAME. Canonical name that uniquely identifies the endpoint participating in the RTCP session.
name	S8	RTCP: NAME. User name used to describe the remote endpoint.
email	S8	RTCP: EMAIL. Electronic mail address. For example: John.Doe@example.com
phone	S8	RTCP: PHONE. Phone number. For example: +1 908 555 1212
loc	S8	RTCP: LOC. Geographic user location.
tool	S8	RTCP: TOOL. Application or tool name.
note	S8	RTCP: NOTE. Notice or status.
reserved	U8	Reserved for future use.

VTMNG_SETCFG_REQ

Set of all requests that perform configuration changes.

Definition

```
typedef struct __vtMngSetCfg_Req
{
    VTMNG_REQ_MSG       common;         /* common portion */

    union
    {
        VTMNG_VTP_CFG   vtp;            /* VTP top-level configuration */
        VTMNG_APP_CFG   app;            /* application entity configuration */
        VTMNG_MON_CFG   mon;            /* monitored process entity configuration */
        VTMNG_CHN_CFG   chn;            /* channel entity configuration */
    } value;

    union
    {
```

```
        VTMNG_VTP_CFG   vtp;            /* VTP top-level configuration */
        VTMNG_APP_CFG   app;            /* application entity configuration */
        VTMNG_MON_CFG   mon;            /* monitored process entity configuration */
        VTMNG_CHN_CFG   chn;            /* channel entity configuration */
    } mask;

} VTMNG_SETCFG_REQ;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_REQ_MSG* on page 289.
value	Union	Record holding all new configuration values. Valid substructures (depending on the set request type): vtp: Video transcoder platform top-level configuration. Specified in *VTMNG_VTP_CFG* on page 299. app: Application entity configuration. Specified in *VTMNG_APP_CFG* on page 265. mon: Monitored process entity configuration. Specified in *VTMNG_MON_CFG* on page 282. chn: Channel entity configuration. Specified in *VTMNG_CHN_CFG* on page 269.
mask	Union	Record holding a mask that identifies which specific values are to be modified. Valid substructures (depending on the set request type): vtp: Video transcoder platform top-level configuration. Specified in *VTMNG_VTP_CFG* on page 299. app: Application entity configuration. Specified in *VTMNG_APP_CFG* on page 265. mon: Monitored process entity configuration. Specified in *VTMNG_MON_CFG* on page 282. chn: Channel entity configuration. Specified in *VTMNG_CHN_CFG* on page 269.

VTMNG_SETCFG_RSP

Set of all responses to configuration modification requests.

Definition

```
typedef struct __vtMngSetCfg_Rsp
{
    VTMNG_RSP_MSG       common;         /* common portion */

    union
    {
        VTMNG_VTP_CFG   vtp;            /* VTP top-level configuration */
        VTMNG_APP_CFG   app;            /* application entity configuration */
        VTMNG_MON_CFG   mon;            /* monitored process entity configuration */
        VTMNG_CHN_CFG   chn;            /* channel entity configuration */
    } u;

} VTMNG_SETCFG_RSP;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_RSP_MSG* on page 290.
u	Union	Valid substructures: vtp: Video transcoder platform top-level configuration. Specified in *VTMNG_VTP_CFG* on page 299. app: Application entity configuration. Specified in *VTMNG_APP_CFG* on page 265. mon: Monitored process entity configuration. Specified in *VTMNG_MON_CFG* on page 282. chn: Channel entity configuration. Specified in *VTMNG_CHN_CFG* on page 269.

VTMNG_ST_COMM

Statistics substructure containing data communication related statistics.

Definition

```
typedef struct __vtMng_St_Comm
{
    U32          packets;            /* packet count */
    U32          bytes;              /* byte count */
    U32          errTooSmall;        /* errors due to packet size less than
                                        minimum */
    U32          errAddrChange;      /* errors due to detected change of remote
                                        address */
    U32          errOutOfRange;      /* errors due to packets with sequence numbers
                                        or timestamps out of current valid range */
    U32          errPartialFrame;    /* errors due to partial frames (frames with
                                        incomplete information) */
    U32          errDupSeqNo;        /* errors due to duplicate sequence number
                                        detected */
    U32          errMultipleLast;    /* errors due to multiple packets (of same
                                        frame) indicating last in frame */
    U32          errInvalidMode;     /* errors due to invalid mode */
    U32          errUnknownType;     /* errors due to unknown packet type detected */
    U32          errOutOfOrder;      /* errors due to packets in non-sequencial
                                        order */
```

```
    U32           errDataLoss;        /* errors due to loss of content in data stream */
    U32           errTooBig;          /* errors due to frame size growing too big */
    U32           errNoTranscode;     /* errors indicating when decoding a frame
                                         did not result in an encoded frame */
    U32           errors;             /* number of errors detected which are outside
                                         errXxxx statistics set */
    U32           lastError;          /* last error code counted in errors */
    U8            reserved[16];       /* reserved for future use */
} VTMNG_ST_COMM;
```

Fields

Field	Type	Description
packets	U32	Packet count.
bytes	U32	Byte count.
errTooSmall	U32	Errors due to packet size less than minimum.
errAddrChange	U32	Errors due to detected change of remote address.
errOutOfRange	U32	Errors due to packets with sequence numbers or timestamps that fall outside the current window.
errPartialFrame	U32	Errors due to partial frames (frames with incomplete information).
errDupSeqNo	U32	Errors due to detected duplicate sequence number.
errMultipleLast	U32	Errors due to multiple packets (of same frame) indicating last in frame.
errInvalidMode	U32	Errors due to invalid mode.
errUnknownType	U32	Errors due to unknown detected packet type.
errOutOfOrder	U32	Errors due to packets in non-sequential order.
errDataLoss	U32	Errors due to loss of content in data stream.
errTooBig	U32	Errors due to frame size growing too big.
errNoTranscode	U32	Errors indicating when decoding a frame did not result in an encoded frame.
errors	U32	Number of errors detected which are outside err*Xxxx* statistics set.
lastError	U32	Last error code counted in errors.
reserved	U8	Reserved for future use.

VTMNG_ST_DIR

Statistics substructure that holds a set of statistics being maintained per direction of data flow.

Definition

```
typedef struct __vtMng_St_Dir
{
    VTMNG_ST_COMM       rtp;             /* RTP statistics */
    VTMNG_ST_VTYPE      vtype;           /* Video-type related statistics */
    VTMNG_ST_COMM       rtcp;            /* RTCP statistics */
    U8                  reserved[16];    /* reserved for future use */
} VTMNG_ST_DIR;
```

Fields

Field	Type	Description
rtp	Structure	RTP statistics. Specified in *VTMNG_ST_COMM* on page 293.
vtype	Structure	Video-type related statistics. Specified in *VTMNG_ST_VTYPE* on page 296.
rtcp	Structure	RTCP statistics. Specified in *VTMNG_ST_COMM* on page 293.
reserved	U8	Reserved for future use.

VTMNG_ST_ENTRY

Common statistics record providing video transcoder platform-level statistics as well as summary statistics for receive and transmit related information.

```
typedef struct __vtMng_St_Entry
{
    VTMNG_VTP_STATS     vtp;             /* VTP-level statistics */
    VTMNG_ST_DIR        rx;              /* receive statistics */
    VTMNG_ST_DIR        tx;              /* transmit statistics */
    U8                  reserved[16];    /* reserved for future use */
} VTMNG_ST_ENTRY;
```

Fields

Field	Type	Description
vtp	Structure	Video transcoder platform-level statistics. Specified in *VTMNG_VTP_STATS* on page 303.
rx	Structure	Receive statistics. Specified in *VTMNG_ST_DIR* on page 295.
tx	Structure	Transmit statistics. Specified in *VTMNG_ST_DIR* on page 295.
reserved	U8	Reserved for future use.

VTMNG_ST_VTYPE

Statistics substructure containing video-type related statistics.

Definition

```
typedef struct __vtMng_St_VType
{
    U32             iframes;            /* count of I-FRAMEs */
    U32             iframeBytes;        /* total number of I-FRAME bytes */
    U32             pframes;            /* count of P-FRAMEs */
    U32             pframeBytes;        /* total number of P-FRAME bytes */
    U32             leadPframe;         /* number of P-FRAMEs before initial I-FRAME */
    U32             waitTooShort;       /* number of times data processing postponed
                                           due to insufficient frame bits */
    U32             waitNoStart;        /* number of times data processing postponed
                                           due to lack of start code */
    U32             skipNoAlign;        /* number of times data bits skipped due to
                                           lack of alignment code */
    U32             skipBadAlign;       /* number of times data bits skipped due to
                                           bad alignment */
    U32             dropEarly;          /* number of times data dropped (arrived early
                                           [before other required info available]) */
    U32             dropGarbage;        /* number of times data dropped due to garbage
                                           detected */
    U32             dropTooBig;         /* number of frames dropped due to data
                                           overflow */
    U32             errors;             /* number of errors encountered */
    U32             lastError;          /* last error encountered */
    U8              reserved[16];       /* reserved for future use */
} VTMNG_ST_VTYPE;
```

Fields

Field	Type	Description
iframes	U32	Count of I-FRAMEs.
iframeBytes	U32	Total number of I-FRAME bytes.
pframes	U32	Count of P-FRAMEs.
pframeBytes	U32	Total number of P-FRAME bytes.
leadPframe	U32	Number of P-FRAMEs before initial I-FRAME.
waitTooShort	U32	Number of times data processing was postponed due to insufficient frame bits.
waitNoStart	U32	Number of times data processing was postponed due to a lack of start code.
skipNoAlign	U32	Number of times data bits skipped due to lack of alignment code.
skipBadAlign	U32	Number of times data bits skipped due to bad alignment.
dropEarly	U32	Number of times data dropped due to early arrival (before other required information was available).

