GLOBAL IP SOLUTIONS

See the Difference. Hear the Difference.

GIPS VideoEngine

API Guide

Headquarters
Global IP Solutions, Inc.
642 Brannan Street, Second Floor
San Francisco, CA 94107
USA

Phone: +1 415 397 2555
Fax: +1 415 397 2577

For additional contact information, please visit the GIPS URL: http://www.gipscorp.com

For information on how to contact Customer Support, please visit the following URL: https://www.gipssolutions.com/customer/login.asp

ViE.API.03.05.2010

Contents

Chapter 1: About This Guide

This document describes the interface of the GIPS VideoEngine, which is delivered as a C++ library or as a DLL.

The GIPS VideoEngine has the following main functionalities:

1. Video Processing
2. RTP Protocol
3. Synchronization with GIPS VoiceEngine

The GIPS VideoEngine API enables a correct call-setup by allowing the user to check what capabilities that is available and to set call-setup information.

NOTE: GIPS VideoEngine does ***not*** handle call setup.

When using GIPS VideoEngine together with GIPS VoiceEngine, audio synchronization is automatically performed.

Product Version

This GIPS VideoEngine API description corresponds to GIPS VideoEngine product version 2.5.2.

In This Guide

This API guide gives a thorough description of GIPS VideoEngine

- `Chapter 2, Compilation,` covers compilation for all platforms.
- `Chapter 3, API Reference,` is the API reference for all classes, structures and functions.
- `Chapter 4, Example code,` gives examples of typical usage.
- `Chapter 5, Error Handling,` describes error handling.
- `Appendix A, Error Codes,` describes error codes.

Document Change History

The following Document Change History table records the technical changes to this document, giving the API version number, revision date, and a summary of the change.

API Version	Date	Change Summary
1.0	Oct 2008	Document migration and revision to new format.
2.5.0	Nov 2009	Added GIPSVideo_EnableRemoteResize Added GIPSVideo_SetCaptureDelay Added GIPSVideo_GetFileInfo Added GIPSVideo_EnableFrameUpScale Added GIPSVideo_SetMaxPacketBurstSize Added GIPSVideo_EnableFEC Added GIPSVideo_EnableKeyFrameRequestCallback Removed GIPSVideo_EnableMissingMarkerBitSupport Updated GIPSVideo_IncomingCapturedFrame Updated GIPSVideo_EnableRTCP Updated GIPSVideo_EnableSRTPSend Updated GIPSVideo_EnableSRTPReceive Updated GIPSVideo_EnableEncryption Added GIPSVideo_StartRTPDump Added GIPSVideo_StopRTPDump Added GIPSVideo_RTPDumpIsActive Added GIPSVideo_SetThresholdToSignalRemoteResize Added GIPSVideo_EnableDirect3D Added GIPSVideo_AddRemoteRenderer for HIViewRef Added GIPSVideo_AddLocalRenderer for HIViewRef Added GIPSVideo_AddLocalRenderer for Cocoa Added GIPSVideo_AddRemoteRenderer for Cocoa
2.5.2	Feb 2010	Updated GIPSVideo_SetSendToS Updated GIPSVideo_GetSendToS Updated GIPSVideo_SetSendGQOS Updated GIPSVideo_GetSendGQOS

GLOBAL IP SOLUTIONS

Writing Conventions

This guide uses the following writing conventions:

Convention	Definition
`Code`	Indicates a parameter to which the description is referring.
`Syntax`	Gives the syntax or usage of a function call.
URL	Indicates a jump to an external information source, such as a Web site or a URL.
`Document Link`	Indicates a jump to a section of this document with more information.
NOTE:	Indicates important information that helps to avoid and troubleshoot problems

Chapter 2: Compilation

Windows

To link to GIPS VideoEngine, GIPS recommends Visual Studio 2003 or Visual Studio 2005. Default Windows deliveries are made with Visual Studio 2005 and can be made using Microsoft Visual Studio 2003 upon request. Some components within GIPS VideoEngine are compiled with the Intel Compiler to provide maximum performance; the following Intel library dependencies may need to be ignored when linking:

- libmmt
- libircmt
- libirc
- libguide40

To ignore these add **/nodefaultlib:LIBRARY_NAME** under **Linker->Command Line->Additional options** in Microsoft Visual Studio.

GIPS VideoEngine requires the following Microsoft Windows SDK libraries:

- quartz.lib
- winmm.lib
- Vfw32.lib, WS2_32.lib (*not required for external transport*)
- strmiids.lib (*only required for DirectShow rendering*)
- ddraw.lib (*only required for DirectDraw rendering*)
- dxguid.lib

Mac OS X

GIPS VideoEngine is built using GCC and will be delivered for Intel processor. GIPS VideoEngine for Mac OS X uses the QuickTime, Carbon, and Cocoa frameworks. These three frameworks are required when building the final application together with the CoreAudio framework and the AudioToolbox framework used by GIPS VoiceEngine. If the VideoEngine library is built with OpenGL support, the AGL and the OpenGL frameworks are also required.

Additionally, Cocoa applications need to include the file GIPSCocoaRenderer.h. This file is a subclass of NSView. Instances of the control should be created at runtime. Since the implementation file is not provided, Interface Builder won't know what GIPSCocoaRenderer is, and will not be able to initialize it properly.

Linux

GIPS VideoEngine for Linux is built using GCC and is delivered as a static Linux library compiled for Intel architecture. It uses the V4L1 and V4L2 (Video for Linux 2) interface to communicate with web cameras supported by this interface. This interface is generally part of the Linux 2.6 core in common Linux distributions and does not need to be linked in. The video is rendered on the screen using the Xlib library, which needs to be linked into the application executable.

NOTE: The Linux system needs to include the X Window System (X11) to make rendering possible.

Windows Mobile

GIPS VideoEngine Mobile is built in Visual Studio 2005 using the Windows Mobile 5.0 Pocket PC SDK and is delivered as a static library. The GIPS VideoEngine library can be used for application development in Visual Studio 2005 and Visual Studio 2008 using the Windows Mobile 5, 6 or 6.1 SDK.

Chapter 3: API Reference

This chapter describes the GIPS VideoEngine API. Sections named "Optional Settings" contain function calls that do not have to be used in a standard call but might be useful in advanced call scenarios. For customers that want to use another transport protocol than the default one, RTP/UDP/IP, the `Network Settings-External Transport Protocol` section is very important. All other users can ignore that section.

Functions marked with **Optional Feature** indicate that this call relies on a GIPS product that might not be included in your GIPS VideoEngine configuration.

All member functions are described in terms of syntax, return values, remarks, code examples and requirements.

Structures and Types

The GIPS VideoEngine holds information about each codec it supports in this format.

GIPSVideo_CodecInst

Syntax

```
struct GIPSVideo_CodecInst
{
   unsigned char pltype;
   char plname[32];
   int bitRate;
   int maxBitRate;
   unsigned char frameRate;
   unsigned short height;
   unsigned short width;
   char quality;
   char level;
   char codecSpecific;
   unsigned char configParameterSize;
```

```
  unsigned char configParameters[128];
  int minBitRate;
  char profile;
  char complexity;
  unsigned char dependency[8];
  GIPSVideo_LayerRange layerRange;
  GIPSVideo_Layers layers;
};
```

Parameters

pltype	RTP payload type.
plname	MIME name.
bitRate	Start bit rate (kbps).
maxBitRate	Maximum bit rate (kbps).
frameRate	Maximum frames per second for codec.
height	Height of the picture.
width	Width of the picture.
quality	Codec specific value*.
level	Codec specific value*.
codecSpecific	Codec specific value*.
configParameterSize	Number of valid char in configParameters*.
configParameters	Out of bound signal settings to the codec*.
minBitRate	Minimum bit rate (kbps).
profile	For future use.
complexity	For future use.
dependency	For future use.
layerRange	For future use.
layers	Codec layer information.

Remarks

See the Codec Settings sections for specific information for each codec. `ConfigParameterSize` can be used for h264 and MPEG-4 codecs and should be set to 0 for all other codecs. `ConfigParameterSize` cannot take a value more than 128.

Example Code

The code below shows how to setup LSVX with payload type 97 in CIF quality mode.

```
GIPSVideo_CodecInst codec;

strncpy(codec.plname, "LSVX", 5);
codec.pltype = 97;
codec.level = 5;
codec.quality = GIPS_QUALITY_DEFAULT;
codec.bitRate = 200;
codec.maxBitRate = 200;
codec.width = 352;
codec.height = 288;
codec.frameRate = 30;
codec.configParameterSize = 0;
codec.codecSpecific = GIPS_LSVX_KEY;
```

GIPSVideoType

Uncompressed video exists in various formats. The following enum contains the GIPS recognized formats. Note that the GIPS engine doesn't support all these formats on all platforms.

Syntax

```
enum GIPSVideoType

{

  GIPS_UNKNOWN = 0,

  GIPS_I420 = 1,

  GIPS_IYUV = 2,

  GIPS_RGB24 = 3,

  GIPS_ARGB = 4,

  GIPS_ARGB4444 = 5,

  GIPS_RGB565 = 6,

  GIPS_ARGB1555 = 7,

  GIPS_YUY2 = 8,

  GIPS_YV12 = 9,

  GIPS_UYVY = 10,
  GIPS_V210 = 11,
  GIPS_HDYC = 12,
  GIPS_MJPG = 13,
  GIPS_H263 = 14,
  GIPS_H264 = 15
}
```

GIPSCameraCapability

The GIPS VideoEngine can enquire the capture device (camera) capabilities. Use the function `GIPSVideo_GetCaptureCapabilities()` to enumerate the capture capabilities.

Syntax

```
struct GIPSCameraCapability
{
  int width;
  int height;
  int maxFPS;
  GIPSVideoType type;
}
```

Parameters

width	[out] Width in pixels supported.
height	[out] Height in pixel supports.
maxFPS	[out] Max frame per second supported.
type	[out] Type of uncompressed video supported.

GIPSVideo_CallStatistics

The structure is used to access statistics from RTCP reports through `GIPSVideo_RTCPStat().` The statistics are computed according to RFC 3550 under Sender and Receiver Reports. Refer to RFC for more information.

Syntax

```
struct GIPSVideo_CallStatistics
{
  unsigned short fraction_lost;
  unsigned long cum_lost;
  unsigned long ext_max;
  unsigned long jitter;
  int RTT;
  int bytesSent;
  int packetsSent;
  int bytesReceived;
  int packetsReceived;
};
```

Parameters

fraction_lost	[out] Fraction of lost packets.
cum_lost	[out] Cumulative number of lost packets**.**
ext_max	[out] Max sequence number received including number of times the sequence number has wrapped.
jitter	[out] Jitter in samples.
RTT	[out] Roundtrip time in milliseconds.
bytesSent	[out] Total number of bytes sent**.**
packetsSent	[out] Total number of packets sent**.**
bytesReceived	[out] Total number of bytes received.
packersReceived	[out] Total number of packets received.

