Dialogic®

Dialogic® Software Video Transcoder – Release Guide

Release 3.0

January 2011 64-1136-01 – Rev A

www.dialogic.com

Copyright and Legal Notice

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. **Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.**

Dialogic, Dialogic Pro, Dialogic Blue, Veraz, Brooktrout, Diva, Diva ISDN, Making Innovation Thrive, Video is the New Voice, Diastar, Cantata, TruFax, SwitchKit, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, TrustedVideo, Exnet, EXS, Connecting to Growth, Fusion, Vision, PowerMedia, PacketMedia, BorderNet, inCloud9, I-Gate, Hi-Gate, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Inc. and its affiliates or subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 926 Rock Avenue, San Jose, California 95131 USA. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Table Of Contents

Revision History

Revision	Release date	Notes
GA	September 2010	
GA Update	January 2011	Added note for VTP Server OS Support
Last modified: January 6, 2011		

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

1. Introduction

This Release Guide describes the enhancements and API updates for the Dialogic® Software Video Trancoder Release 3.0 compared to Release 2.1. The Dialogic® Software Video Trancoder is referred to herein as the SVT.

For a full description of features and API's for SVT, refer to the *Dialogic® Software Video Trancoder Reference Manual*. This reference manual has not been updated for Release 3.0. All changes since Release 2.1 are included in this Release Guide.

All API's in Release 3.0 (excluding those that are new in this document) remain backwards compatible with Release 2.1.

For installation information, please refer to *Installing Dialogic® Video Transcoder.*

2. Software Video Transcoding - Release 3.0 Enhancements

The following are the main enhancements for Software Video Trancoder (SVT) Release 3.0:

- VTP Server now supports 32-Bit Red Hat Linux Enterprise Solutions 4 operating system (as well as 64-bit supported previously)
- Addition of support for H.264 video codec
- Higher density/improved video quality
- Enhanced configuration and management:
  - CPU usage management
  - Enhanced logging capabilities
  - Dynamic VTP configuration
- Other (MONA, etc.)

VTP Server OS Support

SVT Release 3.0 now supports the Red Hat Linux Enterprise Solutions 4, 32-bit OS, where previously it only supported 64-bit. The main benefit is that an application using 32-bit RHES4, can now perform video transcoding using the SVT Release 3.0 VTP server on the same platform. Care must be taken, however, in terms of expected density when running video transcoding on the same server.

H.264 Video Codec

SVT Release 3.0 supports transcoding to and from H.264 format in addition to MPEG-4 and H.263. H.264 transcoding is supported by the following standards:

- ITU-T Recommendation H.264, Baseline Profile Levels 1 to 1.3, 2 to 2.2, 3, and 3GPP specifications TS.26.111, TS.26.911, TS.26.140.
- RFC3984, single-NAL and non-interleaved packetization modes

This release supports the following rates/characteristics for H.264:

- Frame rates: 4 to 30 fps
- Video encoding bit rates: 25kbit/s to 2000 kbit/s
- Picture sizes: QCIF and CIF

The following definition and structure (in trcdefs.h) have been updated as highlighted to support H.264.

```
#define TRC_MAX_DATA_RATE   2000     /* maximum supported data rate (in kilobits/sec) */

/* ----- complete endpoint configuration ----- */
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 */
#define TRC_VIDTYPE_H264        4   /* H.264 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 */
#define TRC_H264_LEVEL_1_0      8   /* H.264 profile level 1 */
#define TRC_H264_LEVEL_1_1      9   /* for H.264 profile level 1.1 */
#define TRC_H264_LEVEL_1b       10  /* for H.264 profile level 1b */
#define TRC_H264_LEVEL_1_2      11  /* for H.264 profile level 1.2 */
#define TRC_H264_LEVEL_1_3      12  /* for H.264 profile level 1.3 */
#define TRC_H264_LEVEL_2_0      13  /* for H.264 profile level 2.0 */
#define TRC_H264_LEVEL_2_1      14  /* for H.264 profile level 2.1 */
#define TRC_H264_LEVEL_2_2      15  /* for H.264 profile level 2.2 */
#define TRC_H264_LEVEL_3_0      16  /* for H.264 profile level 3.0 */
#define TRC_H263_LEVEL_45       17  /* H.263 profile level 45 */
```

```
    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 */


#define TRC_PACKETIZE_3984_SINGLE_NAL_UNIT   4  /* endpoint uses Single NAL Unit mode for
packetizing H.264 as defined in RFC 3984*/
```

```
#define TRC_PACKETIZE_3984_NON_INTERLEAVED   5  /* endpoint uses Non-Interleaved mode for
packetizing H.264 as defined in RFC 3984*/


    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;
```

Improved Density/Quality

SVT Release 2.1 was previously qualified on the following platform with the density as noted:

- Dual 3.4 GHz processor HP Proliant DL380 G4 chassis
- 60 simplex transcoder channels

Based on the improvements to the coders in SVT, there is a density improvement of up to 40 percent when comparing SVT Release 3.0 to SVT Release 2.1 on the same platform (MPEG-4 and H.263 only).