Field	Type	Description
dropGarbage	U32	Number of times data dropped due to detection of an invalid value.
dropTooBig	U32	Number of frames dropped due to data overflow.
errors	U32	Number of errors encountered that were outside of these error conditions.
lastError	U32	Last error code that was counted in the errors field. The most common transcoding errors are counted in error-specific counters (for example, waitTooShort and dropEarly). Errors that are outside of the most common cases are counted in the errors field with lastError providing a code to uniquely identify the most recent cause of the error field being incremented. Use vtMngValueName with a valueType of VTMNG_VALUE_LASTERR to retrieve an ASCII string equivalent to: ascii_error_name = vtMngValueName(VTMNG_VALUE_LASTERR, statsRecord->lastError);
reserved	U8	Reserved for future use.

VTMNG_UPCALLS

Defines the upcall function set used by the video transcoder management interface (VTMNG) to route received management messages to the proper handling function. Any handling function that is set to NULL causes the internal handler function for the VTMNG to be called instead.

By initializing all upcall entries to NULL, the VTMNG is configured for default handling of all responses and notifications. This is the default mode used by the *vtmgr* tool.

For complete control over all received management messages, set each upcall function. The *vtmgr* tool also can operate in this mode and provides an example for each type of upcall function in the *vtmgr.c* sample code.

Definition

```
typedef struct __vtmng_upcalls
{
    /* ----- responses providing lists of elements ----- */
    vtMngGetAppListRsp      vtGetAppListRsp;
    vtMngGetMonListRsp      vtGetMonListRsp;
    vtMngGetChnListRsp      vtGetChnListRsp;

    /* ----- responses providing details of a specific element ----- */
    vtMngGetVtpRsp          vtGetVtpRsp;
    vtMngGetAppRsp          vtGetAppRsp;
    vtMngGetMonRsp          vtGetMonRsp;
    vtMngGetChnRsp          vtGetChnRsp;
    vtMngGetStTotalRsp      vtGetStTotalRsp;
    vtMngGetStCurrMinRsp    vtGetStCurrMinRsp;
    vtMngGetHistPerMinRsp   vtGetHistPerMinRsp;
    vtMngGetHistPerHHrRsp   vtGetHistPerHHrRsp;
```

```
    /* ----- responses to configuration (or overall state) modifications ----- */
    vtMngSetVtpRsp          vtSetVtpRsp;
    vtMngEventVtpRsp        vtEventVtpRsp;
    vtMngSetAppRsp          vtSetAppRsp;
    vtMngEventAppRsp        vtEventAppRsp;
    vtMngSetMonRsp          vtSetMonRsp;
    vtMngEventMonRsp        vtEventMonRsp;
    vtMngSetChnRsp          vtSetChnRsp;
    vtMngEventChnRsp        vtEventChnRsp;

    /* ----- responses to zero-statistics requests ----- */
    vtMngZeroVtpRsp         vtZeroVtpRsp;
    vtMngZeroAppRsp         vtZeroAppRsp;
    vtMngZeroMonRsp         vtZeroMonRsp;
    vtMngZeroChnRsp         vtZeroChnRsp;
    vtMngZeroTotalRsp       vtZeroTotalRsp;

    /* ----- traps (unsolicited asynchronous notifications) ----- */
    vtMngVtpLevelTrap       vtVtpLevelTrap;
    vtMngVtpErrorTrap       vtVtpErrorTrap;
    vtMngAppConnTrap        vtAppConnTrap;
    vtMngMonProcTrap        vtMonProcTrap;
    vtMngChnCreateTrap      vtChnCreateTrap;
    vtMngChnStartTrap       vtChnStartTrap;
    vtMngChnStopTrap        vtChnStopTrap;
    vtMngChnDeadTrap        vtChnDeadTrap;
    vtMngChnErrorTrap       vtChnErrorTrap;

    /* ----- optional keyboard input handling ----- */
    vtMngCmdNotif           vtCmdNotif;

} VTMNG_UPCALLS;
```

Fields

This field...	Is called when...
vtGetAppListRsp	Response to vtMngGetAppList is received.
vtGetMonListRsp	Response to vtMngGetMonList is received.
vtGetChnListRsp	Response to vtMngGetChnList is received.
vtGetVtpRsp	Response to vtMngGetVtp is received.
vtGetAppRsp	Response to vtMngGetApp is received.
vtGetMonRsp	Response to vtMngGetMon is received.
vtGetChnRsp	Response to vtMngGetChn is received.
vtGetStTotalRsp	Response to vtMngGetStTotal is received.
vtGetStCurrMinRsp	Response to vtMngGetStCurrMin is received.
vtGetHistPerMinRsp	Response to vtMngGetHistPerMin is received.
vtGetHistPerHHrRsp	Response to vtMngGetHistPerHHr is received.
vtSetVtpRsp	Response to vtMngSetVtp is received.
vtEventVtpRsp	Response to vtMngEventVtp is received.

This field...	Is called when...
vtSetAppRsp	Response to vtMngSetApp is received.
vtEventAppRsp	Response to vtMngEventApp is received.
vtSetMonRsp	Response to vtMngSetMon is received.
vtEventMonRsp	Response to vtMngEventMon is received.
vtSetChnRsp	Response to vtMngSetChn is received.
vtEventChnRsp	Response to vtMngEventChn is received.
vtZeroVtpRsp	Response to **vtMngZeroVtp** is received.
vtZeroAppRsp	Response to **vtMngZeroApp** is received.
vtZeroMonRsp	Response to **vtMngZeroMon** is received.
vtZeroChnRsp	Response to **vtMngZeroChn** is received.
vtZeroTotalRsp	Response to **vtMngZeroTotal** is received.
vtVtpLevelTrap	A video transcoder platform threshold level asynchronous notification (trap) is received.
vtVtpErrorTrap	A video transcoder platform-level asynchronous error notification is received.
vtAppConnTrap	An application connection state change notification is received.
vtMonProcTrap	A monitored process lost/recovered notification is received.
vtChnCreateTrap	A channel created notification is received.
vtChnStartTrap	A channel started notification is received.
vtChnStopTrap	A channel stopped notification is received.
vtChnDeadTrap	A channel dead notification is received.
vtChnErrorTrap	A channel-level error notification is received.
vtCmdNotif	Keyboard input enabled and operator command is entered.

VTMNG_VTP_CFG

Contains all video transcoder platform-level configuration.

Definition

```
typedef struct __vtMng_Vtp_Cfg
{
    S8    vtpName[VTMNG_LONGNAME_SZ]; /* name of this VTP instance
```

```
                                    [@|EMPTY = use hostname] */
    S8    vtpDesc[VTMNG_DESC_SZ];     /* VTP description (usually version/
                                    revision, etc.)
                                    [@|EMPTY = trc_agent-generated] */
    U8    vtpEvent;                 /* optional event to issue to the
                                    top-level VTP controller
                                    (VTMNG_VTP_E_xxx) */
    U8    vtpInitState;             /* set whether VTP will init to a
                                    disabled state (VTMNG_VTP_I_xxx) */
    U8    rtcpMode;                 /* set whether channels act as RTCP
                                    translators by default
                                    [VTMNG_CFG_DISABLED|VTMNG_CFG_ENABLED] */
    U8    decodePartials;           /* set whether partial frames should be
                                    passed to the decoder
                                    [VTMNG_CFG_DISABLED|VTMNG_CFG_ENABLED] */
    U8    overlayExclusive;         /* whether VTP is reserved for channels
                                    requiring overlays
                                    [VTMNG_CFG_DISABLED|VTMNG_CFG_ENABLED] */
    U8    logToConsole;             /* whether "to-file" logging should be
                                    forked to console
                                    [VTMNG_CFG_DISABLED|VTMNG_CFG_ENABLED] */
    U8    avail2[3];
    U32   licenseHighWater;         /* percentage of licenses in use at which time
                                    VTMNG_EVENT_VTP_LICENSE notif issued */
    U32   licenseLowWater;          /* percentage of licenses in use at which
                                    time VTMNG_EVENT_VTP_LICENSE notif issued */
    U32   usageHighWater;           /* percentage of estimated usage at which
                                    time VTMNG_EVENT_VTP_USAGE notif issued */
    U32   usageLowWater;            /* percentage of estimated usage at which
                                    time VTMNG_EVENT_VTP_USAGE notif issued */
    U32   spxMaxChans;              /* maximum number of simplex transcoding
                                    channels to allow [2-<total port licenses>]
                                    (must be even) */
    S32   trcpCount;                /* number of transcoder processes to create
                                    [+ = TRCP count [2-spxMaxChans]; - =
                                    "simplex channels per TRCP" [1-spxMaxChans] */
    S8    mediaAddress[VTMNG_LONGNAME_SZ];    /* IP address used for media endpoint
                                    access to VTP [@|EMPTY = use same IP address
                                    for control and media] */
    U32   maxRtpPayload;            /* maximum size of any outbound RTP packet
                                    payload */
    U32   apiTimeout;               /* TRC API watchdog timeout (time allowed for
                                    TRC API response) [in msecs] (0 = infinite) */
    U32   initTimeout;              /* time (after connect) to wait for INIT REQ
                                    from TRC API [in msecs] (0 = infinite) */
    U32   appLostTimeout;           /* time (after disconnect) before considering
                                    app lost [in msecs] (0 = infinite) */
    U32   debugLogMask;             /* global debug log mask (set of VSLOG_xxx
                                    bits) */
    U32   trcpLogMask;              /* global trcp debug log mask (set of
                                    VSLOG_xxx bits) */
    U32   rtcpInTimeout;            /* default RTCP idle (no RTP or RTCP RX)
                                    input endpoint timeout [in msecs] */
    U32   rtcpOutTimeout;           /* default RTCP idle (no RTCP RX) output
                                    endpoint timeout [in msecs] */
    U32   trapMask;                 /* mask of all event types VTP will issue
                                    traps for (VTMNG_EVENT_xxx) */
```

```
    S8    trapAddress[VTMNG_LONGNAME_SZ]; /* IP address that all asynchronous
                                    indications (traps) are issued to
                                    [@|EMPTY = do not issue any traps;
                                    & = use requester's address] */
    U32   trapPort;                 /* UDP port number that async traps are
                                    issued to (0 = use well-known
                                    [VT_MANAGE_PORT_NOTIF_PORT]) */]
    U8    reserved[16];             /* reserved for future use */
} VTMNG_VTP_CFG;
```