GIPSVideo_FrameStatistics

`GIPSVideo_FrameStatistics` contains information about the current encoder and decoder for a given channel.

Syntax

```
struct GIPSVideo_FrameStatistics
{
  unsigned int sentKeyFrames;
  unsigned int sentDeltaFrames;
  unsigned int receivedKeyFrames;
  unsigned int receivedDeltaFrames;
```

```
    unsigned int zeroEncodeFrames;
    unsigned int zeroDecodeFrames;
    unsigned int errorDecodeFrames;
};
```

Parameters

sentKeyFrames	[out] Number of key frames sent.
sentDeltaFrames	[out] Number of delta frames sent.
receivedKeyFrames	[out] Number of received key frames.
receivedDeltaFrames	[out] Number of received delta frames.
zeroEncodeFrames	[out] Number of encodes resulting in no encoded frame to send.
zeroDecodeFrames	[out] Number of decodes resulting in no decoded frame to render.
errorDecodeFrames	[out] Number of decodes resulting in error.

GIPSVideo_Picture

`GIPSVideo_Picture` is used to store a video frame in memory.

Syntax

```
struct GIPSVideo_Picture
{
    unsigned char*    data;
    unsigned int      size;
    unsigned int      width;
    unsigned int      height;
    GIPSVideoType     type;
};
```

Parameters

data	[in/out] Picture data.
size	[in/out] Number of data bytes.
width	[in/out] Picture width, in pixels.
height	[in/out] Picture height, in pixels.
type	[in/out] Picture video format.

Enumerator GIPS_TraceFilter

This enumerator specifies what type of trace filter to use.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
namespace GIPS
{
  enum TraceLevel
  {
    TR_NONE          = 0x0000,
    TR_STATE_INFO    = 0x0001,
    TR_WARNING       = 0x0002,
    TR_ERROR         = 0x0004,
    TR_CRITICAL      = 0x0008,
    TR_APICALL       = 0x0010,
    TR_MODULE_CALL   = 0x0020,
    TR_DEFAULT       = 0x00FF,
    TR_MEMORY        = 0x0100,
    TR_TIMER         = 0x0200,
    TR_STREAM        = 0x0400,

    // everything bellow will be encrypted and is used for GIPS debug
purposes
    TR_DEBUG         = 0x0800,
    TR_INFO          = 0x1000,
    TR_CUSTOMER      = 0x2000,
    TR_ALL           = 0xFFFF
  };
};
```

Enumerators

TR_NONE	Disables all trace messages.

TR_STATE_INFO	Used for status messages, such as “incoming bit rate is 100 kbps”, or “function X is now active”.
TR_WARNING	Used for warning messages, such as “CPU load is too high”, or “function is already active”.
TR_ERROR	Used for error messages, such as “invalid parameter”, or “unable to open file”.
TR_CRITICAL	Used for critical messages, such as “soundcard failed to play out data”.
TR_APICALL	Used for all GIPS API calls, such as “GIPSVideo_Init()”.
TR_MODULE_CALL	Used for GIPS internal module calls; will lead to a very large amount of trace messages.
TR_DEFAULT	Used for default, non encrypted messages.
TR_MEMORY	Internal GIPS debug information.
TR_TIMER	Internal GIPS debug information; can lead to large amount of trace messages.
TR_STREAM	Internal GIPS debug information; will lead to a very large amount of trace messages.
TR_DEBUG	[encrypted] Internal GIPS debug information; can lead to large amount of trace messages.
TR_INFO	[encrypted] Internal GIPS debug information; can lead to large amount of trace messages.
TR_CUSTOMER	[encrypted] Internal GIPS debug information.
TR_ALL	Enables all trace messages.

Remarks

It is possible to combine several different values into one single filter using the OR (|) operator.
Example: `GIPS::TR_STATE_INFO| GIPS::TR_WARNING |GIPS:: TR_ERROR | GIPS::TR_CRITICAL`.

Declared in the GIPS_common_types.h header file.

Note that these enumerators are within in GIPS namespace.

Example Code

The below code shows the usage of different filters within SetTraceFilter api.

```
GipsVideoEngineWindows* _videoEngine;

_videoEngine->GIPSVideo_SetTraceFilter(TR_ERROR|TR_WARNING|TR_CRITICAL);
```

Initialization

GetGipsVideoEngine

Use this function to return a static instance of the VideoEngine. This is the preferred method of getting one instance since it always exists.

Syntax

```
GipsVideoEngineWindows &GetGipsVideoEngine()
GipsVideoEngineWindowsCe &GetGipsVideoEngine()
GipsVideoEngineLinux &GetGipsVideoEngine()
GipsVideoEngineMac &GetGipsVideoEngine()
```

Return Values

The function returns a reference to a static instance of VideoEngine.

Requirements

Supported platforms	Windows (incl. Mobile), Linux, Mac
Header	Declared in GipsVideoEngineWindows.h, GipsVideoEngineWindowsCE.h, GipsVideoEngineLinux.h, GipsVideoEngineMac.h

GetNewVideoEngine

This function returns a new instance of the VideoEngine. Only use this function if more than one instance of the VideoEngine is needed.

Syntax

```
GipsVideoEngineWindows *GetNewVideoEngine()
GipsVideoEngineWindowsCe *GetNewVideoEngine()
GipsVideoEngineLinux *GetNewVideoEngine()
GipsVideoEngineMac *GetNewVideoEngine()
```

Return Values

The function returns a pointer to a new instance of VideoEngine.

Requirements

Supported platforms	Windows (incl. Mobile), Linux, Mac, iPhone
Header	Declared in GipsVideoEngineWindows.h, GipsVideoEngineWindowsCE.h, GipsVideoEngineLinux.h, GipsVideoEngineMac.h, GipsVideoEngineiPhone.h

DeleteVideoEngine

This function deletes an instance of VideoEngine. Note that the static instance of VideoEngine cannot be deleted.

Syntax

```
DeleteGipsVideoEngine(GipsVideoEngine *videoEngine)
```

Requirements

Supported platforms	Windows (incl. Mobile)
Header	Declared in GipsVideoEngineWindows.h, GipsVideoEngineWindowsCE.h

GIPSVideo_Authenticate

If VideoEngine is delivered as a DLL, the DLL needs to be unlocked before any call to the VideoEngine can be made. The purpose of this is to prevent unauthorized usage of the VideoEngine DLL. A password string is delivered together with the DLL and this string is used to unlock the DLL.

Syntax

```
int GIPSVideo_Authenticate(char *auth_string, int len)
```

Parameters

auth_string	Pointer to password string.
len	Length of password string.

NOTE: The password string MUST be embedded in the calling exe-file, and not stored in any resource-file or registry key.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngine.h

GIPSVideo_Init

This function initiates the VideoEngine instance. A VoiceEngine object must be provided to enable audio synchronization.
If you have received a VideoEngine library that is time limited, you need to provide the expiry information here. For example, if the library will expire Jan 21 2010, that means you should set month = 1, day = 21 and year = 2010. That way a time limited library will never be deployed by mistake.

Syntax

```
int GIPSVideo_Init(  GIPSVoiceEngine* obj,
                     int month = 0,
                     int day = 0,
                     int year = 0)
```

Parameters

obj	VoiceEngine object.
month	Month when the video engine expires.
day	Day when the video engine expires.
year	Year when the video engine expires.

Return Values

0 is returned if the initialization was successful and –1 otherwise.

Remarks

During initialization of VideoEngine non-fatal errors might occur. If a non-fatal error occurs `GIPSVideo_GetLastError` will return an error code that is non-zero.

Example Code

The below example will initialize VideoEngine on windows. It takes a NULL VoiceEngine object. No audio synchronization would be provided in this case.

```
GipsVideoEngineWindows* _videoEngine = &GetGipsVideoEngine();
_videoEngine->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetVoiceEngine

This function can be used to link a VoiceEngine instance to a VideoEngine instance if no VoiceEngine was given as input in `GIPSVideo_Init`.

Syntax

```
int GIPSVideo_SetVoiceEngine(GIPSVoiceEngine* ve)
```

Parameters

ve	VoiceEngine object.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_InitThreadContext

This function initiates the COM context of the calling thread. It's required if you want to use several threads to access the VideoEngine API. The funtion calls `CoCreateInitializeEx()` method to initialize the COM context of the calling thread. Only certain functions which access COM requires to call this when called from a different thread. Refer to MSDN for more information on `CoCreateInitializeEx()` method.

Syntax

```
int GIPSVideo_InitThreadContext()
```

Remarks

The following functions have this requirement:

```
GIPSVideo_Terminate
GIPSVideo_Run
GIPSVideo_Stop
GIPSVideo_SetSendCodec
GIPSVideo_GetCaptureCapabilities
GIPSVideo_GetCaptureCapabilities
GIPSVideo_GetCaptureDevice
GIPSVideo_GetCaptureDeviceId
GIPSVideo_SetCaptureCardProperties
GIPSVideo_SetCaptureDevice
GIPSVideo_SetCaptureDeviceId
GIPSVideo_ViewCaptureDialogBox
```

The following functions have this requirement if you use DirectShow rendering:

```
GIPSVideo_CreateChannel
GIPSVideo_DeleteChannel
GIPSVideo_EnableMixingRender
GIPSVideo_AddLocalRenderer
GIPSVideo_AddRemoteRenderer
GIPSVideo_SetCropping
GIPSVideo_ConferenceDemuxing
GIPSVideo_ChangeHWND
GIPSVideo_ConfigureMixer
GIPSVideo_OnPaint
GIPSVideo_OnDisplayMode
GIPSVideo_OnSize
GIPSVideo_GetAssociatedRenderFilter
GIPSVideo_GetSnapShot
```

NOTE: This is only required on Windows platforms.

Remarks

VideoEngine uninitialize the thread context using `CoUnitializeEx()` in `GIPSVideo_Terminate()` api. Refer to MSDN for more information on `CoUnInitializeEx()` method.

Requirements

Supported platforms	Windows (incl. Mobile)
Header	Declared in GipsVideoEngine.h

Termination

GIPSVideo_Terminate

This function deletes the instance of the GIPS VideoEngine created by `GIPSVideo_Init`.

Syntax

```
int GIPSVideo_Terminate()
```

Return Values

The function returns 0 if the VideoEngine was successfully deleted and –1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Channel functions

The GIPS VideoEngine supports several channels i.e. it can send, receive and play out several video streams.

GIPSVideo_GetNoOfChannels

Function to get the number of active video channels in use by GIPS VideoEngine.

Syntax

```
int GIPSVideo_GetNoOfChannels();
```

Return Values

The return value is the number of active video channels in use by GIPS VideoEngine.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_CreateChannel

This function creates a new video channel in GIPS VideoEngine.

Syntax

```
int GIPSVideo_CreateChannel(int audioChannel);
```

Parameters

audioChannel	ID returned by calling GIPS VoiceEngine `GIPSVE_CreateChannel`. Can be -1 if voice is not used or if the voice channel is created at a later time.