SVT Release 3.0 supports densities of up to 480 simplex channels on a dual-E5540 system, depending on the codec configuration/picture size/rates/bitstream complexity. Many combinations will yield substantially lower densities, and some could actually yield higher densities, but have not been tested above 480 simplex channels.

Enhancements to the video coders have been made in SVT Release 3.0 to also improve video quality when compared to SVT Release 2.1.

Enhanced Configuration and Management

As noted above, SVT Release 3.0 offers enhanced capabilities for the following areas:

- CPU usage management
- Logging
- Dynamic VTP configuration

The following VTP configuration structure (in transmanage.h) has been updated as shown for some of the enhanced management capabilities. The values in this structure are set in vtp.cfg and can be modified using VTMNG (same as for SVT Release 2.1).

```
/*-------------------------------------------------------------------------*
 * VTP Configuration (Get|Set|Zero):
 *-------------------------------------------------------------------------*/
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_NOTIF_PORT]) */
    U8                  codecValidationLevel;       /* type of validation to
perform on incoming bitstreams */
```

```
    U32                                codecDebugMask;                                /* bitmask of
debugging levels at the codec level */
    S32                SystemLogFileMaxSize;       /* maximum size of log files in KB
                                                      (-1 = no limit, file will grow infinitely)
*/
    S32                SystemLogFileMaxNum;        /* maximum # of rollovers per log file
                                                      (-1 = no limit on # of rollovers (no
rollovers will be deleted),
                                                        0 = no rollovers maintained (only active
files exist)) */
       U8                  CPUCalcSamples;             /*number of samples to calculate average
CPU usage*/
       U32                 rejectHighWater;            /*CPU usage level to start rejecting
calls */
       U32                 rejectLowWater;             /*CPU usage level to start re-accepting
calls once rejectHighWater limit was reached */
    U8                 displayTimeSrc;             /* source for timestamps [0 = rtp timestamp;
1 = temporal reference normalized to 90KHz; 2 = srt]  */
    U8                 reserved[16];               /* reserved for future use */
} VTMNG_VTP_CFG;
```

The last parameter in the structure, displayTimeSrc, is added to allow flexibility in the timing source for use with encoder timing. The default is RTP (value 0), which is what was used for SVT Release 2.1. Other possible settings are to use temporal reference values (bitstream timing) or internal system reference time.

The first two new values in the structure are used for new codec-level debugging. Definitions that can be used for setting these fields are shows below, and can be found in trcdefs.h.

```
/* ----- Levels for codec validation ----- */
#define TRC_CODEC_VALIDATION_NONE            0
#define TRC_CODEC_VALIDATION_HEADER          1
#define TRC_CODEC_VALIDATION_FULL            2

#define TRC_CODEC_VALIDATION_LEVEL_DEFAULT          TRC_CODEC_VALIDATION_HEADER

/* ----- Levels for IPP codec debugging ----- */
/* These are used to translate from values specified in vtp.cfg to
   the values used by the IPPs

       #     0x00000001 = trace any detected error
       #     0x00000002 = trace warning indications
       #     0x00000004 = trace upper layer (application) interface
       #     0x00000008 = write bitstream
       #     0x00000010 = write rate control information
       #     0x00000020 = write frame level information
       #     0x00000040 = write macroblock level information
       #     0x00000080 = write block level information
*/
```

```
#define TRC_IPPLOG_NONE  0
#define TRC_IPPLOG_ERR   (1 << 0)
#define TRC_IPPLOG_WRN   (1 << 1)
#define TRC_IPPLOG_API   (1 << 2)
#define TRC_IPPLOG_BSI   (1 << 3)
#define TRC_IPPLOG_RCI   (1 << 4)
#define TRC_IPPLOG_FLI   (1 << 5)
#define TRC_IPPLOG_MBI   (1 << 6)
#define TRC_IPPLOG_BLI   (1 << 7)

#define TRC_IPPLOG_DEFAULT           TRC_IPPLOG_NONE
```

For normal behavior/debugging, set codecValidationLevel to 1, and set codecDebugMask to 0.

CPU Usage Management

The following new fields have been added to the VTMNG_VTP_CFG structure as noted above

- CPUCalcSamples
- rejectHighWater
- rejectLowWater

The older CPU usage fields in the structure, usageHighWater and usageLowWater, are still used in the same manner to control notifications of CPU alarm levels. The new "reject" values are used to determine at what levels call rejection takes place or not.