Fields

Field	Type	Description
vtpName	S8	Name of this VTP instance [@|EMPTY = use hostname].
vtpDesc	S8	Video transcoder description (usually version, revision, and so on). [@|EMPTY = trc_agent-generated]
vtpEvent	U8	Optional event to issue to the top-level video transcoder platform controller (VTMNG_VTP_E_***xxx***).
vtpInitState	U8	Set whether video transcoder platform will initialize to a disabled state (VTMNG_VTP_I_***xxx***).
rtcpMode	U8	Set whether channels act as RTCP translators by default. Valid values: VTMNG_CFG_DISABLED VTMNG_CFG_ENABLED
decodePartials	U8	Set whether partial frames should be passed to the decoder. Valid values: VTMNG_CFG_DISABLED VTMNG_CFG_ENABLED
overlayExclusive	U8	Set whether video transcoder platform is reserved for channels requiring overlays. Valid values: VTMNG_CFG_DISABLED VTMNG_CFG_ENABLED
logToConsole	U8	Whether file logging should be forked to console. Valid values: VTMNG_CFG_DISABLED VTMNG_CFG_ENABLED
avail2	U8	Available for future use.
licenseHighWater	U32	Percentage of licenses in use at which time VTMNG_EVENT_VTP_LICENSE notification is issued to indicate high usage of transcoder licenses.
licenseLowWater	U32	Percentage of licenses in use when VTMNG_EVENT_VTP_LICENSE notification is issued to indicate a return to normal license usage levels.
usageHighWater	U32	Percentage of estimated usage when VTMNG_EVENT_VTP_USAGE notification is issued to indicate high CPU usage.

Field	Type	Description
usageLowWater	U32	Percentage of estimated usage when VTMNG_EVENT_VTP_USAGE notification is issued to indicate a return to normal CPU usage.
spxMaxChans	U32	Maximum number of simplex transcoding channels to allow [2 - <total port licenses>] (must be even).
trcpCount	S32	Number of transcoder processes to create [+ = TRCP count [2 -spxMaxChans]; - = "simplex channels per TRCP" [1 - spxMaxChans].
mediaAddress	S8	IP address used for media endpoint access to video transcoder platform [@|EMPTY = Use same IP address for control and media].
maxRtpPayload	U32	Maximum size of any outbound RTP packet payload.
apiTimeout	U32	TRC API watchdog timeout (time allowed for TRC API response) [in milliseconds] (0 = infinite).
initTimeout	U32	Time (after connect) to wait for INIT REQ from TRC API [in milliseconds] (0 = infinite).
appLostTimeout	U32	Time (after disconnect) before considering application lost [in milliseconds] (0 = infinite).
debugLogMask	U32	Global debug log mask (set of agent-specific trace bits).
trcpLogMask	U32	Global trcp debug log mask (set of trcp-specific trace bits).
rtcpInTimeout	U32	Default RTCP idle (no RTP or RTCP RX) input endpoint timeout [in milliseconds].
rtcpOutTimeout	U32	Default RTCP idle (no RTCP RX) output endpoint timeout [in milliseconds].
trapMask	U32	Mask of all event types video transcoder platform will issue traps for (VTMNG_EVENT_***xxx***).
trapAddress	S8	IP address to which all asynchronous indications (traps) are issued: @|EMPTY = Do not issue any traps. ASCII IP address = Issue all traps to a given address.
trapPort	U32	UDP port number to which asynchronous traps are issued: 0 = Use well-known port (VT_MANAGE_NOTIF_PORT)
reserved	U8	Reserved for future use.

VTMNG_VTP_ENTITY

Contains all video transcoder platform-level information.

Definition

```
typedef struct __vtMng_Vtp_Entity
{
    VTMNG_VTP_CFG        cfg;         /* VTP top-level configuration */
    VTMNG_VTP_STATUS     status;      /* current VTP top-level status */
    VTMNG_VTP_STATS      stats;       /* top-level statistics */
    U8                   reserved[16];    /* reserved for future use */
} VTMNG_VTP_ENTITY;
```

Fields

Field	Type	Description
cfg	Structure	Video transcoder platform top-level configuration. Specified in *VTMNG_VTP_CFG* on page 299.
status	Structure	Current video transcoder platform top-level status. Specified in *VTMNG_VTP_STATUS* on page 304.
stats	Structure	Top-level statistics. Specified in *VTMNG_VTP_STATS* on page 303.
reserved	U8	Reserved for future use.

VTMNG_VTP_STATS

Set of statistics maintained at the video transcoder platform-level.

Definition

```
typedef struct __vtMng_Vtp_Stats
{
    U32          errors;         /* count of errors encountered */
    U32          warnings;       /* count of warnings encountered */
    U32          appCount;       /* number of video applications connected to server */
    U32          usageLevel;     /* current usage level of the VTP (expressed as
                                    percent) */
    U32          usedLicenses;   /* number of port licenses that are currently in use */
    U32          spxInUse;       /* current number of simplex channels in use */
    U32          fdxInUse;       /* current number of full-duplex channels in use */
    U8           reserved[16];   /* reserved for future use */
} VTMNG_VTP_STATS;
```

Fields

Field	Type	Description
errors	U32	Count of errors encountered.
warnings	U32	Count of warnings encountered.
appCount	U32	Number of video applications connected to server.
usageLevel	U32	Current usage level of the video transcoder platform (expressed as percent).

Field	Type	Description
usedLicenses	U32	Number of port licenses that are currently in use.
spxInUse	U32	Current number of simplex channels in use.
fdxInUse	U32	Current number of full-duplex channels in use.
reserved	U8	Reserved for future use.

VTMNG_VTP_STATUS

Current video transcoder platform status information.

Definition

```
typedef struct __vtMng_Vtp_Status
{
    U8              vtpState;               /* current overall state (VTMNG_VTP_S_xxx) *
/
    U8              avail[3];
    VSLOG_TIME      vtpStartTime;           /* time when VTP server started */
    U32             vtpLicensedChannels;    /* number of channel (port) licenses that
                                               have been obtained */
    U32             vtpLicensedOverlays;    /* indication of whether VTP is licensed for
                                               overlays */
    U8              reserved[16];           /* reserved for future use */
} VTMNG_VTP_STATUS;
```

Fields

Field	Type	Description
vtpState	U8	Current overall state (VTMNG_VTP_S_***xxx***) values and descriptions: **State** / **Value** / **Description** VTMNG_VTP_S_INITIAL / 0x01 / Initializing. VTMNG_VTP_S_ENABLED / 0x02 / Enabled (new channel assignments allowed). VTMNG_VTP_S_DISABLED / 0x03 / Disabled (no current channels and no new channels allowed). VTMNG_VTP_S_DISABLING / 0x04 / Waiting for current channels to complete before being disabled. VTMNG_VTP_S_ABORTING / 0x05 / Forcing all channels to terminate immediately (hard disable).
avail	U8	Available for future use.
vtpStartTime	Structure	Time when video transcoder platform server started. Uses the VSLOG_TIME record to represent time stamps provided through the VTMNG.
vtpLicensedChannels	U32	Number of purchased channel (port) licenses.
vtpLicensedOverlays	U32	Indication of whether video transcoder platform is licensed for overlays.
reserved	U8	Reserved for future use.

VTMNG_VTPADDR

Provides video transcoder platform addressing information. For requests, this is the video transcoder platform destination address. For responses and traps, this is the video transcoder platform source address.

Definition

```
typedef struct __vtmng_vtpaddr
{
    S8      hostName[VTMNG_LONGNAME_SZ];    /* name of VTP ("1.2.3.4" or "vtp1") */
    U32     ipv4Addr;                       /* IPv4 address */
    U32     sendkey;                        /* used as senderID for requests; gives
                                               received senderID
                                             * for responses and notifications */
    U8      reserved[32];                   /* reserved for future use */
} VTMNG_VTPADDR;
```

Fields

Field	Type	Description
hostName	S8	Name of the video transcoder platform.
ipv4Addr	U32	IPv4 address.
sendkey	U32	Allows the calling application to set a send-specific key. This key is provided back to the caller when the response is received as part of the VTMNG_VTPADDR record for the upcall.
reserved	U8	Reserved for future use.