Return Values

The return value is the video channel id if successful, -1 is returned if an error occurred.

Remarks

If `audioChannel` is set to –1, then the corresponding GIPS VoiceEngine channel should be specified later with `GIPSVideo_SetAudioChannel` for audio synchronization. If GIPSVoiceEngine is not set in the GIPSVideoEngine either with `GIPSVideo_Init` or `GIPSVideo_SetVoiceEngine`, then this parameter must be set to -1.

Example Code

The below example will initialize VideoEngine on windows and creates a video channel.

```
GipsVideoEngineWindows* _videoEngine = &GetGipsVideoEngine();

_videoEngine->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);
int channel = _videoEngine->GIPSVideo_CreateChannel(-1);
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetAudioChannel

This function is used to specify the corresponding GIPS VoiceEngine channel for a GIPS VideoEngine channel that was created without specifying the audio channel.

Syntax

```
int GIPSVideo_SetAudioChannel(   int videoChannel,
                                 int audioChannel)
```

Parameters

videoChannel	ID returned by `GIPSVideo_CreateChannel`.
audioChannel	ID returned by calling GIPS VoiceEngine `GIPSVE_CreateChannel`.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_DeleteChannel

Function to free an existing video channel and all its resources.

Syntax

```
int GIPSVideo_DeleteChannel(int videoChannel);
```

Parameters

videoChannel	ID returned by `GIPSVideo_CreateChannel`.

Return Values

The return value is 0 if the channel was successfully deleted or –1 if an error occurred.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Codec settings

This section contains function calls necessary to perform the codec part of the call-setup.

GIPSVideo_GetNofCodecs

This function returns the number of supported codecs.

Syntax

```
int GIPSVideo_GetNofCodecs();
```

Return Values

The function returns the number of supported codecs or –1 if an error has occurred at initialization.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetCodec

This function iterates all available video codecs supported by the engine.

Syntax

```
int GIPSVideo_GetCodec( int listnr,
                        GIPSVideo_CodecInst *codec_inst)
```

Parameters

listnr	Requested codec in internal prioritized codec list (0=highest priority).
codec_inst	Pointer to the struct in which the returned codec information is copied.

Return Values

This function returns 0 if successfully and –1 if an error occurred.

Remarks

To get the length of the list use the function `GIPSVideo_GetNofCodecs()`.

Example Code

The below example will initialize VideoEngine on windows and gets the codec list.

```
GipsVideoEngineWindows* _ptrVie = &GetGipsVideoEngine();

int res = _ptrVie->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);
int numOfCodecs = _ptrVie->GIPSVideo_GetNofCodecs();
for(int i=0; i<numOfCodecs;++i)
{
  GIPSVideo_CodecInst codec;
  _ptrVie->GIPSVideo_GetCodec(i,&codec);
  DISPLAY_CODEC_INFO(i,codec);
}
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetSendCodec

This function sets the codec for a channel that is to be used for sending.

Syntax

```
int GIPSVideo_SetSendCodec(      int channel,
                                 GIPSVideo_CodecInst *codec_inst,
                                 bool def)
```

Parameters

channel	[in] The channel ID number.
codec_inst	[in] Pointer to the `GIPSVideo_CodecInst` struct, containing the settings for the encoder (the outgoing video stream).
def	[in] Set to true if the same encoder instance will be shared with other channels.

Return Values

Returns 0 if the codec and the settings are supported and –1 otherwise.

Remarks

The MIME information (payload name) must be the same as those types supported by the GIPS VideoEngine. Height and width of `codec_inst` must be the same if several channels are used. If this setting is going to be used for all channels, set the def argument to true to enable efficient encoding.

Example Code

The below example will set LSVX as the codec on a video channel.

```
GipsVideoEngineWindows* _ptrVie = &GetGipsVideoEngine();

_ptrVie->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);
int channel = _ptrVie->GIPSVideo_CreateChannel(-1);

GIPSVideo_CodecInst codec;
strncpy(codec.plname, "LSVX", 5);
codec.pltype = 97;
codec.level = 5;
codec.quality = GIPS_QUALITY_DEFAULT;
codec.bitRate = 200;
codec.maxBitRate = 200;
codec.width = 352;
codec.height = 288;
codec.frameRate = 30;
codec.configParameterSize = 0;
codec.codecSpecific = GIPS_LSVX_KEY;
_ptrVie->GIPSVideo_SetSendCodec(channel, &codec,false);
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSendCodec

This function gets the currently configured video codec used to encode the video stream on a channel.

Syntax

```
int GIPSVideo_GetSendCodec(      int channel,
                                 GIPSVideo_CodecInst *codec_inst)
```

Parameters

Channel	[in] The channel ID number.
codec_inst	[out] Pointer to a struct with the settings for the outgoing video stream, struct allocated by caller

Return Values

The return value is 0 if the current send codec information for that channel was successfully copied to the given address space and –1 if an error occurred.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetReceiveCodec

This function registers a possible decoder setting for a received video stream on a certain channel. Call this function for all received codec settings in your call-setup.

Syntax

```
int GIPSVideo_SetReceiveCodec(    int channel,
                                   GIPSVideo_CodecInst* codec_inst,
                                  bool force)
```

Parameters

channel	[in] The channel ID number.
codec_inst	[in] Pointer to a `GIPSVideo_CodecInst` struct with the settings for the incoming video stream.
force	[in] Apply new `codec_inst`, if this API is already called with same payload type.

Remarks

The MIME information (payload name) must be the same as those types supported by the GIPS VideoEngine. Call `GIPSVideo_SetReceiveCodec` for each combination of size and payload type that is negotiated by the call-setup.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetReceiveCodec

This function gets the currently active video codec received on a channel.

Syntax

```
int GIPSVideo_GetReceiveCodec(   int channel,
                                 GIPSVideo_CodecInst* codec_inst)
```

Parameters

channel	[in] The channel ID number.
codec_inst	[out] Pointer to a `GIPSVideo_CodecInst` struct with the settings for the incomging video stream. The struct must be allocated by caller.

Return Values

The return value is 0 if the current receive codec information for the channel was successfully copied to the given address space and –1 if an error occurred.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SendKeyFrame

This function is called to force VideoEngine to encode the next video frame as a keyframe. Used if you use out-band signaling to request new key frames. This api can be used efficiently with the callback api `RequestNewKeyFrame()` to handle the keyframe requests for h263 and h264 video codecs.

Syntax

```
int GIPSVideo_SendKeyFrame(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

The return value is 0 if successful and –1 if an error occurred.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableRemoteResize

When using LSVX the sender will get feedback on the remote rendering size, this size can be used to go up or down in frame size for better utilization or higher quality. **This feature is only available when using LSVX.**

Syntax

```
int GIPSVideo_EnableRemoteResize(       int channel,
                                        bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] Set to true to enable this feature.

Remarks

This api should not be used in Conferencing scenarios.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetInverseH263Logic

Some Microsoft clients have inversed the logic for signaling key and delta frames. This function allows interoperability with those clients.

Syntax

```
int GIPSVideo_SetInverseH263Logic(int channel, bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] Set to true to enable this feature.

Remarks

GIPS cannot guarantee if or when Microsoft will change to a correct behavior that follows the standard.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnablePacketLossBitrateAdaption

This function enables GIPS traffic-shaping to go down in bit rate due to packet loss alone. If this feature is not enabled packet loss alone is not enough for reducing the bit rate. Roundtrip time (RTT) or bandwidth estimate will also have to signal poor network condition to reduce the bit rate.

Syntax

```
int GIPSVideo_EnablePacketLossBitrateAdaption(     int channel,
                                                   bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] Set to true to enable this feature.

Remarks

Some networks might have constant high (> 10%) packet loss. To avoid the bit rate being adapted down to minimum bit rate, set enable = FALSE.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetThresholdToSignalRemoteResize

This function requests the remote end to reduce the complexity, if CPU usage hits certain threshold limit. If the CPU usage goes down to below the threshold – 30% of threshold, original resolution will be restored.

Syntax

```
int GIPSVideo_SetThresholdToSignalRemoteResize(int threshold,
                                               int remoteWidth,
                                               int remoteHeight)
```

Parameters

threshold	[in] CPU threshold limit.
remoteWidth	[in] New width if CPU reaches threshold limit.
remoteHeight	[in] New height if CPU reaches threshold limit.

Remarks

This function only works with LSVX codec. The remoteWidth and remoteHeight should be less than the original size in GIPSVideo_SetSendCodec. No sanity checking is done in the VideoEngine for the height and width input values.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Codec sizes

GIPS VideoEngine supports the following sizes in pixels:

Codec Size			H.263	H.264	MPEG-4	VP7	LSVX	LSVX-S
Name	**Width**	**Height**						
SQCIF	128	96	X	X	X	X	X	
QQVGA	160	120		X	X	X	X	
QCIF	176	144	X	X	X	X	X	X
QVGA	320	240		X	X	X	X	X
CIF	352	288	X	X	X	X	X	X
VGA	640	480		X	X	X	X	X
Wide VGA	800	480		X			X	
4CIF	704	576	X	X			X	
SVGA	800	600		X			X	
HD	960	720		X			X	
Wide HD	1280	720		X			X	
XGA	1024	768		X			X	
Full HD	1440	1080		X				
W Full HD	1920	1080		X				

SQCIF is not supported by all cameras, the cameras that don't support it can still be used to send an encoded SQCIF stream; the engine will automatically cut the QQVGA to SQCIF.

Many camcorders use widescreen formats. The engine accepts all of these and will automatically cut or pad the images to the appropriate size.

NOTE: Your build might be restricted to a smaller size. Please refer to the GIPS sales engineer for your details.

Codec Settings - LSVX and LSVX-S

GIPS VideoEngine can use three different approaches to LSVX encode a video stream regarding target bit rate.

Enum GIPSVideoQuality

The quality parameter holds information about how the codec will adapt to network conditions and bitrates.

Syntax

```
enum GIPSVideoQuality
{
  GIPS_QUALITY_DEFAULT = 0;
  GIPS_QUALITY_MIN_FRAME_RATE = 1;
  GIPS_QUALITY_VIDEO = 2;
  GIPS_QUALITY_TALKING_HEAD = 3;
};
```

Parameters

Quality/ Codec	LSVX	LSVX-S
GIPS_QUALITY_DEFAULT	GIPS default behavior	GIPS default behavior
GIPS_QUALITY_MIN_FRAME_RATE	High quality, low frame rate	Will fail
GIPS_QUALITY_VIDEO	High frame rate, low quality	Will fail
GIPS_QUALITY_TALKING_HEAD	Best quality, low frame rate	Will fail

Set the quality parameter, in the `GIPSVideo_SetSendCodec()` input argument `GIPSVideo_CodecInst`, to one of the following options to determine what mode to use:

- If the available bit rate is lower than the current send bit rate, the `GIPS_QUALITY_MIN_FRAME_RATE` mode will keep the picture quality high and lower the send frame rate if needed. Set the level member in `GIPSVideo_CodecInst` to the minimum allowed frame rate.
- `GIPS_QUALITY_VIDEO` mode keeps the frame rate high and reduces the picture quality if needed.
- `GIPS_QUALITY_TALKING_HEAD` mode will always use the best picture quality on the outgoing video stream. To keep the bit rate constraints, the frame rate will be lowered.

Four different modes of signaling is available in LSVX. It's configured via the codecSpecific input argument in `GIPSVideo_CodecInst`.

LSVX and LSVX-S syntax