The field rejectHighWater is used to determine at what CPU usage level the VTP should stop accepting calls. The range is 0 to 100, and the default value is 100 to mimic SVT Release 3.0 behavior (which did not reject calls). The value should be set greater than or equal to the value of usageHighWater.

The field rejectLowWater is used to determine at what CPU usage to begin accepting calls after rejectHighWater had previously been reached. The range is 0 to 100 and the default value is 90. Be careful in setting this value much less that rejectHighWater or it could take a long time to begin accepting calls again.

The value CPUCalcSamples, with a range of 1-20 and a default of 5, sets the number of internal CPU usage samples that are used to determine the average CPU utilization calculation. Internally, CPU usage samples are taken approximately every 200 milliseconds, thus a value of 5 will yield a period of 1 second for every new average CPU utilization calculation.

Logging

Modifications to logging include the following:

- trcp and trc_agent logs (xc.log_* and xc.log will be renamed to be more intuitive (trcp.*.log and trc_agent.*.log)
- Rollover index numbers and timestamps have been added to log filenames
- New xcf/xpf logs

- Ability to specify maximum log file size and maximum number of rollovers

This last feature is controlled by the following parameters in the VTMNG_VTP_CFG as shown above.

SystemLogFileMaxSize: specifies maximum size of logfiles in KB. A value of -1 sets no maximum size to log files and they will grow indefinitely as they did on SVT Release 2.1. The range of normal values is 1 to 512000 (1KB to 500MB). The default value is 10240 (10 MB).

SystemLogFileMaxNum: specifies the maximum number of rollovers per log file. A value of -1 specifies no limit on the number of rollovers. A value of 0 specifies no rollovers are maintained (as with SVT Release 2.1). Normal range is 1 to 500. The default is 5.

Most logfile names use one of the following formats:

- process.xxxx.iii.yyyymmdd-hhmmss.log
- process.iii.yyyymmdd-hhmmss.log

Where:

- xxxx = instance or channel number
- iii = index number that increments with each log rollover for the give process or channel.
- yyyymmdd-hhmmss = creation time

XCF/XPF logs are for transcoder control function and transcoder packetizer function logging. Old media stream data dumps are replaced by XCF logs.

Dynamic VTP Configuration

An SVT Release 2.1 application cannot add or remove a VTP without restarting - calling trcShutdown() and initializing again with trcInitialize(). On SVT Release 3.0, dynamic VTP configuration will be allowed – using the following new function and structure.

```
/* Update TRC module - adding or removing connections to VTPs*/
U32 trcUpdateVTPConfig(tTrcUpdateInfo *trcUpdateInfo);


typedef struct
{
    void           *reserved;           /* reserved for future use */
} tTrcUpdateInfo;
```

The function trcupdateVTPConfig() reads the same config file that is specified in trcinitialize(). If that config file has been updated with new VTP's in the list of has had VTP's removed from the list, then the TRC will either add new connections to those new VTP's or remove its connections to the VTP's that are no longer in the list. Any active channels on currently used VTP's will be destroyed.

Additional Changes

The following are the other minor changes in SVT Release 3.0.

MONA Support

A MONA field has been added to the structure shown below (in file trcdefs.h). This field is used to optimize the video encoder in support of MONA and ensure the proper MONA DCI.

```
/* ----- endpoint info specific to transcoder transmitting to endpoint ----- */
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]
*/
        tTrcCfgValue    monaSupport;        /* the encoder output is to be optimized for MONA 3G
endpoint
                                                                                * monaSupport.isSet =
TRC_TRUE implies the encoder output is to be optimized for MONA 3G endpoint
                                                                                * monaSupport.isSet =
TRC_FALSE implies the encoder output is not optimized for MONA 3G endpoint */
    U8              reserved[24];       /* reserved for future use */
} tTrcEndOutput;
```

Data Communications Statistics

A frame count field has been added as shown to the following structure in VTMNG (in file transmanage.h).

```
/*-------------------------------------------------------------------------*
 * Statistics Sub-Structure: Data Communication Related stats
 *-------------------------------------------------------------------------*/
typedef struct __vtMng_St_Comm
{
    U32             frames;             /* frame count */
    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;
```

TRC Logging

The TRC client relies on the application to implement its own file size limits and file rotations by allowing the application to specify a callback function to be called by TRC every time there is something to be logged in the TRC client. A new API is added in TRC to allow the application to specify the callback function.

```
/*--------------------------------------------------------------------------*
 * Application defined logging function prototype
 *
 * This function is provided by the calling application as a method of
 * servicing all logging initiated by the TRC.
 *
 * INPUTS: logline - text to be logged
 *               filehandleindex - RESERVED FOR FUTURE USE
 *
 * OUTPUT: reserved - always returns 0 [RESERVED FOR FUTURE USE]
 *--------------------------------------------------------------------------*/
typedef U32 (* tTrcLogAppFunc) (S8 *logline, U32 filehandleindex);



/* Specifies a user-defined callback function for logging */
U32 trcSetLogFunction(tTrcLogAppFunc pLogFunc);
```

The trcSetLogFunction() API will set the callback function for all TRC logging. It must be called before trcInitialize() for it to take effect.