VTMNG_VTPLVL_NOTIF

Information provided by notifications indicating video transcoder platform-level thresholds are being crossed.

Definition

```
typedef struct __vtMngVtpLvl_Notif
{
    VTMNG_NOTIF_MSG     common;         /* common portion */

    U32                 currLevel;      /* current level */
    U32                 prevLevel;      /* previous level */
    U32                 usedLicenses;   /* number of port license currently in use */
    U32                 spxInUse;       /* number of simplex channels currently in use */
    U32                 fdxInUse;       /* number of full-duplex channels currently
                                           in use */
    U8                  reserved[16];   /* reserved for future use */

} VTMNG_VTPLVL_NOTIF;
```

Fields

Field	Type	Description
common	Structure	Common portion. Specified in *VTMNG_NOTIF_MSG* on page 288.
currLevel	U32	Current level.

Field	Type	Description
prevLevel	U32	Previous level.
usedLicenses	U32	Number of port license currently in use.
spxInUse	U32	Number of simplex channels currently in use.
fdxInUse	U32	Number of full-duplex channels currently in use.
reserved	U8	Reserved for future use.

VTMNG_ZEROSTATS_RSP

Set of all responses to zero current statistics requests.

Definition

```
typedef struct __vtMngZeroStats_Rsp
{
    VTMNG_RSP_MSG       common;     /* common portion */

    union
    {
        VTMNG_VTP_STATS     vtp;    /* VTP top-level statistics (before being
                                       zero'd) */
        VTMNG_APP_STATS     app;    /* application statistics (before being zero'd) */
        VTMNG_MON_STATS     mon;    /* monitored process statistics (before being
                                       zero'd) */
        VTMNG_CHN_STATS     chn;    /* channel statistics (before being zero'd) */
        VTMNG_ST_ENTRY      stats;  /* total statistics (before being zero'd) */
    } u;

} VTMNG_ZEROSTATS_RSP;
```

Fields

Field	**Type**	**Description**
common	Structure	Common portion. Specified in VTMNG_RSP_MSG structure.
u	Union	Valid substructures: vtp: Video transcoder platform top-level statistics (before being zeroed). Specified in *VTMNG_VTP_STATS* on page 303. app: Application statistics (before being zeroed). Specified in *VTMNG_APP_STATS* on page 267. mon: Monitored process statistics (before being zeroed). Specified in *VTMNG_MON_STATS* on page 284. chn: Channel statistics (before being zeroed). Specified in *VTMNG_CHN_STATS* on page 276. stats: Total statistics (before being zeroed). Specified in *VTMNG_ST_ENTRY* on page 295.

12. Errors, events, and log files

Handling errors

This topic describes how to handle the following types of errors:

- Connection errors
- TRC agent errors
- Transcoder process errors

Connection errors

Errors in which the TRC loses its connection to a video transcoder platform are reported through the TRC in the form of resource change events and through **trcVTPStatus**.

The following table provides a series of troubleshooting actions to take if this occurs:

Step	If...	Then...
1	The application is unable to connect to a video transcoder platform.	The application can use **trcResetVTP** to reboot the video transcoder platform remotely.
2	The remote reboot attempt is not successful.	The video transcoder platform must be rebooted locally.
3	If a reboot of the video transcoder platform does not correct the problem.	Verify the IP address of the video transcoder platform that appears in the TRC configuration file. Check the cabling between the chassis hosting the application and the video transcoder platform. Examine the local IP address configuration of the video transcoder platform.

Do not reset a video transcoder platform because connectivity was lost with that video transcoder platform. All channels in use on a given video transcoder platform will fail if a video transcoder platform is reset. These channels include all channels owned by the calling application, as well as all channels owned by other applications sharing the same video transcoder platform. The decision to reset a video transcoder platform should be considered carefully.

TRC agent errors

The trc_agent reports error conditions in the agent's transcoder log file */opt/nms/video/logs/xc.log*. If errors are encountered related to video transcoding on a particular video transcoder platform, examine the agent's transcoder log file. If error indications are not present in the agent's log file, search the full set of process log files for any reported errors.

Transcoder process errors

The transcoder processes report error conditions into channel-specific log files:

```
/opt/nms/video/logs/xc.log_01...(max trcp)
```

Any errors that the process encounters are logged.

TRC error summary

All functions return a status code. If the return code is not TRC_SUCCESS (0), it is an error code indicating that the function failed and the reason for the failure. TRC error codes are defined in the *trcdefs.h* include file. The error codes are prefixed with TRCERR.

The following table lists the TRC errors. All errors are 32 bits.

Error	Description
TRC_OVLEVT_TRCP_DOES_NOT_EXIST	The request failed because the direction provided does not exist.
TRC_OVLEVT_TRCP_ENCODER	The encoder configuration failed.
TRC_OVLEVT_TRCP_INVALID_OVL_DATA	An overlay or a content configuration parameter was invalid. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_TRCP_INVALID_OVL_HANDLE	The supplied overlay handle is unknown.
TRC_OVLEVT_TRCP_INVALID_STATE	Overlay is not in the valid state to process the request.
TRC_OVLEVT_TRCP_INVALID_TYPE	Channel type is not appropriate for this request.
TRC_OVLEVT_TRCP_MAKE_HEADER	An error occurred while trying to create the MPEG-4 headers.
TRC_OVLEVT_TRCP_NO_RESOURCE	A resource shortage caused the request to fail.
TRC_OVLEVT_TRCP_NOT_SUPPORTED	Request is not supported.
TRC_OVLEVT_TRCP_OUT_OF_MEMORY	Unable to allocate memory necessary to process request.
TRC_OVLEVT_TRCP_OVL_CREATE_FAILED	Overlay creation failed.
TRC_OVLEVT_TRCP_SOCKET	An error occurred while trying to send a request to the render process.
TRCERR_ALREADY_INTIALIZED	TRC library has already been initialized.

Error	Description
TRCERR_FAILURE	Generic TRC error.
TRCERR_FUNCTION_NOT_SUPPORTED	Requested function is not currently supported.
TRCERR_INVALID_APPNAME	Invalid application name was provided to **trcInitialize**.
TRCERR_INVALID_CHANNEL_HANDLE	TRC channel handle provided does not correspond to a created channel.
TRCERR_INVALID_CHANNEL_PARAM	One or more of the parameters provided to a TRC function is invalid.
TRCERR_INVALID_CHANNEL_STATE	Channel is not in a valid state for the requested operation.
TRCERR_INVALID_CONFIG_FILE	Invalid configuration file.
TRCERR_INVALID_DATA_RATE	Invalid data rate.
TRCERR_INVALID_DIRECTION	Direction provided to a TRC function is invalid.
TRCERR_INVALID_FILE	Invalid configuration file provided to **trcInitialize**, or the image file is reported as invalid on the given video transcoder platform.
TRCERR_INVALID_FRAME_RATE	Invalid frame rate.
TRCERR_INVALID_FRAME_RES	Invalid frame resolution.
TRCERR_INVALID_FUNC	Invalid pointer to the callback function provided to **trcInitialize**.
TRCERR_INVALID_INPUT_PARAM	A parameter provided to **trcStartVideoChannel** related to transcoder input control is invalid.
TRCERR_INVALID_JITTER_MODE	Invalid input (to the transcoder) jitter mode.
TRCERR_INVALID_LEVEL	Invalid profile level.
TRCERR_INVALID_OUT_IPADDR	Invalid output from the transcoder IP address.
TRCERR_INVALID_OUT_OPTION	Invalid output endpoint option.

Error	Description
TRCERR_INVALID_OUTPUT_PARAM	A parameter provided to **trcStartVideoChannel** related to transcoder output control is invalid.
TRCERR_INVALID_PACKETIZE	Invalid packetization mode.
TRCERR_INVALID_PROFILE	Invalid profile type.
TRCERR_INVALID_TYPE	Invalid channel type was provided to **trcCreateVideoChannel**.
TRCERR_INVALID_VERSION	A version mismatch was detected during a call to **trcInitialize**. Rebuild the application against the current TRC version.
TRCERR_INVALID_VIDEO_TYPE	Invalid video type. Video type should be H.263 or MPEG-4.
TRCERR_INVALID_VTP_ID	Video transcoder platform identifier provided to the TRC function is not a valid video transcoder platform ID.
TRCERR_LIB_NOT_INITIALIZED	TRC library is not initialized. Call **trcInitialize** first.
TRCERR_NORESOURCES	No transcoder resources are available.
TRCERR_OPERATION_ABORTED	Given operation was aborted
TRCERR_OUT_OF_MEMORY	Not enough system memory to complete the request.
TRCERR_PROCESS_ERROR	Error detected with the given process.
TRCERR_PROCESS_LOST	A transcoder process was lost due to an unexpected termination.
TRCERR_RESET_FAILED	**trcResetVTP** was unable to pass the reset request to the video transcoder platform. The video transcoder platform cannot be restarted remotely and must be restarted locally.
TRCERR_RING_FULL	Ring buffer is full.

Error	Description
TRCERR_SOCKET_FAILURE	TRC was unable to open a TCP/IP socket to a video transcoder platform, or it encountered an error while reading data from a video transcoder platform socket.
TRCERR_THREAD_USAGE	Improper thread usage by application. API calls cannot be made from within the asynchronous event upcall notification function.
TRCERR_UNEXPECTED_MSG	A message of a known type was encountered when not expected.
TRCERR_UNKNOWN_MSG	An unknown message type was encountered.
TRCWARN_FUTURE_REVISION	Warning to the application indicating that the revision of the TRC is older than the revision that the application was built against. This warning can be ignored, with the understanding that requested features are limited to those provided by the TRC revision only.