```
enum GIPSVideoSignalingLSVX
{
  GIPS_LSVX_KEY = 0,
  GIPS_LSVX_NACK = 0x01,
  GIPS_LSVX_FEC = 0x02,
  GIPS_LSVX_RECEIVE = 0x04
};
```

LSVX and LSVX-S parameters

	LSVX and LSVX-S behavior when a packet is detected as lost
GIPS_LSVX_KEY	Signal key frame request.
GIPS_LSVX_NACK	Signal negative acknowledgement.
GIPS_LSVX_FEC	No action. Forward error correction is used to minimize the effect.
GIPS_LSVX_RECEIVE	Sends LSVX signaling packets. Codec in receive mode which means that no other packets are sent.

Codec Settings H.263

H.263 only supports 4CIF, CIF, QCIF and SQCIF formats. GIPS VideoEngine does not support H263+ or H.263++.

H.263 syntax

```
enum GIPSH263FrameDrop
{
  GIPS_H263_DECODE_P_FRAMES = 0,
  GIPS_H263_DROP_P_FRAMES = 1
};
```

H.263 parameters

	H263 behavior
GIPS_H263_DECODE_P_FRAMES	Decode all P-frames.
GIPS_H263_DROP_P_FRAMES	Decode (all) P-frames only after I-frame has been received.

GIPS_H263_DROP_P_FRAMES

Available in mixing mode only, drops all incoming P-frames before an I-frame has been received.

- Set codec Specific to `GIPS_H263_DROP_P_FRAMES` in `GIPSVideo_SetReceiveCodec()` to drop all incoming P-frames before an I-frame has been received.
- Default, `GIPS_H263_DECODE_P_FRAMES`, is to decode all incoming P-frames even though no I-frame has been received.

SIP/SDP Call-setup

Remarks

The 1996 H.263 standardized version doesn't require any out-of-band signaling. The RTP profile is specified by RFC 2190.

Codec Settings – H.264

The following table lists the possible frame sizes and frame rates the codec can dynamically switch between for a given level. It's signaled via the message "profile-level-id" in SDP; see section 8.1 of RFC 3984.

To set the H.264 level, use the level member in `GIPSVideo_CodecInst`. The following list provides information about how to map H.264 level to SDP level, GIPS codec setting, frame size and maximum frame rates.

NOTE: The below table only serves as an example of the size and rate combinations that are allowed within that level. H264 codec is capable of encoding and decoding the size and rate less than level parameter set. In most of the cases application can set level 5.1. By setting level to 5.1, VideoEngine can encode and decode all the sizes supported.

Table 2 provides the maximum bit rates supported at each level

H264 Level		1	1b	1.1	1.2	1.3	2.0	2.1	2.2	3.0	3.1	3.2	4.0	4.1	4.2	5	5.1
Codec Level		10	10	11	12	13	20	21	22	30	31	32	40	41	42	50	51
SDP Level				0x0B	0x0C	0x0D	0x14	0x15	0x16	0x1E	0x1F	0x20	0x28	0x29	0x2A	0x32	0x33

Format

SQCIF		30.9	30.9	62.5	125.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QCIF		15.0	15.0	30.3	60.6	120.0	120.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0
QVGA				10.0	20.0	39.6	39.6	66.6	67.5	135.0	172.0	172.0	172.0	172.0	172.0	172.0	172.0
CIF					15.2	30.0	30.0	50.0	51.1	102.3	172.0	172.0	172.0	172.0	172.0	172.0	172.0
VGA									16.9	33.8	90.0	172.0	172.0	172.0	172.0	172.0	172.0
4CIF									12.8	25.6	68.2	136.4	155.2	155.2	172.0	172.0	172.0
XGA											35.2	70.3	80.0	80.0	172.0	172.0	172.0
Wide HD											30.0	60.0	68.3	68.3	145.1	163.8	172.0
W Full HD													30.1	30.1	64.0	72.3	120.5

Table 1 - H264 Maximum frame rates

GLOBAL IP SOLUTIONS

H264 Level	Max Bit rate (kbps)
1	64
1b	128
1.1	192
1.2	384
1.3	768
2	2000
2.1	4000
2.2	4000
3	10000
3.1	14000
3.2	20000
4	20000
4.1	50000
4.2	50000
5	135000
5.1	240000

Table 2 - Max Bit Rate vs. Level

To set the complexity member in the `GIPSVideo_CodecInst`, use the following information.

```
enum GIPSVideoComplexity
{
      GIPS_COMPLEXITY_NORMAL = 0x00,
      GIPS_COMPLEXITY_HIGH   = 0x01,
      GIPS_COMPLEXITY_HIGHER = 0x02,
      GIPS_COMPLEXITY_MAX    = 0x03
};
```

Parameters

	H264 behavior
GIPS_COMPLEXITY_NORMAL	QuadCore 2.4 GHz: CIF @ 150-350 fps, depending on video contents, bitrate and packet loss rate
GIPS_COMPLEXITY_HIGH	1.2 – 1.5 times normal complexity
GIPS_COMPLEXITY_HIGHER	2.0 – 2.5 times normal
GIPS_COMPLEXITY_MAX	3.2 – 4.5 times normal

Quality Parameter

To set the quality parameter, use GIPSVideo_CodecInst.quality.

- Range: 0 – 14; higher is better quality
- Default: -1, results in quality = 4
- Determines the minimum image quality
- Implicitly determines when the encoder should reduce frame rate in order to maintain image quality

SIP/SDP call-setup

The *profile-level-id* needs to be translated. The first byte indicates the profile, 0x42 indicates Base line. The second byte indicates that the NAL unit stream also obeys all constraints of the indicated profiles.The third byte indicates the level, see rfc3984 for details.

The sprop-parameter-sets should be passed to the codec by setting the configParameter member in the `GIPSVideo_CodecInst` struct if it is received.

For SIP/SDP packetization-mode equal to 0 set codecSpecific to GIPS_H264SingleMode. For packetization-mode equal to 1 set codecSpecific to GIPS_H264NonInterLeavedMode. Packetization-mode 2, interleaved is not supported by GIPS VideoEngine today.

Remarks

sprop-interleaving-depth, sprop-deint-buf-req, prop-init-buf-time and deint-buf-cap are all related to interleaving which is not supported by GIPS VideoEngine.

Codec Settings – H264-SVC

H264-SVC uses 3 layers.

- Base layer (layer 1) is the quarter the size, mentioned in the GIPSVideo_CodecInst.height and GIPSVideo_CodecInst.width
- Layer 2, is an enhancement layer with the same size set in GIPSVideo_CodecInst.height and GIPSVideo_CodecInst.width
- Layer 3, is an enhancement layer with the same size as layer 2, but better quality

NOTE: The bit rate specified in GIPSVideo_CodecInst.maxbitrate and GIPSVideo_CodecInst.bitrate is for the complete stream. It is not possible to set the bit rate for each layer. Of the total bit rate, layer1 uses 25%, layer 2 uses 25% and layer3, uses 50%.

Codec Settings - MPEG-4

Available levels in MPEG-4 are 0, 1, 2 and 3.

To set the MPEG-4 level, use the level member in `GIPSVideo_CodecInst.`

Codec Settings - VP7

Can be used in dial-up mode, configure frameRate to 10 and bitrate to a value lower than 20 in `GIPSVideo_CodecInst` to enable dialup-mode.

Picture Enhancements

GIPSVideo_EnableDeflickering

All cameras run the risk of getting in almost perfect sync with florescent lamps this will result in a very annoying flickering of the image. Most cameras have some type of filter to protect against this but not all of them succeed. Enabling this function will remove the flicker.

Syntax

```
int GIPSVideo_EnableDeflickering(bool enable)
```

Parameters

enable	[in] Set to true to enable this feature , this feature is default off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableDenoising

Some cheap cameras produce noisy images especially in low light conditions. Enable this function to reduce the camera noise.

Syntax

```
int GIPSVideo_EnableDenoising(bool enable)
```

Parameters

enable	[in] Set to true to enable this feature , this feature is default off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableColorEnhancement

This function enhances the colors in the camera video stream.

Syntax

```
int GIPSVideo_EnableColorEnhancement(int channel, bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] Set to false to disable this feature , this feature is default on.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableFrameUpScale

If the send codec size is bigger than the frame received from the camera, black border will be added to the frame. Use this function to up scale the image to send codec size, from the native camera frame size.

NOTE: This is CPU intensive task, use only if necessary.

Syntax

```
int GIPSVideo_EnableFrameUpScale(bool enable)
```

Parameters

enable	[in] Set to true to enable this feature , this feature is default off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableInterpolateScaling

This function enables the scaling mechanism, which down scales the camera framesize to the size specified in GIPSVideo_SetSendCodec. Down scaling is only required, when size of the frames received from the camera is larger than what is specided in GIPSVideo_SetSendCodec. This is not enabled, by default. Default mechanism crops the image size rather than the interpolating it to match the sending frame size.

NOTE: This is a high CPU intensive task therefore please use only if necessary.

Syntax

```
int GIPSVideo_EnableInterpolateScaling(bool enable)
```

Parameters

enable	[in] Set to true to enable this feature , this feature is default off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_MirrorLocalPreview

This function enables rendering of the mirrored local preview image from left – to - right

NOTE: This is a high CPU intensive task therefore please use only if necessary.

Syntax

```
int GIPSVideo_MirrorLocalPreview(bool enable)
```

Parameters

enable	[in] Set to true to enable this feature , this feature is default off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Network Settings - Standard Settings

Standard settings are function calls necessary to make a call using the built in socket support.

GIPSVideo_SetLocalReceiver

The port number to receive on is set for that specific channel. Also the source port for sending from is set to the same port number as default.

Syntax

```
int GIPSVideo_SetLocalReceiver( int channel,
                                unsigned short portnr,
                                char ip[64] = NULL,
                                unsigned short rtcpPortnr = 0,
                                char multiCastAddr[64] = NULL)
```

Parameters

channel	[in] The channel ID number.
portnr	[in] Local UDP port to listen on.
ip	[in] If defined local IP address to listen on.
rtcpPortnr	[in] If defined RTCP port to listen on, otherwise is port + 1 used for RTCP.
multiCastAddr	Not supported.

Return Values

0 is return if it successful and –1 otherwise.