Once trcSetLogFunction() is called, the logFileName parameter in trcInitialize() becomes irrelevant since the application will be expected to have its own log file where the TRC logs will go.

The filehandleindex variable in the callback function signature has been included in the API signature for flexibility.

Sample VTP Configuration File

```
###############################################################################
#
# 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 can 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
#-------------------------------------------------------------------------------
# trapPort: set UDP port number that async traps are issued to
#   SPECIAL VALUE: 0 = use default port number [VT_MANAGE_NOTIF_PORT]
#-------------------------------------------------------------------------------
# SystemLogFileMaxSize: maximum size of log files
#   UNITS: KB
#   VALID RANGE:   1-512000 (i.e. 1KB - 500MB)
#   SPECIAL VALUE: -1 = no filesize limit, log file will grow infinitely
#-------------------------------------------------------------------------------
```

```
# SystemLogFileMaxNum: maximum # of rollovers maintained for each log file
#   VALID RANGE:   0-500
#   SPECIAL VALUE: -1 = no limit on # of rollovers (i.e. no rollovers will be deleted)
#   SPECIAL VALUE: 0  = no rollovers will be maintained (only active log files will exist)
#-------------------------------------------------------------------------------
# CPUCalcSamples: set rolling window average  size
#   VALID RANGE:   1-20
#   SPECIAL VALUE: 1 = Instantaneous value of CPU usage
#-------------------------------------------------------------------------------
# rejectHighWater: percentage of estimated usage beyond which reject calls
#   VALID RANGE: usageHighWater-100
#-------------------------------------------------------------------------------
# rejectLowWater: percentage of estimated usage after hitting reject high water#
to start accepting calls
#   VALID RANGE: 0-100 (must be <= rejectHighWater and >= usageLowWater)
#-------------------------------------------------------------------------------
# codecValidationLevel: type of validation to perform on incoming bitstreams
#   VALID RANGE:   0-2
#   SPECIAL VALUE: 0 = No validation
#   SPECIAL VALUE: 1 = Codec Header validation only (default value)
#   SPECIAL VALUE: 2 = Codec Header and Bitstream validation
#-------------------------------------------------------------------------------
# codecDebugMask: bitmask of debugging levels at the codec level
#   Logfiles will be named as follows (example for h264 encoder):
#      h264.enc.xxxx.iii.yyyymmdd-hhmmss.log
#      h264.enc.xxxx.iii.yyyymmdd-hhmmss.264
#        xxxx: channel number
#        iii: sequence # that increments for each backup log of a given encoder/decoder,
#        yyyymmdd-hhmmss: creation time (year, month, day, hour, minute, second) of the log file)
#
#   VALID RANGE: A mask consisting of any number of values below ORed together
#   SPECIAL VALUE: 0 = NO CODEC DEBUG (this is the default)
#
#     0x00000001 = trace any detected error
#     0x00000002 = trace warning indications
#     0x00000004 = trace upper layer (application) interface
#     0x00000008 = write bitstream
#     0x00000010 = write rate control information
#     0x00000020 = write frame level information
#     0x00000040 = write macroblock level information
#     0x00000080 = write block level information
#-------------------------------------------------------------------------------
# displayTimeSrc: decoder display time source
#   VALID RANGE:   0-2
#   SPECIAL VALUE: 0 = rtp timestamp
#   SPECIAL VALUE: 1 = temporal reference normalized to 90KHz
```

```
#   SPECIAL VALUE: 2 = srt
#-------------------------------------------------------------------------------
################################################################################

#===============================================================================
# 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        = 1460
apiTimeout           = 5000
initTimeout          = 5000
appLostTimeout       = 300000
debugLogMask         = 0x00000003
trcpLogMask          = 0x00000003
logToConsole         = DISABLED
rtcpInTimeout        = 0
rtcpOutTimeout       = 0
trapMask             = 0x03100007
trapAddress          = @
trapPort             = 0
SystemLogFileMaxSize = 10240
SystemLogFileMaxNum  = 5
CPUCalcSamples       = 5
rejectHighWater      = 100
rejectLowWater       = 90
codecValidationLevel = 1
codecDebugMask       = 0x00000000
displayTimeSrc       = 0
#===============================================
```