Management error summary

All VTMNG functions return a status code. If the return code is not VS_SUCCESS (0), it is an error code indicating that the function failed and the reason for the failure. Management error codes are defined in the *vtmng.h* include file.

The following table lists the management errors. All errors are 32 bits.

Error	Description
VTMNG_ERR_DOES_NOT_EXIST	Item indicated does not exist.
VTMNG_ERR_INVALID_SIZE	Size indication out of acceptable range.
VTMNG_ERR_INVALID_STATE	Invalid state for requested operation.
VTMNG_ERR_INVALID_TYPE	Invalid type encountered.
VTMNG_ERR_LIST	List-related error.
VTMNG_ERR_NO_RESOURCE	Unable to obtain required resource.
VTMNG_ERR_NOT_IMPLEMENTED	Requested function not yet implemented.
VTMNG_ERR_NOT_INITIALIZED	Module has not been initialized.

Error	Description
VTMNG_ERR_NOT_SUPPORTED	Requested function not currently supported.
VTMNG_ERR_OUT_OF_MEMORY	Insufficient memory to complete request.
VTMNG_ERR_OUT_OF_RANGE	Value encountered that is out of valid range.
VTMNG_ERR_POLL	Failure during poll operation.
VTMNG_ERR_SOCKET	Communication socket reported error.
VTMNG_ERR_STATE_EVENT	State / event error.
VTMNG_ERR_UNKNOWN_TYPE	Unknown field type encountered.
VTMNG_WRN_DISCONNECT	Error due to connection disconnect.
VTMNG_WRN_FULL	Currently full.
VTMNG_WRN_INVALID_CONFIG	Invalid configuration entry encountered.
VTMNG_WRN_TIMEOUT	Idle timeout period elapsed without activity.
VTMNG_WRN_UNEXPECTED	Item of known type encountered when given type not expected.

Transcoder resource controller events

The transcoder resource controller module can send the events listed in this topic to the application using the user-defined callback function. TRC events are defined in the *trcapi.h* include file.

All TRC events use the tTrcMessage structure. For more information about this structure, refer to *tTrcMessage* on page 239.

The following table provides an alphabetical listing of the TRC events:

Event	Description
TRCEVN_CHANNEL_FAILED	An active transcoding channel failed. The application must destroy the channel using **trcDestroyVideoChannel**.
TRCEVN_CHANNEL_LOST	A channel connection was lost. The application can wait for the channel to be automatically recovered by the TRC.
TRCEVN_CHANNEL_OVL_EVENT	Indicates an asynchronous overlay event. These events can either be informational or indicate that an error occurred while processing an overlay. The specific type of event is reported in the field result of the tTrcMessage structure.

Event	Description
TRCEVN_CHANNEL_RECOVERED	A channel previously reported as lost was recovered. The failure was due to an interruption in communication with the video transcoder platform on which the channel resides. No interruption in transcoding has occurred.
TRCEVN_CHANNEL_RTCP_BYE	Channel has received an RTCP BYE message.
TRCEVN_CHANNEL_RTCPTIMEOUT	Channel RTCP layer has detected a timeout related to RTP/RTCP reception.
TRCEVN_CREATE_CHANNEL_DONE	Completion of a call to **trcCreateVideoChannel**.
TRCEVN_CREATE_OVL_DONE	Overlay creation request is complete.
TRCEVEN_DESTROY_CHANNEL_DONE	Completion of a call to **trcDestroyVideoChannel**.
TRCEVN_DESTROY_OVL_DONE	**trcDestroyOverlay** is complete. The overlay or overlays are no longer being displayed and all related resources were released.
TRCEVN_IFRAME_CHANNEL_DONE	Completion of a call to **trcIframeVideoChannel**. This event returns the following data: TRCDATA_IFRAME_CHANNEL_DIRECTION Direction for which the I-frame was generated.
TRCEVN_RESOURCE_CHANGE	Number of available transcoder resources (port licenses) has changed. This change may be due to additional transcoder resources becoming available or previously available transcoder resources being lost. **Note:** The trcChHandle and userKey fields are not used for this event.
TRCEVN_SHUTDOWN_DONE	Completion of a call to **trcShutdown**. **Note:** The trcChHandle and userKey fields are not used for this event.
TRCEVN_START_CHANNEL_DONE	Completion of a call to **trcStartVideoChannel**.
TRCEVN_START_OVL_DONE	**trcStartOverlay** is being processed.

Event	Description
TRCEVN_STOP_CHANNEL_DONE	Completion of a call to **trcStopVideoChannel**.
TRCEVN_STOP_OVL_DONE	Indicates that the overlay is no longer being displayed.

Overlay event result codes

The TRCEVN_CHANNEL_OVL_EVENT transcoder resource controller event indicates that an overlay-related event has occurred. The result field of the tTrcMessage structure identifies the specific type of overlay event that occurred. Overlay events are defined in the *trcapi.h* include file.

TRCEVN_CHANNEL_OVL_EVENT uses the tTrcMessage structure. For more information about this structure, refer to *tTrcMessage* on page 239.

The following table provides an alphabetical listing of specific overlay events. In each case, the overlay for which the event is issued is identified by TRCDATA_OVERLAY_OVLHANDLE and TRCDATA_OVERLAY_USERKEY in the data field of the tTrcMessage:

Event	Description
TRC_OVLEVT_TRCP_INVALID_OVL_DATA	Either an overlay or a content configuration parameter was invalid. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_TRCR_RENDER_FAILED	The rendering process was unable to create the requested rendered content. Check the *trcr.log* and *xc.log_nn* file for error messages.
TRC_OVLEVT_TRCR_RENDER_SUCCESS	The content associated with the overlay was successfully rendered and is ready to display. The content is overlaid unless the overlay is in the stopped state.
TRC_OVLEVT_VTC_CREATEOVL_FAILED	An internal request to create the overlay has failed. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_VTC_DESTROYOVL_FAILED	An internal request to destroy the overlay has failed. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_VTC_SCROLL_END	The overlay has finished scrolling.
TRC_OVLEVT_VTC_STARTOVL_FAILED	An internal request to start the overlay has failed. Check the *xc.log_nn* file for an error message.

Event	Description
TRC_OVLEVT_VTC_STOPOVL_FAILED	An internal request to stop the overlay has failed. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_VTC_SUBMITC_FAILED	An internal request to submit the rendered content has failed. Check the *xc.log_nn* file for an error message.
TRC_OVLEVT_VTC_SUBMITF_FAILED	Not used.

Management events

VTMNG events are handled by a management application. The TRC Control Agent (*trc_agent*) and the Process Monitor (*vtmon*) can issue asynchronous notifications to a manager application. These notifications are often referred to as traps. The video transcoder platform-level configuration includes the following fields for controlling video transcoder platform trap output:

Field	Description
trapMask	Mask of all trap types to be issued (see event table).
trapAddress	IP address that all traps are issued to.
trapPort	UDP port number that traps are issued to.

The following table provides an alphabetical listing of the management events:

Event	Description
VTMNG_EVENT_APP_CONNECT	A control application has connected to the video transcoder platform. The following information is provided: app: Application management entity (current configuration, status, and statistics).
VTMNG_EVENT_APP_DEAD	An application connection has been destroyed. The following information is provided: app: Application management entity (current configuration, status, and statistics).
VTMNG_EVENT_APP_DISCON	A previously established control application connection has been lost. The following information is provided: app: Application management entity (current configuration, status, and statistics).

Event	Description
VTMNG_EVENT_APP_NAMED	The name of a newly connected application has been determined. The following information is provided: app: Application management entity (current configuration, status, and statistics).
VTMNG_EVENT_CHN_CREATE	A video channel has been created. The following information is provided: appUnique: Unique value identifying the owning application (same as the application's appUnique). chnUnique: Unique value identifying this channel. chnName: Optional name applied to channel (assigned by controlling application). chnBasicType: Basic channel type (TRC_CH_SIMPLEX|TRC_CH_FDX). overlayReq: Channel overlay requirements (TRC_CH_OVERLAY_***xxx***).
VTMNG_EVENT_CHN_DEAD	A channel was terminated (no additional information).
VTMNG_EVENT_CHN_ERROR	A channel-level error has been detected. The following information is provided: errCode: Error code associated with event. severity: Severity level of notification (VSLOG_SEV_***xxx***). options: Options associated with the event (VSLOG_EVT_***xxx***). alarmState: State associated with alarm condition (VSLOG_ALARM_***xxx***). locName: Optional. Name of location where error was generated. notifDesc: Text description of error. stats: Snapshot of statistics after error was detected.
VTMNG_EVENT_CHN_START	A channel has been started. The following information is provided: cfg: Configuration in use by channel. status: current status information [remote RTP addressing not yet known].