Remarks

If another port is desired as source port the function `GIPSVideo_SetSrcPort()` should be called. There are two optional parameters to this call. To listen on a specific interface, if you have several NICs, set `IP` to the desired IP address to listen on.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetLocalReceiver

The local IP address of the channel is copied into the buffer ip with length 16. The local port number is copied to port.

Syntax

```
  int GIPSVideo_GetLocalReceiver ( int channel,
                                   unsigned short& rtpPort,
                                   unsigned short& rtcpPort,
                                   char ip[64])
```

Parameters

channel	[in] The channel ID number.
rtpPort	[out] Local RTP port that we are listening on.
rtcpPort	[out] Local RTCP port that we are listening on.
ip	[out] Local IP address that we are listening on.

Return Values

The function returns -1 if an error occurred, i.e. the channel does not exist.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetSendDestination

The IP address and port that packets are going to be sent to.

Syntax

```
int GIPSVideo_SetSendDestination(      int channel,
                                       unsigned short portnr,
                                       char ipaddr[64],
                                       unsigned short rtcpPortnr = 0)
```

Parameters

channel	[in] The channel ID number.
portnr	[in] UDP port to send RTP packet to.

ipaddr	[in] Remote IP address to send to.
rtcpPortnr	[in] If defined UDP port to send RTCP packets to, otherwise is portnr + 1 used.

Return Values

The function returns 0 if successful and –1 otherwise.

Example Code

The below example will set up in loopback using GIPS sockets.

```
GipsVideoEngineWindows* _ptrViE = &GetGipsVideoEngine();
_ptrViE->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);
int channel = _ptrViE->GIPSVideo_CreateChannel(-1);
_ptrViE->GIPSVideo_SetSendDestination(channel, 12345, “127.0.0.1”);
_ptrViE->GIPSVideo_SetLocalReceiver(channel, 12345);
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSendDestination

The remote IP address of the channel is copied into the buffer IP with length 16. The remote port number is copied to port.

Syntax

```
int GIPSVideo_GetSendDestination(      int channel,
                                       unsigned short& rtpPort,
                                       unsigned short& rtcpPort,
                                       char ipadr[64])
```

Parameters

channel	[in] The channel ID number.
rtpPort	[out] Remote UDP rtp port we send to.
rtcpPort	[out] Remote UDP rtcp port we send to.
ipadr	[out] Remote IP address we send to.

Return Values

The function returns -1 if an error occurred, i.e. the channel does not exist.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetFromPort

This function gets the sending port of the remote client

Syntax

```
unsigned short GIPSVideo_GetFromPort(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

Returns the source port of the packet last received on the channel or 0 if not available.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Network Settings – Optional Settings

This section contains information on advanced optional settings.

GIPSVideo_SetSrcPort

Sets the port number to send from (source port) for a specific channel.

Syntax

```
int GIPSVideo_SetSrcPort( int channel,
                          unsigned short RTPport
                           unsigned short RTCPport)
```

Parameters

channel	[in] The channel ID number.
RTPPort	[in] Source port of RTP packets.
RTCPPort	[in] Source port of RTCP packets.

Return Values

The function returns 0 if the port was set successfully and –1 otherwise.

NOTE: This function call must be made before `GIPSVideo_SetSendDestination()` Otherwise the default source port will be used (the default source port is the same as the receiving source port).

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableIPv6

This function sets a channel to use IPv6 instead of IPv4.

Syntax

```
int GIPSVideo_EnableIPv6(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetSendSSRC

According to the RFC the SSRC field in the RTP header is generated as a random number. GIPS VideoEngine usually generates this value. However, through this function call it is possible to specify the SSRC explicitly.

Syntax

```
int GIPSVideo_SetSendSSRC(       int channel,
                                 unsigned long int ssrc)
```

Parameters

channel	[in] The channel ID number.
ssrc	[in] The ssrc to use in the stream.

Remarks

This should be done before `GIPSVideo_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSendSSRC

Extract the RTP SSRC of the channel. The SSRC value corresponds to what was explicitly set by `GIPSVideo_SetSendSSRC()` or automatically generated by VideoEngine.

Syntax

```
unsigned long int GIPSVideo_GetSendSSRC(int channel, unsigned long int&
ssrc)
```

Parameters

channel	[in] The channel ID number.
ssrc	[out] SSRC.

Remarks

If VideoEngine generated, the value is unspecified before `GIPSVideo_StartSend()` is called.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetMTU

This function sets a Maximum Transition Unit (MTU) for a channel; the MTU is in bytes. The RTP packet will be packetized based on this MTU. The default MTU is 1460 in order to pass most IP networks un-fragmented.

Syntax

```
int GIPSVideo_SetMTU(int channel, int mtu)
```

Parameters

channel	[in] The channel ID number.
mtu	[in] max transmission unit in bytes, default is 1460 to guarantee that the RTP packet fits in a TCP packet. Valid values are between 100 and 1500 bytes.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetMaxPacketBurstSize

This function sets the maximum number of packets GIPS VideoEngine send to the network layer in one burst.

Syntax

```
int GIPSVideo_SetMaxPacketBurstSize(int channel, int maxNumberOfPackets)
```

Parameters

channel	[in] The channel ID number.
maxNumberOfPackets	[in] The maximum number of packets in each burst. Zero means no limit. The minimum value is 8.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SendExtraRTPPacket

This function handles sending a raw data packet over existing RTP channel.

Syntax

```
int GIPSVideo_SendExtraRTPPacket(int channel,
                                 const char* data,
                                 unsigned int length,
                                 unsigned short portnr,
                                 const char*ip)
```

Parameters

channel	[in] The channel ID number.
data	[in] A pointer to an array containing the data to be sent.
length	[in] The size of the array pointed to by data in bytes.
portnr	[in] Destination port number *optional.*
ip	[in] Destination IP address *optional.*

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux

Header	Declared in GipsVideoEngine.h

No RTP header is added to the data, only UDP/IP headers.

Sending must be active for this function to be valid.

GIPSVideo_SendExtraRTCPPacket

This function handles sending a raw data packet over existing RTCP channel.

Syntax

```
int GIPSVideo_SendExtraRTCPPacket(int channel,
                                  const char* data,
                                  unsigned int length,
                                  unsigned short portnr,
                                  const char*ip)
```

Parameters

channel	[in] The channel ID number.
data	[in] A pointer to an array containing the data to be sent.
length	[in] The size of the array pointed to by data in bytes.
portnr	[in] Destination port number *optional.*
ip	[in] Destination IP address *optional.*

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

No RTP header is added to the data, only UDP/IP headers

Sending must be active for this function to be valid

GIPSVideo_SetSendTOS

This function sets the six-bit Differentiated Service Code Point(DSCP) in the IP header of the outgoing stream for a specific channel.

Syntax

```
int GIPSVideo_SetSendTOS(  int channel,
                           int TOS,
                                    bool useSetSockopt = false)
```

Parameters

channel	[in] The channel ID number.
TOS	[in] The six-bit DSCP value. Valid range is 0-643. As defined in RFC 2472, the DSCP value is high-order 6 bits of the IP version 4(IPv4) TOS field and the IP version 6(IPv6) Trffic Class field.
useSetSockopt	[in_opt] I this parameter is true, the Windows Socket(Winsock) function setsockopt() is used internally. If the parameter is false, traffic conrol APIs are utilized instead.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

Remarks

It is recommended to use GIPSVideo_SetSendGQoS() on Windows instead of this function if possible.

This function must be always be called after GIPSVideo_SetLocalReceiver(), since it requires that socket already exists.

The useSetSockopt parameter is ignored on Linux. It is always interpreted as true internally, i.e. using the setsockopt() API is the only option on Linux.

By default (on Windows 2000/XP/2003) you must first specify a receiving IP address by calling GIPSVideo_SetLocalReceiver(). The NIC for a socket is found by IP address when dealing with Window Traffic Control. Binding to the local IP addess is not required if useSetSockopt is set to true.

Accoring to http://support.microsoft.com/kb/248611: *Microsoft Windows 2000, Microsoft Windows XP, and Microsoft Windows Server 2003 do not support the marking of Internet Protocol(IP) Type of Service(ToS) bits with the setsockopt() function.*

Setting the DSCP value on Windows requires that the executable runs with Administrator privileges. The DSCP value will not be modified unless this condition is fulfilled.

It is possible to modify the DSCP value "on the fly", i.e. while sending is active. However, it is recommended to consider this as a permanent setting for each RTP session.

Requirements

Supported platforms	Windows (incl. Mobile)
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSendTOS

This function gets the type of service field configured on a channel.

Syntax

```
int GIPSVideo_GetSendTOS( int channel,
                          int& DSCP,
                          bool& useSetSockopt)
```

Parameters

channel	[in] The channel ID number.
DSCP	[out] An integer reference where six-bit DSCP will be placed on return.
useSetSockopt	[out] A binary reference output which is set to true if the Windows socket (winsock) function setsockopt() is used internally. It is set to false if the traffic control APIs are utilized instead.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

NOTE: This call is not supported for the Mac and Linux platforms

Requirements

Supported platforms	Windows (incl. Mobile)
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetSendGQoS

This function sets the Generic Quality of Service (GQoS) service level. The Windows operating system then maps to a Differentiated Services Code Point (DSCP) and to an 802.1p setting.

Syntax

```
int GIPSVideo_SetSendGQoS( int channel,
                           bool enable,
                           int servicetype,
                          int overrideDSCP = 0)
```

Parameters

channel	[in] The channel ID number.
enable	[in] If this parameter is true, GQoS is enabled. If the parameter is false, GQoS is disabled.

servicetype	[in] The GQoS service type. The Windows operating system then maps to a DiffServ codepoint (DSCP) and to an 802.1p setting. See Table 3 - GQoS Values table below for more details.
overrideDSCP	[in_opt] Specifying this parameter overrides the DSCP value as mapped from the *servicetype* value, and the traffic control APIs will be used internally. If set to 0, the QoS APIs and normal mapping from the *serviceType* will be used. See Remarks section for more details.

Return Values

The return value is 0 if function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

Remarks

Setting the GQoS service level on Windows requires that the executable runs with Administrator privileges. The GQoS service level will not be modified unless this condition is fulfilled.

This function must be called after GIPSVideo_SetLocalReceiver, GIPSVideo_SetSendDestination and GIPSVideo_SetSendCodec to have any effect.

On Windows 2000/XP/2003, you must specify a local IP when calling GIPSVideo_SetLocalReceiver() if overrideDSCP is specified (i.e. > 0). This NIC for a socket is found by IP address when dealing with Windows Traffic Control.

The Windows GQoS API is used to modify both the DSCP and 802.1p marker bits. This function does this by setting a GQoS service level. The Windows operating system maps this to corresponding DSCP and 802.1p settings. The following table lists the supported default GQoS values.

Service Type Name	serviceType value (defined in qos.h)	DSCP	802.1p
Guaranteed Service	SERVICETYPE_GUARANTEED	0X28 (class selector 5)	5
Controlled Load	SERVICETYPE_CONTROLLEDLOAD	0X18(3)	3
Qualitative	SERVICETYPE_QUALITATIVE	0X0(0)	0
Best Effort	SERVICETYPE_BESTEFFORT	0X0(0)	0

Table 3 - GQoS Values

Using overrideDSCP will utilize the traffic control APIs, similar to when SetSendToS (without setsockopt) is called. The difference is that SetSendGQoS will set up an internally specified flow specification, including serviceType and other parameters. SetSendToS sets up a flow specification containing default values and unspecified parameters. Refer to MSDN library for details.

In order to change the DSCP value when using overrideDSCP, GQoS must first be disabled and then enabled again with the new value.

It is possible to modify the DSCP value "on the fly", i.e., while sending is active. However, it is recommended to consider this as a permanent setting for each RTP session.

NOTE: This call is not supported for the Windows Mobile, Mac and Linux platforms.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSendGQOS

This function gets the configured service type for a channel.

Syntax

```
int GIPSVideo_GetSendGQOS(int channel,
                          bool& enabled,
                          int& serviceType,
                          int& overrideDSCP)
```

Parameters

channel	[in] The channel ID number.
enabled	[out] The current GQoS state. If enabled is set to true, GQoS is enabled. If enabled is set to false, GQoS is disabled.
serviceType	[out] The GQoS service level is placed here on return.
overrideDSCP	[out] The non-default DSCP value (overrides the default mapping according to the Table 3 - GQoS Values above) is placed here on return.

Return Values

The return value is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

NOTE: This call is not supported for the Windows Mobile, Mac and Linux platforms.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngine.h

Network Settings - External Transport Protocol

The standard configuration of GIPS VideoEngine uses RTP/UDP/IP to transmit data over the network, but it does also support usage of an external transport protocol, if configured in that particular mode. In the external transportation-mode, the user of VideoEngine must handle sending and receiving packets from the network. Therefore an additional interface is needed so that the VideoEngine can call a send-function once a block of data has been encoded and packetized and a call to pass the packet received from the network to the VideoEngine. Note that the VideoEngine will deliver RTP/RTCP packets to the send-function and expects to receive the RTP/RTCP packets from the network. The information is packetized in RTP/RTCP-format because information such as payload type and sequence number is vital to make a correct decoding of the data. The following class and function calls enable an external transport protocol:

Class GIPS_transport

```
class GIPS_transport
{
public:
        int SendPacket(int channel, const void* data, int len)
        int SendRTCPPacket(int channel, const void* data,int len)
};
```

This class must be implemented by the user, which then will allow VideoEngine to call the send function once a block of data is captured, encoded and packetized. The following function should be called with the "`GIPS_transport`"-object so that VideoEngine has access to the object and thus is able to call the "`SendPacket()`"-function.

GIPSVideo_SetSendTransport

This function registers the external transport callback class.

Syntax

```
int GIPSVideo_SetSendTransport(  int channel,
                                 GIPS_transport *transport)
```

Parameters

channel	[in] The channel ID number.
transport	[in] GIPS_transport object, set to NULL to remove registration.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_ReceivedRTPPacket & GIPSVideo_ReceivedRTCPPacket

Packets received from the network should be passed to these functions.

Syntax

```
int GIPSVideo_ReceivedRTPPacket( int channel,
                                 const char* data,
                                 int length)
```

```
int GIPSVideo_ReceivedRTCPPacket(int channel,
                                 const char* data,
                                 int length)
```

Parameters

channel	[in] The channel ID number.
data	[in] Pointer to the memory of the incoming RTP packet.
length	[in] Length of the data pointed to by data.

NOTE: The data including the RTP/RTCP-header should be given to the VideoEngine.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Selecting Capture Device

GIPS VideoEngine requires I420, YUY2 or RGB24 support by the capture device for all video codecs to work properly. Call `GetCaptureDevice` with `listNr 0, 1, 2`… until it returns `–1`. Select the capture device with `SetCaptureDevice.`

NOTE: The list can change at any time, usually due to plug-in or removal of a USB camera. In Windows you can use `ON_WM_DEVICECHANGE` to get a notice. `OnDeviceChange(UINT nID, DWORD lParam)` will be called `with nID == DBT_DEVNODES_CHANGED` when a USB devices are inserted or removed.

GIPSVideo_GetCaptureDevice

This function returns the name of the capture device.

Syntax

```
int GIPSVideo_GetCaptureDevice(  int listNr,
                                 char* deviceName,
                                 int size)
```

Parameters

listNr	[in] Requested capture device in a non-sorted list.
deviceName	[out] Pointer to a char vector that will get the name of the capture device, UTF8 encoded. Vector allocated by caller.
size	[in] Size of char vector pointed to by **deviceName.**

Return Values

Returns 0 if the name of the capture device was successfully copied to the given space. Returns –1 if some error occurred.

Remarks

The `deviceName` is coded as UTF-8.

Example Code

Sample code to show usage of GetCaptureDevice() api.

```
GipsVideoEngineWindows* _viE = &GetGipsVideoEngine();
_viE->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);

char str[64];
bool found = false;
int captureIdx = 0;
while (-1 != _viE->GIPSVideo_GetCaptureDevice(captureIdx,str,sizeof(str)))
{
    int length = (int)strlen(str);
    if(length > 4)
    {
        printf("\tFound Camera: %s\n",str);
        found = true
    }
    captureIdx++;
    memset(str, 0, 64);
}
if(!found)
{
    printf("Error no camera connected, required for test");

}
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetCaptureDevice

This function selects the capture device listed by the `GetCaptureDevice`.

Syntax

```
int GIPSVideo_SetCaptureDevice(const char* deviceName, int size)
```

Parameters

deviceName	[in] Pointer to the name of capture device, UTF8 encoded.
size	[in] Size of char vector pointed to by **deviceName.**

Remarks

A previously set capture device can be released by calling this function with `deviceName` set to `NULL.`
`The deviceName` is UTF-8 coded.

Example Code

Sample code to show usage of GIPSVideo_SetCaptureDevice() api:

```
GipsVideoEngineWindows* _viE = &GetGipsVideoEngine();
_viE->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);

char str[64];
bool found = false;
int captureIdx = 0;
while (-1 != _viE->GIPSVideo_GetCaptureDevice(captureIdx,str,sizeof(str)))
{
     int length = (int)strlen(str);
     if(length > 4)
     {
         printf("\tFound camera: %s\n",str);
         found = true;
     }
     captureIdx++;
     memset(str, 0, 64);
 }
 if(!found)
 {
     printf("ERROR no camera connected, required for test\n");

  }

if(-1 == viE->GIPSVideo_GetCaptureDevice(0,str,sizeof(str)))
 {
     printf("ERROR in GIPSVideo_GetCaptureDevice\n");

 }
 printf("\tUsing camera: %s\n", str);

 if(-1 ==viE->GIPSVideo_SetCaptureDevice(str, sizeof(str)))
 {
     printf("ERROR in GIPSVideo_SetCaptureDevice\n");

}
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetCaptureCapabilities

This function is used to enumerate the capabilities of the current capture device (camera). Call with `listNr 0,1,2`… until it returns `–1`. Especially the size information is useful since it allows you to not negotiate a larger size than the quality of the camera supports.

Syntax

```
int GIPSVideo_GetCaptureCapabilities(int listNr,
                                      GIPSCameraCapability* capability)
```

Parameters

listNr	[in] index in the capture capabilities list.
capability	[out] `GIPSCameraCapability` struct containing the information about this capture device capabilities allocated by caller.

Remarks

This call has to be made after calling `GIPSVideo_SetCaptureDevice.`

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetCaptureCapabilities

Use this function to enumerate the capabilities of a capture device (camera) with name `deviceName` before calling `GIPSVideo_SetCaptureDevice`, `deviceName` is UTF-8 coded. Call with `listNr 0, 1, 2`... until it returns `–1.`

Syntax

```
int GIPSVideo_GetCaptureCapabilities( const char* deviceName,
                                      int size,
                                      int listNr,
                                      GIPSCameraCapability*)
```

Parameters

deviceName	[in] name of the capture device (camera) UTF-8 encoded.
size	[in] Size in bytes of the deviceName.
listNr	[in] index in the capture capabilities list.
GIPSCameraCapability	[out] `GIPSCameraCapability` struct containing the information about this capture device capabilities allocated by caller.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetCaptureDelay

Use this function to set the camera delay if it is known. I.e the time from when frame is captured by the capture device until it is available to GIPS VideoEngine. This can be used to improve lip synchronization if the capture delay is known. If this API is not called a standard delay value that can depend on the camera is used.

Syntax

```
int GIPSVideo_SetCaptureDelay(int cameraDelay)
```

Parameters

cameraDelay	[in] the delay in ms

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Start and Stop

This section describes the functions that start and stop rendering and capture.

NOTE: The functions `StartRender, StopRender, StartSend and StopSend` don't affect the preview or rendering of other channels.

GIPSVideo_StartRender

The packets are forwarded to the decoder for that specific channel and then rendered in the selected window.

Syntax

```
int GIPSVideo_StartRender(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

The function returns 0 if play out was started successfully and –1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopRender

Stops sending frames from the specified channel to the decoder, however, packets are still received as long as the VideoEngine is listening to the port.

Syntax

```
int GIPSVideo_StopRender(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

This function returns 0 if it is successful and –1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartSend

The channel starts encoding the frames and sending packets to the pre-specified IP address and port number.

Syntax

```
int GIPSVideo_StartSend(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

This function returns 0 if successful and –1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopSend

This function stops sending packets from that channel.

Syntax

```
int GIPSVideo_StopSend(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

The function returns 0 if the stop was made successfully and -1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_Run

This function starts the engine; packets are captured by the capture device and pushed through the encoder and to the local render if one is selected.

Syntax

```
int GIPSVideo_Run()
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_Stop

This function stops the engine started by `GIPSVideo_Run`.

Syntax

```
int GIPSVideo_Stop()
```

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Callbacks

This section specifies available callback classes.

GIPSVideoCallback

`GIPSVideoCallback` class is used for runtime feedback from the VideoEngine library.

Syntax

```
class GIPSVideoCallback
{
public:
  virtual void PerformanceAlarm(int value) = 0;
  virtual void BrightnessAlarm(int value) = 0;
  virtual void LocalFrameRate(int frameRate) = 0;
  virtual void MotionUpdate(unsigned char value) = 0;
  virtual void NoPictureAlarm(bool active = true) = 0;
};
```

`PerformanceAlarm` is called when the CPU has a high load (>80%). The value is the average CPU percentage.

`LocalFrameRate` gives the current frame rate used by the capture device.

`BrightnessAlarm` is called with the value if the brightness alarm is enabled, see `GIPSVideo_EnableBrightnessAlarm`, and a bright picture from the camera is detected. `BrightnessAlarm` is only called once for every alarm change. 2 means that the input from the capture device is very bright and 0 means the brightness is back to normal again.