Event	Description
VTMNG_EVENT_CHN_STOP	A channel has been stopped. The following information is provided: cfg: Configuration in use by channel. status: Status information at the time the channel was stopped. stats: Statistics at time channel was stopped. Can be used as call detail record.
VTMNG_EVENT_MON_LOST	A monitored process has been lost (automatic process recovery in progress). The following information is provided: mon: Monitored process management entity (current configuration, status, and statistics).
VTMNG_EVENT_MON_RECOVER	A monitored process that had been reported as lost has now been recovered. The following information is provided: mon: Monitored process management entity (current configuration, status, and statistics).
VTMNG_EVENT_VTP_ERROR	A video transcoder platform-level error has occurred. The following information is provided: errCode: Error code associated with event. severity: Severity level of notification (VSLOG_SEV_***xxx***). options: Options associated with the event (VSLOG_EVT_***xxx***). alarmState: State associated with alarm condition (VSLOG_ALARM_***xxx***). locName: Optional name of location where error was generated. notifDesc: Text description of error.

Event	Description
VTMNG_EVENT_VTP_LICENSE	This event indicates that license usage for the given video transcoder platform has crossed a threshold (either the high water or low water mark). licenseHighWater: Configuration field - license usage high water mark. marklicenseLowWater: Configuration field - license usage low water mark. The following information is provided: currLevel: Current number of total licenses in use for the given video transcoder platform. prevLevel: Previous number of licenses in use. usedLicenses: Same as currLevel. spxInUse: Number of simplex channels currently in use. fdxInUse: Number of full-duplex channels currently in use.
VTMNG_EVENT_VTP_USAGE	This event indicates that the estimation of total CPU usage required to perform all current transcoding has crossed a threshold (either the high water or low water mark). usageHighWater: Configuration field. Estimated CPU usage high water mark. markusageLowWater: Configuration field. Estimated CPU usage low water mark. The following information is provided: currLevel: Current estimate of required CPU usage (percent) prevLevel: Previous estimated CPU usage. usedLicenses: Current number of total licenses in use for the given video transcoder platform. spxInUse: Number of simplex channels currently in use. fdxInUse: Number of full-duplex channels currently in use.

Using log files

All transcoder log files are created in the */opt/nms/video/logs* directory of each video transcoder platform.

When the video transcoder platform starts up, the */opt/nms/video/logs* directory is created. This directory holds all current transcoder logs. If the log directory already exists at startup time, the current log directory is renamed using a cycling history extension of .1 (moving any current .1 to .2, and so on). The depth of these archived log directories is configured through *vtmon* (see variable definition logDirs). If a log directory depth exceeds the configured maximum, then the oldest log directory is deleted.

The trc_agent log file name is set to *xc.log*. Any errors detected by the agent are reported in this log file.

Each transcoder process (trcp) can manage a number of transcoder channels. A separate log file is created by each trcp using the following format:

xc.log_<unique process number>

For example,

```
xc.log_01 and xc.log_02
```

The process monitor uses a log file named *vtmon.log*. Any errors detected by the process monitor are reported in this log file.

The overlay text and image render process uses a log file named *trcr.log*. Errors detected by the renderer are reported in this log file.

Diagnostic logging options

Normally, logging levels should be set to log any detected error or warning conditions. These log categories are always defined as:

- BIT 0: [ERR] ERROR
- BIT 1: [WRN] WARNING

This is why all video transcoder processes and APIs default to a log mask of 3. Additional log bits are provided to assist in certain diagnostic situations. For the TRC API, trace types are defined in *trfcdefs.h* (TRCTR_***xxx***) with the additional trace output being logged to the file name provided to **trcInitialize**.

trc_agent trace types

The trace types for the trc_agent are set using the video transcoder platform-level configuration field debugLogMask and include the following options:

Tag	Bit in logMask	Description
ERR	0	Trace any detected error.
WRN	1	Trace warning indications.
API	2	Trace control interface (TRC API).
COM	3	Trace management interface.

Tag	Bit in logMask	Description
CTL	4	Trace control information.
FSM	5	Finite state machine tracing.
HDR	6	Not currently used.
DAT	7	Add hexadecimal/ASCII trace of message data (to COM|API).
INI	8	Initialization tracing.
TOP	9	Top-level (video transcoder platform-level) information.
APP	10	Application connection related information.
CHN	11	Channel related information.
DBG	12	Trace low-level debug information.
HIS	16	Trace histogram maintenance.

Note: The list of options supported by trc_agent can always be viewed by using the help option (trc_agent -h).

trcp trace types

The trace types for the trcp are set using the video transcoder platform-level configuration field trcpLogMask and include the following options:

Tag	Bit in logMask	Description
ERR	0	Trace any detected error.
WRN	1	Trace warning indications.
API	2	Trace channel configuration, and so on.
COM	3	Trace encode/decode layer.
CTL	4	Trace control information.
FSM	5	Finite state machine tracing.
HDR	6	MPEG4 header tracing.
DAT	7	Add hexadecimal/ASCII trace of message data (to COM|TOP_APP).
INI	8	Initialization tracing.
TOP	9	Top-level (video transcoder platform-level) information.

Tag	Bit in logMask	Description
APP	10	trc_agent connection state and message summary.
CHN	11	Channel statistics and completion summary.
DBG	12	Trace low-level debug information.
RCM	16	Trace RTCP messages sent and received.
RCD	17	Trace RTCP low-level debug information.
RTI	18	Trace RTP input.
RTO	19	Trace RTP output.
XCL	20	Trace transcoder library data stream errors.
MFL	20	Trace transcoder message flow.
E2F	21	Trace binary data sent to encoder leg. This option causes the trcp to generate a binary file (named enc_<*trcpId*>_<*slotID*>_<*date/time*>) in the logs directory.
D2F	22	Trace binary data received from decoder leg. This option causes the trcp to generate a binary file (named dec_<*trcpId*>_<*slotID*>_<*date/time*>) in the logs directory.

Note: The list of options supported by trcp can always be viewed by using the help option (trcp -h).

trcr trace types

The trace types for the trcr are set through the debuglogmask parameter in the [Global] section of the *trcr.cfg* configuration file and include the following options:

Tag	Bit in logMask	Description
ERR	0	Trace any detected error.
WRN	1	Trace warning indications.
API	2	Not currently used.
COM	3	Trace management interface.
CTL	4	Not currently used.
FSM	5	Finite state machine tracing.
HDR	6	Not currently used.
DAT	7	Add hexadecimal/ASCII trace of message data to COM.

Tag	Bit in logMask	Description
INI	8	Initialization tracing.
TOP	9	Top-level information.
APP	10	Not currently used.
CHN	11	Not currently used.
DBG	12	Trace low-level debug information.

Process monitor trace types

The trace types for the process monitor are set using the d command line option to *vtmon.* For example, vtmon d 0x020B would activate TOP, COM, WRN and ERR.

Tag	Bit in logMask	Description
ERR	0	Trace any detected error.
WRN	1	Trace warning indications.
API	2	Not currently used.
COM	3	Trace management interface.
CTL	4	Not currently used.
FSM	5	Finite state machine tracing.
HDR	6	Not currently used.
DAT	7	Add hexadecimal/ASCII trace of message data to COM.
INI	8	Initialization tracing.
TOP	9	Top-level information.
APP	10	Not currently used.
CHN	11	Not currently used.
DBG	12	Trace low-level debug information.

Note: The list of options supported by vtmon can always be viewed by using the help option (vtmon -h).

Log file errors

See *Handling errors* on page 307 for a description of the types of errors that can be seen in log files.

As part of periodic maintenance, the video transcoder platform logs can be checked for errors. All errors begin with the text, ERROR, to allow for easy searching. Certain problems are also reported as warnings. These begin with the text, WARNING.

The trc_agent also logs problems related to communication with transcoder processes. These indications include the text, TRCP. These indications are usually self-correcting, since the agent restarts a trcp if problems are detected.

Log file banners

Each log file includes banner entries similar to the following examples.

Agent log banner example

The following example shows an agent log banner:

```
=========================================================================================
Log File: logs/xc.log
DATE: Mar 09,2007  TIME: 07:55:19
MMM dd hh:mm:ss(mmm) <s> [hostName] [sysI-sysInst ] [subI] [bit|event] {ent-loc}: text
-----------------------------------------------------------------------------------------
Mar 09 07:55:19(492) <D> [v1] [AGNT] [LOGS] [FIL]: ----- Agent V2.28 Configuration ------
<heading> [FIL]: Simplex Transcoding Channels: 60
<heading> [FIL]:              Number of TRCPs: 30 (2 simplex channels per TRCP)
<heading> [FIL]:    Name of this VTP Instance: vtp4
<heading> [FIL]:              VTP Description: NMS Video Transcoder:
                                   Server Control Agent version 2.28
<heading> [FIL]:            VTP Initial State: ENABLED
<heading> [FIL]:            Default RTCP Mode: DISABLED
<heading> [FIL]:        Decode Partial Frames: DISABLED
<heading> [FIL]:            Exclusive Overlay: DISABLED
<heading> [FIL]:          Fork Log to Console: DISABLED
<heading> [FIL]:      License High Water Mark: 80 percent
<heading> [FIL]:      License Low  Water Mark: 60 percent
<heading> [FIL]: Estim. Usage High Water Mark: 80 percent
<heading> [FIL]: Estim. Usage Low  Water Mark: 60 percent
<heading> [FIL]:                Media Address: NO SEPARATE ADDRESS
<heading> [FIL]: Maximum Outbound RTP Payload: 1342 bytes
<heading> [FIL]:     TRC API Watchdog Timeout: 5000 msecs
<heading> [FIL]:    Wait for INIT REQ Timeout: 5000 msecs
<heading> [FIL]: Wait Before APP Lost Timeout: 300000 msecs
<heading> [FIL]:         Agent Debug Log Mask: 0x00010003
<heading> [FIL]:          TRCP Debug Log Mask: 0x00000003
<heading> [FIL]:  RTCP Idle In  Endpt Timeout: 0 msecs
<heading> [FIL]:  RTCP Idle Out Endpt Timeout: 0 msecs
<heading> [FIL]:                    Trap Mask: 0x00100007
<heading> [FIL]:                 Trap Address: NO TRAPS ISSUED
<heading> [FIL]
<heading> [FIL]: -------------------------- Current Status --------------------------
<heading> [FIL]:    Current Overall VTP State: ENABLED
<heading> [FIL]:         TRC Agent Start Time: Mar 09 07:55:19(491)
<heading> [FIL]:            Licensed Channels: 60 port licenses obtained
<heading> [FIL]:              Overlay License: ENABLED
<heading> [FIL]
<heading> [FIL]:
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]: !!! Beginning main processing loop !!!
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]:
```