`MotionUpdate` is called with a value describing the amount of motion, where `0` means no motion and `255` is the highest amount of motion. Call `GIPSVideo_EnableMotionUpdate` to enable the callback.

`NoPictureAlarm` is called when GIPS VideoEngine detects there are no incoming pictures from the capture device. Same callback method is being used to raise and to clear the NoPictureAlarm. A true value of the active variable raises the alarm while false clears the alarm.

NOTE: The callbacks might be generated within a critical section. It is recommended not to call VideoEngine API's within the user implemented callback functions. Instead ensure that the callback functions are kept as short as possible to prevent possible deadlocks.

GIPSVideoChannelCallback

`GIPSVideoChannelCallback` class is used for runtime feedback from the VideoEngine library.

Syntax

```
class GIPSVideoChannelCallback
{
public:
  virtual void IncomingRate(          int channel,
                                      int frameRate,
                                      int bitrate) = 0;
  virtual void IncomingCodecChanged(  int channel,
                                      int payloadType,
                                      int width,
                                      int height) = 0;
  virtual void IncomingCSRCChanged(   int channel,
                                      unsigned int csrc,
                                      bool added) = 0;

  virtual void RequestNewKeyFrame(    int channel) = 0;
  virtual void SendRate(              int channel,
                                      int frameRate,
                                      int bitrate) = 0;
};
```

`IncomingRate` is called with the current incoming framerate and bitrate. The bitrate is in bits per second and frameRate is in frames per second.

When a new payload type is received `IncomingCodecChanged`, with the channel number, payload type, width and height, is called with the new payload type so that the application can take the appropriate action.

`IncomingCSRCChanged` is called when the incoming SSRC or CSRC is changed, Boolean added signals if the SSRC/CSRC is added or removed. The added variable will be set to true when a CSRC/SSRC is added to the channel and will be set to false if the CSRC/SSRC is removed.

`RequestNewKeyFrame` is called when there are decode errors for H.263 and H.264. This makes it possible to request a new key frame from the remote side.

`SendRate` is called approximately once per second, it reports how many frames per second that are sent out on the network on this channel and the bit rate in bits per second on the channel.

NOTE: The callbacks might be generated within a critical section. It is recommended not to call VideoEngine API's within the user implemented callback functions. Instead ensure that the callback functions are kept as short as possible to prevent possible deadlocks.

GIPSEffectFilter

`GIPSEffectFilter` class is used for adding effects to an incoming stream; the `Transform` function is called once per frame, before passing the frame to the render.

Syntax

```
class GIPSEffectFilter
{
public:
  virtual int Transform(int size, unsigned char* frameBuffer, unsigned
                        int timeStamp90KHz);
};
```

The format of the `frameBuffer` is I420.

NOTE: The callbacks might be generated within a critical section. It is recommended not to call VideoEngine API's within the user implemented callback functions. Instead ensure that the callback functions are kept as short as possible to prevent possible deadlocks.

GIPSVideoRenderCallback

The `GIPSVideoRenderCallback` class allows you to render the incoming and local video streams using an external render. Each frame is delivered through a pure virtual callback class.

Syntax

```
class GIPSVideoRenderCallback
{
public:
    virtual int FrameSizeChange( int width,
         int height,
         int numberOfStreams) = 0;
    virtual int DeliverFrame(    unsigned char* buffer,
         int bufferSize, unsigned int timeStamp90KHz) = 0;
    virtual ~GIPSVideoRenderCallback() {};
};
```

Changes in the streams size will be signaled through `FrameSizeChange.` The input parameters are the width and height of the frame and the number of streams mixed in the frame.

`DeliverFrame` is responsible for the actual rendering. This will be invoked when a frame is ready to be rendered. The video engine assumes that the frame is rendered as soon as possible. The input parameters are the frame buffer to be rendered, the size of the buffer and the timestamp of the buffer.

NOTE: The callbacks might be generated within a critical section. It is recommended not to call VideoEngine API's within the user implemented callback functions. Instead ensure that the callback functions are kept as short as possible to prevent possible deadlocks.

External Capture

GIPSVideo_IncomingCapturedFrame

With a video stream originating from an unsupported device or another source than a video camera you can send in the frames direct to the engine.

Syntax

```
int GIPSVideo_IncomingCapturedFrame(
   GIPSVideoType incomingVideoType,
   unsigned char* incomingFrame,
   int width,
  int height,
  int bufferLength=0,
  unsigned long timeStamp=0)
```

Parameters

incomingVideoType	[in] Any of the supported GIPS video types such as I420, RGB24, YUY2 H264 or H263.
incomingFrame	[in] A pointer to the frame buffer.
widh	[in] Width of the input frame.
height	[in] Height of the input frame.

bufferLength	[in] Length of the frame buffer. Optional if the incomingVideoType is not an encoded frame.
timeStamp	[in] Timestamp in 90 kHz time base of when the frame was captured. If not provided a timestamp will be generated internally.

Remarks

`GIPSVideo_Run` must be called prior to this function.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

External render

GIPSVideo_Add LocalRenderer

This function registers a local render callback. It requires a GIPSVideoRenderCallback class object.

Syntax

```
int GIPSVideo_AddLocalRenderer(  GIPSVideoType videoFormat,
                                 GIPSVideoRenderCallback* obj)
```

Parameters

videoFormat	[in] Uncompressed video format that the render object will render in.
obj	[in] Render object that will receive all local frames to render, call with NULL to remove the object and stop the callbacks.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_AddRemoteRenderer

This function registers a remote render callback. It requires a GIPSVideoRenderCallback class object.

Syntax

```
int GIPSVideo_AddRemoteRenderer( int channel,
                                 GIPSVideoType videoFormat,
                                 GIPSVideoRenderCallback* obj,
                                 int renderWidth = 0,
                                 int renderHeight = 0)
```

Parameters

channel	[in] The channel ID number.
videoFormat	[in] Uncompressed video format that the render object will render in.
obj	[in] Render object that will receive all local frames to render, call with NULL to remove the object and stop the callbacks.
renderWidth	[in] Width of the image in pixels.
renderHeight	[in] Height of the image in pixels.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetCaptureDeviceId

This function enables distinguishing between devices with the same name since devices are not guaranteed to have unique names in Windows.

Syntax

```
int GIPSVideo_GetCaptureDeviceId(      int listNr,
                                       char* deviceNameUTF8,
                                       int sizeDeviceName,
                                       char*uniqueIdUTF8
                                       int sizeUniqueId)
```

Parameters

listNr	[in] Index in the capture capabilities list.
deviceNameUTF8	[out] Name of the capture device (camera) UTF-8 encoded.
sizeDeviceName	[in] Size in bytes of the deviceNameUTF8.
uniqueIdUTF8	[out] Unique identifier of the capture device (camera) UTF-8 encoded.
sizeUniqueId	[in] Size in bytes of UniqueIdUTF8.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetCaptureDeviceID

This function sets the capture device using its unique device ID.

Syntax

```
int GIPSVideo_SetCaptureDeviceId( const char* uniqueIdUTF8,
                                  int sizeUniqueId)
```

Parameters

uniqueIdUTF8	[in] Unique identifier of the capture device (camera) UTF-8 encoded.
sizeUniqueId	[in] Size in bytes of uniqueIdUTF8.

NOTE: This may change when you plug in/out the same capture device.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngine.h

Windows Specific Video Functions

The functions that use Windows specific types as argument are defined in the file GipsVideoEngineWindows.h.

GIPSVideo_SetCaptureCardProperties

This function enables the application to match the camera setting with the capture card settings. There are no negotiation between the camcorders and capture cards. The camera and the capture card must be configured with the same settings to get a video stream with good quality.

Syntax

```
int GIPSVideo_SetCaptureCardProperties(int width,
                                       int height,
                                       int frameRate = -1,
                                       bool interlaced = false)
```

Parameters

width	[in] Width in pixels of the capture card.
height	[in] Height in pixels of the capture card.
frameRate	[in] Frame rate in frames per second.
interlaced	[in] True if the source is interlaced.

Remarks

Capture cards normally don't support converting from one size to another. If the camera delivers HD 1280x720 the capture card also needs to be set to deliver HD 1280*720 frames using this function. Scaling can instead be done by setting a different width and height on the send codec. If this function is called with a frame size that differs from the camera settings some type of pause image will be sent from the capture card instead of the camera images.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_ViewCaptureDialogBox

The capture filter usually has a dialog box associated to it. This dialog box can be displayed with this function; the dialog box can be used to configure the capture device. The camera needs to be started before this api is called.

Syntax

```
int GIPSVideo_ViewCaptureDialogBox(    const char* dialogTitle,
                                        HWND m_hWnd = NULL,
                                        unsigned int x = 200,
                                        unsigned int y = 200);
```

Parameters

dialogTitle	[in] Title text of the window.
m_hWnd	[in] Window handle, if NULL a new window will be created.
x	[in] Horizontal position for the dialog box.
y	[in] Vertical position for the dialog box.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_SetBackgroundImage

This function sets a background image for the local preview, or for a specific channel.

Syntax

```
int GIPSVideo_SetBackgroundImage(      int channel,
                                       HBITMAP bitMap,
                                       unsigned int time = 0)
```

Parameters

channel	[in] The channel ID number. For local preview, use -1.
bitmap	[in] Handle to the bitmap.
time	[in] Time in milliseconds until the bitmap image is rendered.

Remarks

If `time` is set to zero, `bitMap` will be rendered before the first frame has been decoded, or captured in the case of local preview.

If `time` is larger than zero `bitMap` will be rendered if no new frame has been decoded when `time` milliseconds has elapsed since the last frame was rendered.

For local preview, `time` must be 0 milliseconds or equal to or higher than 1000 milliseconds.

Call the function with `bitMap` set to `NULL` to remove the picture. The startup picture will be removed if `time` is zero, otherwise will the timeout picture be removed.