Transcoder process log banner example

The following example shows a transcoder process log banner:

```
=========================================================================================
Log File: logs/xc.log_01
DATE: Mar 09,2007  TIME: 07:55:19
MMM dd hh:mm:ss(mmm) <s> [hostName] [sysI-sysInst ] [subI] [bit|event] {ent-loc}: text
-----------------------------------------------------------------------------------------
Mar 09 07:55:19(492) <D> [v1] [TRCP] [LOGS] [FIL]:
<heading> [FIL]: Command Line: trcp 1 2 7 0x00000003 0 1342 0 1 trcp.log 9 0
<heading> [FIL]:   where:  First Simplex Chan No.  = 1
<heading> [FIL]:           Number of Simplex Chans = 2
```

```
<heading> [FIL]:           Control Socket ID       = 7
<heading> [FIL]:           Debug Log Mask          = 0x00000003
<heading> [FIL]:           Log to Console Flag     = 0
<heading> [FIL]:           Max RTP Payload Size    = 1342
<heading> [FIL]:           Write Pixels            = 0
<heading> [FIL]:           Overlay support         = 1
<heading> [FIL]:           Log Filename            = trcp.log
<heading> [FIL]:           Manager Socket ID       = 9
<heading> [FIL]:           Decode Partial Frames   = 0
<heading> [FIL]
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]: !!!! TRCP 01 [V2.10] started successfully !!!!
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]
```

Render process log banner example

The following example shows a render process log banner:

```
==========================================================================================
Log File: logs/trcr.log
          DATE: Apr 20,2007  TIME: 19:54:04
MMM dd hh:mm:ss(mmm) <s> [hostName ] [sysI-sysInst ] [subI] [bit|event] {ent-loc}: text
------------------------------------------------------------------------------------------
<heading> [FIL]
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]: !!!! TRCR [V1.1] started successfully !!!!
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]
```

Process monitor log banner example

The following example shows a vtmon (process monitor) log banner:

```
==========================================================================================
Log File: logs/vtmon.log
          DATE: Apr 20,2007  TIME: 19:54:04
MMM dd hh:mm:ss(mmm) <s> [hostName ] [sysI-sysInst ] [subI] [bit|event] {ent-loc}: text
------------------------------------------------------------------------------------------
<heading> [FIL]: ---------- Video Transcoder Process Monitor V1.2 ----------
<heading> [FIL]: Process trc_agent [PID 13618 ] started successfully
<heading> [FIL]: Process trcr [PID 13619 ] started successfully
<heading> [FIL]: Process xscontrol [PID 13620 ] started successfully
<heading> [FIL]: VTP Entered MONITORING_state
<heading> [FIL]:
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]: !!! Beginning main processing loop !!!
<heading> [FIL]: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
<heading> [FIL]:
```

13. Using the management utility

vtmgr - Management utility overview

The management utility (*vtmgr*) provides a text-based operator command interface to the VTMNG API. It is provided as source code and as an executable file that can be used as a VTMNG API sample application.

The *vtmgr* acts as a fully-functional operator console tool, a trap aggregator and logging tool, or both.

The vtmgr is located in:

Operating system	**Path**
Windows	The executable *vtmgr.exe* and the dynamically linked library *vtmngapi.dll* are installed into the location specified at install time (for example, *c:\NMS\bin)*.
UNIX	*/opt/nms/bin* The environment variable LD_LIBRARY_PATH must include */opt/nms/lib.*

Usage

[-v <***vtpAddr***>] [-f <***eventLog***>] [-r <***reqPort***>] [-t <***trapPort***>]
[-c <***chnTrcLvl***> [-d <***dbgMask***>] [-R] [-T] [-K]

```
>vtmgr -v 10.3.6.164
```

Valid options are:

Option	Description
-v <*vtpAddr*>	Set destination video transcoder platform address.
-f <*eventLog*>	Set the file name to log asynchronous events to and register for traps.
-r <*reqPort*>	Specify UDP port to issue requests and receive responses over. The default is to select any available port. If -r is not specified (the default), the operating system registers for any available UDP port.
-t <*trapPort*>	Specify UDP port number to listen on for asynchronous traps. The default is to not listen for traps. Note: If the optional eventlog file is specified (using -f <*eventLog*), the *vtmgr* registers to receive traps regardless of whether it is specified. If –t is not specified, the *vtmgr* will register to receive traps on the well-known management trap notification port (VT_MANAGE_NOTIF_PORT as defined in *vsport.h*).

Option	Description
-c *<chnTrcLvl>*	Specify the level of information that is traced on channel events. Valid values for *chnTrcLvl* include: off = no information is traced on channel events. state = trace channel state change information only (default). all = trace all channel-level indication information.
-d *<dbgMask>*	Specify debug log mask: Bit 0 = log any errors detected. Bit 1 = log any warnings detected. Note: No other bits are defined by default. Specify -d ? to view the set of optional logging bits.
-R	Register for management response handling.
-T	Register for management trap handling.
-K	Register for keyboard command handling.

The -R, -T, or -K options allow *vtmgr* to be a sample application that provides all of the hooks necessary to begin developing a management tool.

Note: When *vtmgr* is run with the -R, -T, or -K options specified, it becomes less useful as a utility because it uses more primitive parsing than is possible when VTMNG is given complete control.

vtmgr commands

vtmgr supports the following commands:

Command	Description
app	Display all information for an application connection.
apps	Display list of all current application connections.
cfg	Display configuration of a specific channel.
chan	Display all information for a specific channel or event channel.
chans	Display list of all defined channels.
currmin	Display statistics being collected for current one-minute period.
dest	Set destination of outbound requests.
hhr	Display statistics histogram entry from *x* half-hours ago.

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td>last</td><td>Display histogram summary for last hour or last day. Syntax options are:
vtmgr> last hour
vtmgr> last day
<table>
<tr><th>If the operator says...</th><th>A summary report appears containing the...</th></tr>
<tr><td>last hour</td><td>Last hour period in one-minute increments (60 lines of output).</td></tr>
<tr><td>last day</td><td>Last 24-hour period in half-hour increments (48 lines of output).</td></tr>
</table>
</td></tr>
<tr><td>min</td><td>Display statistics histogram entry from x minutes ago.</td></tr>
<tr><td>mon</td><td>Display all information for a monitored process or event monitor.</td></tr>
<tr><td>mons</td><td>Display list of all currently monitored processes.</td></tr>
<tr><td>prompt</td><td>Toggle whether prompt displayed after asynchronous indications.</td></tr>
<tr><td>quit</td><td>Quit (exit) the application.</td></tr>
<tr><td>result</td><td>Display the ASCII name associated with a given result value.</td></tr>
<tr><td>stats</td><td>Display current statistics of a specific channel.</td></tr>
<tr><td>status</td><td>Display current status of a specific channel.</td></tr>
<tr><td>total</td><td>Display total (overall) statistics.</td></tr>
<tr><td>vtp</td><td>Display video transcoder platform top-level information or issue a video transcoder platform event.</td></tr>
<tr><td>zero</td><td>Get current statistics then zero statistics.</td></tr>
</table>

Using vtmgr command help

Use the question mark (?) command to view a list of the command sets that are provided by *vtmgr*. The default command set is [vtmgr].

Command sets allow for like commands to be grouped together. This allows the help summary command (?) to limit the output. Any command from any command set can be executed regardless of the selected set.

The vtmgr command sets are:

Command set	Description
[vtmgr]	Management commands.

Command set	Description
[vtp]	Video transcoder platform set control.
[mon]	Monitor set control.

To view a specific command set help summary, enter the command set before you enter the help command.

Use the name in brackets to select a different command set:

```
vtmgr> [vtp]
vtmgr[vtp]>
vtmgr[vtp]> [mon]
vtmgr[mon]>
```

To return to the default command set, enter the [] command:

```
vtmgr[mon]> []
vtmgr>
```

Changing the command set alters what appears when you enter the help summary command (?):

```
vtmgr [vtp]> ?
[vtp] VTP set control:
  vtpName             - set the name of this VTP instance [@ = use hostname]
  vtpDesc             - set the VTP description (version,etc)[@ = trc_agent-generated]
  vtpInitState        - set VTP init state (whether VTP will init to a disabled state)
  rtcpMode            - set whether VTP acts as an RTCP translator by default
  decodePartials      - set whether partial frames should be passed to the decoder
  licenseHigh         - set percent of licenses in use when VTP_LICENSE notified
  licenseLow          - set percent of licenses in use when VTP_LICENSE notified
  usageHigh           - set percent of estimated usage when VTP_USAGE notified
  usageLow            - set percent of estimated usage when VTP_USAGE notified
  spxMaxChans         - set maximum number of simplex channels to allow (must be even)
  trcpCount           - set number of TRCPs to create [+ = count; - = spx_per_trcp]
  mediaAddress        - set IP address for media endpoint access to VTP [@ = use ctl]
  maxRtpPayld         - set maximum size of any outbound RTP packet payload
  apiTimeout          - set time for TRC API response [in msecs] (0 = infinite)
  initTimeout         - set time for INIT REQ from TRC API [in msecs] (0 = infinite)
  appLostTimeout      - set time (after discon) when app lost [msecs] (0 = infini)
  debugLogMask        - set global debug log mask (set of VSLOG_xxx bits)
  trcpLogMask         - set global trcp debug log mask (set of VSLOG_xxx bits)
  logToConsole        - set whether to-file logging should be forked to console
  rtcpInTime          - set default RTCP idle (no RTP,RTCP RX) input endpoint timeout
  rtcpOutTime         - set default RTCP idle (no RTCP RX) output endpoint timeout
  trapMask            - set mask of events VTP will issue traps for (VTMNG_EVENT_xxx)
  trapAddress         - set IP address that all async traps issued to [@ = no traps]
  trapPort            - set UDP port number that async traps issed to [@ = well-known]
  quit                - quit (exit) the application

vtmgr[mon]> ?
[mon] MON set control:
  monName             - set the name of this monitored process
  monInitState        - set process monitor initial state
  exec                - set the name of executable file for this process
  cmdLine             - set the command-line options provided on process creation
  var                 - set the name and string value of one of 8 per-process variables

  quit                - quit (exit) the application
```

Management utility tasks

Use the management utility to perform the following tasks:

Task	Description	Commands
vtmgr general control	Set destination for management requests, control prompt verbosity, or quit from *vtmgr*. vtmgr> dest vtmgr> prompt vtmgr> quit	dest: Display current destination IP address. dest <*IP addr*>: Set new destination for all outbound requests. prompt: Toggle prompt verbosity. quit: Quit (exit) the application.
Video transcoder platform-level configuration and control	View all video transcoder platform-level configuration, status and statistics. Modify any video transcoder platform-level configuration.	vtp: Display video transcoder platform top-level information. vtp enable: Enable

Task	Description	Commands
	vtmgr> vtp vtmgr> vtp <event> vtmgr> (all [vtp] commands)	channel assignment. vtp disable: Disable channel assignment. vtp abort: Abort all channels and disable. vtp restart: Stop all transcoder processes and then restart (requires *vtmon* activation). vtp reboot: Cause the video transcoder platform to reboot. [vtp] commands: Modify the video transcoder platform-level configuration.
Process monitor (*vtmon*) control	Maintain all per monitored process information for the following processes: vtmon trc_agent trcr xscontrol vtmgr> mons vtmgr> (all [mon] commands)	mons: Show list of all monitored processes. mon <***ID***>: Display all information for a monitored process. mon <***ID***> term: Terminate the process (triggering auto-recovery). mon <***ID***> stop: Terminate the process (do not automatically restart). mon <***ID***> start: Start up a process that was stopped. [mon] commands: Modify the configuration of a monitored process.
Controlling application	Display the controlling application connections. vtmgr> apps	apps: Displays a list of all currently connected control applications.
Channel monitoring and channel abort	Display the simplex or full-duplex channels that are created by controlling applications. vtmgr> chans	chan <***channel ID***>: Displays all information for a specific channel. chan <***channel ID***> abort: Terminates the

Task	Description	Commands
	Can also abort a channel. vtmgr> chan <ID> abort	channel immediately. cfg <***channel ID***>: Displays the configuration of a specific channel. status <***channel ID***>: Displays the current status of a specific channel. stats <***channel ID***>: Displays the current statistics of a specific channel. stats <***channel ID***> rtp: Limits channel statistics to RTP-related information. stats <***channel ID***> rtcp: Limits channel statistics to RTCP-related information. zero chan <***channel ID***>: Zeroes the current statistics for the channel.
View summary statistics	Summary-level statistics are collected and presented in several forms. The summary statistics totals can also be zeroed.	total currmin min <***minutes ago***> hhr <***half-hours ago***> last zero total

Management utility events

Use management utility events to alter a state. Events represent configuration changes that alter state. The following types of events can be performed:

Event	Description
vtp abort	Abort all channels and disable.
vtp disable	Disabled channel assignment.
vtp enable	Enable channel assignment.

Event	Description
chan <*ID*> abort	Terminate the channel immediately.
mon <*ID*> start	Start up a process that had been stopped.
mon <*ID*> stop	Terminate process (do not automatically restart).
mon <*ID*> term	Terminate the process (triggering auto-recovery).
vtp restart	Stop all processes and then restart.
vtp reboot	Cause the video transcoder platform to reboot.

Using vtmgr commands

The following table provides examples of how to use some the *vtmgr* commands:

Example	Enter this command...	Description
Performing a warm start of all transcoder processes	vtmgr> vtp restart	The trc_agent terminates all trcps and then exits. The *vtmon* process detects the trc_agent termination which triggers *vtmon* to terminate and then restart all other transcoder processes.
Rebooting the video transcoder platform	vtmgr> vtp reboot	Reboots the video transcoder platform and causes it to go through the entire reboot sequence.
Terminating a monitored process	vtmgr> mon 1 term	Use the mon<ID> term command to terminate a monitored process. In this example, the command terminates *vtmon.* Note: For most processes (including *vtmon*), terminating that process can result in *vtmon* terminating other processes as part of the recovery process.
Terminating a process so it is not	vtmgr> mon 2 stop <<*replace the trc_agent executable*>>	This is useful for applying patches. This example shows how to

Example	Enter this command...	Description
automatically restarted by *vtmon*	vtmgr> mon 2 start	apply a patched version of the trc_agent. Note: The only way to stop *vtmon* and not have it automatically restart is to remove *vtmon* from the *inittab* list and then terminate *vtmon*. Use the *monitorXC.sh* script to turn the process monitor mode to OFF before replacing the *vtmon* executable: monitorXC.sh off <<replace vtmon>> monitor.sh on
Aborting a channel	vtmgr> chan 0x30001 abort	Any channel can be aborted using the management interface. When a channel is aborted, the controlling application is given a notification indicating that the channel was aborted by management.

Video transcoder platform-level configuration

All video transcoder platform-level configuration settings can be viewed using the vtp command and can be modified using any of the commands listed as part of the [vtp] command set. The following table describes the video transcoder platform-level configurable fields:

Command	Description
apiTime	Set time for TRC API response [in milliseconds] (0 = infinite).
appLostTime	Set time (after disconnect) when the application lost [milliseconds] (0 = infinite).
debugLogMask	Set trc_agent debug log mask. For more information, refer to *Using log files* on page 319.
decodePartials	Set whether partial frames should be passed to the decoder.
initTime	Set time for INIT REQ from TRC API [in milliseconds] (0 = infinite).

Command	Description
licenseHigh	Set percent of licenses in use when VTP_LICENSE notified (high water).
licenseLow	Set percent of licenses in use when VTP_LICENSE notified (low water).
logToConsole	Set whether to-file logging should be forked to console.
maxRtpPayld	Set the maximum size of any outbound RTP packet payload.
mediaAddress	Set IP address for media endpoint access to video transcoder platform [@ = use ctl].
overlayExcl	Set whether video transcoder platform is reserved for channels requiring overlays.
quit	Quit (exit) the application.
rtcpInTime	Set default RTCP idle (no RTP, RTCP RX) input endpoint timeout.
rtcpMode	Set whether video transcoder platform acts as an RTCP translator by default.
rtcpOutTime	Set default RTCP idle (no RTCP RX) output endpoint timeout.
spxMaxChans	Set maximum number of simplex channels to allow (must be even).
trapAddress	Set IP address that all asynchronous traps are issued to [@ = no traps].
trapMask	Set mask of events the video transcoder platform will issue traps for (VTMNG_EVENT_*xxx*).
trapPort	Set UDP port number that asynchronous traps are issued to.
trcpCount	Set number of TRCPs to create [+ = count; - = spx_per_trcp].
trcpLogMask	Set trcp debug log mask. For more information, refer to *Using log files* on page 319.
usageHigh	Set percent of estimated usage when VTP_USAGE notified (high water).
usageLow	Set percent of estimated usage when VTP_USAGE notified (low water).
vtpDesc	Set the video transcoder platform description (for example, version)[@ = trc_agent - generated].
vtpInitState	Set video transcoder platform initialization state (whether video transcoder platform will initialize to a disabled state).

Command	Description
vtpName	Set the name of this video transcoder platform instance [@ = use hostname].

Monitored process configuration

All monitored processes can be independently configured using the commands in the [mon] command set. The following table describes the per-monitored-process configurable fields:

Command	Description
monName *<ID>* *<procName>*	Set the name of this monitored process.
monInitState *<ID>* *<state>*	Set the initial state of the monitored process: create = create process and then monitor the process. locate = locate the process (not created by process monitor). skip = skip the process (not created and not monitored).
exec *<ID>* *<executable>*	Set the name of the executable file for this process.
cmdLine *<ID>* *<cmdString>*	Set the command-line options provided on process creation.
var *<ID>* *<num>* *<name>* *<string>*	Set the name and string value of one of eight per-monitored process environment variables.