NOTE: GIPS VideoEngine Windows supports 2 types of rendering on Windows DirectShow and DirectDraw. DirectDraw is recommended for all new projects.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddLocalRenderer

This function sets the Window in which the local video stream is rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(        HWND hWnd,
                                       int zOrder,
                                       float left,
                                       float top,
                                       float right,
                                       float bottom)
```

Parameters

hWnd	[in] Window handle in which this video stream will be rendered.
zOrder	[in] Z-order of this video stream.
left	[in] Left edge of video stream in window.
top	[in] Top edge of video stream in window.
right	[in] Right edge of video stream in window.
bottom	[in] Bottom edge of video stream in window.

Remarks

Argument `zOrder` is used if several streams are rendered to the same `HWND` overlap, `zOrder` 0 is on top and `zOrder` 1 is one layer behind layer 0. The arguments `left, top, right, bottom`, are the placement within the `HWND`, 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner. Setting `HWND` to `NULL` will release a previously added `HWND`. Note the same `hWnd` can be sent into `GIPSVideo_AddLocalRenderer` and `GIPSVideo_AddRemoteRenderer` for picture in picture or conference layouts.

NOTE: GIPS VideoEngine Windows supports 2 types of rendering on Windows DirectShow and DirectDraw. DirectDraw is recommended for all new projects.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddRemoteRenderer

This function sets the Window in which the remote video stream is rendered.

Syntax

```
int GIPSVideo_AddRemoteRenderer(       int channel,
                                       HWND hWnd,
                                       int zOrder,
                                       float left,
                                       float top,
                                       float right,
                                       float bottom)
```

Parameters

channel	[in] The channel ID number.
hWnd	[in] Window handle in which this video stream will be rendered.
zOrder	[in] Z-order of this video stream.
left	[in] Left edge of video stream in window.
top	[in] Top edge of video stream in window.
right	[in] Right edge of video stream in window.
bottom	[in] Bottom edge of video stream in window.

Remarks

Argument `zOrder` is used if several streams are rendered to the same HWND overlap. `zOrder` 0 is on top and `zOrder` 1 is one layer behind layer 0, `zOrder` 2 is behind `zOrder` 1, … The arguments `left, top, right, bottom,` are the placement within the HWND, 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner. Setting HWND to NULL will release a previously added HWND. Note the same hWnd

GLOBAL IP SOLUTIONS

can be sent into `GIPSVideo_AddLocalRenderer` and `GIPSVideo_AddRemoteRenderer` for picture in picture or conference layouts.

NOTE: GIPS VideoEngine Windows supports 2 types of rendering on Windows DirectShow and DirectDraw. DirectDraw is recommended for all new projects.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_SetCropping

This function crops the displayed video image for a specified `channel` and `streamId`.

Syntax

```
int GIPSVideo_SetCropping(      int channel,
                                unsigned char streamID,
                                float left,
                                float top,
                                float right,
                                float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Index of stream to crop*.
left	[in] Left edge of video stream in window.
top	[in] Top edge of video stream in window.
right	[in] Right edge of video stream in window.
bottom	[in] Bottom edge of video stream in window.

Remarks

To crop the local rendered image, set `channel` to `-1`. The arguments `left, top, right, bottom,` are the placement within the `HWND`, 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner. For one stream the streamID would be 0. In mixing, StreamID's are incremented from 0 to the number of streams in the signal. If there are 3 streams in a mixed signal, there StreamID's should be 0, 1 and 2.

NOTE: GIPS VideoEngine Windows supports 2 types of rendering on Windows DirectShow and DirectDraw. DirectDraw is recommended for all new projects.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_EnableDirectDraw [DirectDraw]

This function enables DirectDraw as rendering method.

Syntax

```
int GIPSVideo_EnableDirectDraw(bool enable)
```

Parameters

enable	[in] `turn on/off direct draw rendering`

NOTE: DirectDraw is required for transparent bitmaps and true full screen mode.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_EnableDirect3D [Direct3D]

This function enables Direct3D as rendering method.

Syntax

```
int GIPSVideo_EnableDirect3D(bool enable)
```

Parameters

enable	[in] `Turn on/off direct 3d rendering.`

Requirements

Supported platforms	Windows

Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_GetD3DSurface [Direct3D]

This function returns handle to Direct3DSurface which is being used by the Direct3D renderer.

Syntax

```
LPDIRECT3DSURFACE9 GIPSVideo_GetD3DSurface(HWND hWnd)
```

Parameters

hWnd	[in] Window handle used during the creation of the Direct3D renderer (local or remote).

Remarks

On failure, this function returns NULL.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_EnableTransparentBackground [DirectDraw]

This function enables transparent background.

Syntax

```
int GIPSVideo_EnableTransparentBackground(HWND hWnd, bool enable)
```

Parameters

hWnd	[in] Window handle.
enable	[in] On/Off.

Remarks

If the full HWND is not filled with the current rendering, this setting make the background transparent so that underlying GUI is visible, if this is not set the un-configured area will be painted black.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddFullScreenRender [DirectDraw]

This function makes the hWnd fill the full-screen. The screen will be changed to exclusive mode and no other applications will have access to the screen.

Syntax

```
int GIPSVideo_AddFullScreenRender(HWND hWnd)
```

Parameters

hWnd	[in] Window handle.

Remarks

Make sure that your application catches ESC and CTRL+ALT+DEL to your liking. Recommended behavior is to change to normal rendering in a normal hWnd after ESC or CTRL+ALT+DEL. To exit full-screen mode call `GIPSVideo_AddFullScreenRender` with `NULL.`

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddRenderBitmap [DirectDraw]

This function enables RGB bitmaps to be rendered on top of any streaming video.

Syntax

```
int GIPSVideo_AddRenderBitmap(   HWND hWnd,
                          unsigned char pictureId,
                          HBITMAP bitMap,
                          float left,
                          float top,
                          float right,
                          float bottom,
                          DDCOLORKEY* transparentColorKey = NULL)
```

Parameters

hWnd	[in] Window handle.
pictureId	[in] Index of picture.

bitMap	[in] Handle to bitmap.
left	[in] Left border of bitmap.
top	[in] Top border of bitmap.
right	[in] Right border of bitmap.
bottom	[in] Bottom border of bitmap.
transparentColorKey	[in] Color key for transparency.

Remarks

RGB bitmaps can be rendered on-top of any streaming video. `bitMap` is the handle to the bitmap that you want to render, left, top, right and bottom is the location in the hWnd where the bitmap will be placed. The valid range for left, top, right and bottom is 0.0f to 1.0f. If left, top, right and bottom describe a point, the original bitmap will be placed in the hWnd without any stretching or scaling. The bitmap can have transparent fields in any color. The transparent color is described in a color range supplied by `transparentColorKey.` Default is no transparent fields.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddRenderText [DirectDraw]

This function enables text to be rendered on-top of any bitmap or streaming video.

Syntax

```
int GIPSVideo_AddRenderText(     HWND hWnd,
                                 unsigned char textId,
                                 const char* text,
                                 int textLength,
                                 COLORREF colorText,
                                 COLORREF colorBg,
                                 float left,
                                 float top,
                                 float right,
                                 float bottom,
                                 bool transparent)
```

Parameters

hWnd	[in] Window handle.
textId	[in] Index of text .
text	[in] Text to render.
textLength	[in] Length of the text.
colorText	[in] Color of the text.
colorBg	[in] Background color of text.
left	[in] Left border of textbox.
top	[in] Top border of textbox.
right	[in] Right border of textbox.
bottom	[in] Bottom border of textbox.
transparent	[in] Transparent background in textbox.

Remarks

`textId` is the identifier for this string. `colorText` is the color of the text. `colorBg` is the color of the background. `Left, top, right` and `bottom` is the position of a textbox in which the text will be displayed. Set `transparent` to true if you don't want a background for the text.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_ConfigureRender [DirectDraw]

This function changes the layout of the incoming stream or channel. If the incoming signal is a mixed signal, this api should be called for all sub-streams in the mixed signal.

Syntax

```
int GIPSVideo_ConfigureRender(   int channel,
                                 unsigned char streamID,
                                 HWND hWnd,
                                 int zOrder,
                                 float left,
                                 float top,
                                 float right,
                                 float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Video stream index*.
hWnd	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Example Code

This example assumes that the incoming video contains 3 mixed streams. Mixed stream will be demuxed and co-ordinates for the 3 streams should be withtin the HWND. It also assumes that you have initialized the video engine and created a video channel and are rendering on remote myhwnd.

```
_videoEngine->GIPSVideo_ConfigureRender(channel1, 0, myhwnd, 1,
                                         0.0f, 0.0f,0.3f,0.3f);
_videoEngine->GIPSVideo_ConfigureRender(channel1, 1, myhWnd, 1,
                                         0.7f, 0.0f,1.0f,0.3f);
_videoEngine->GIPSVideo_ConfigureRender(channel1, 2, myhWnd, 1,

                                         0.0f, 0.7f,0.3f,1.0f);
```

Remarks

For one stream the streamID would be 0. In mixing, streamID's are incremented from 0 to the number of streams in the signal. If there are 3 streams in a mixed signal, the streamID's should be 0, 1 and 2. If the channel receives multiple sub-streams, each sub-stream can be demuxed and positioned in any way within the `hWnd` with the ConfigureRender function. `Channel` and `streamID` describe which stream is to be positioned. `left, top, right, and bottom` describe the position within the `hWnd.` The valid range is 0.0f to 1.0f for `left, top, right and bottom` arguments. `zOrder` describes the order in which the sub streams are rendered. If multiple streams have the same `zOrder` and are overlapping the rendering, the order is undefined.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddRemoteDemuxRender [DirectDraw]

This function is similar to `GIPSVideo_ConfigureRender` but allows demux of one sub-stream to its own `hWnd.`

Syntax

```
int GIPSVideo_AddRemoteDemuxRender(    int channel,
                                       unsigned char streamID,
                                       HWND hWnd,
                                       int zOrder,
                                       float left,
                                       float top,
                                       float right,
                                       float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Video stream index.
hWnd	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Example Code

This code will create 4 different HWND windows for the 4 streams to be rendered on. It assumes that the engine has been setup, channel has been created and remote rendering has started. If there is only one 1 incoming stream on channel1, then it would display the remaining 3 HWND windows as black.

```
HWND _hwnd1, _hwnd2, _hwnd3, _hwnd4;
 GIPS_CreateWindow(_hwnd1, 0, 352, 288);
 GIPS_CreateWindow(_hwnd2, 1, 352, 288);
 GIPS_CreateWindow(_hwnd3, 2, 352, 288);
 GIPS_CreateWindow(_hwnd4, 3, 352, 288);
_videoEngine->GIPSVideo_AddRemoteDemuxRender(channel1, 0, _hwnd1, 0,
                                            0.0f, 0.0f,1.0f,1.0f);
 _videoEngine->GIPSVideo_AddRemoteDemuxRender(channel1, 1, _hwnd2, 0,
                                            0.0f, 0.0f,1.0f,1.0f);
 _videoEngine->GIPSVideo_AddRemoteDemuxRender(channel1, 2, _hwnd3, 0,
                                            0.0f, 0.0f,1.0f,1.0f);
_videoEngine->GIPSVideo_AddRemoteDemuxRender(channel1, 3, _hwnd4, 0,
                                            0.0f, 0.0f,1.0f,1.0f);
```

Remarks

For one stream the streamID would be 0. In mixing, streamID's are incremented from 0 to the number of streams in the signal. If there are 3 streams in a mixed signal, thestreamID's should be 0, 1 and 2. If the channel receives multiple sub-streams each sub-stream can be demuxed and positioned in any way within their own `hWnd` with the addRemoteDemuxRenderer api. 0.0f to 1.0f is the valid range for `left, top, right and bottom` arguments.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_EnableMixingRender [DirectShow]

This function enables mixing of the rendered images, which is needed for picture-in-picture functionality or running as conference mixer.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_EnableMixingRender(bool enable)
```

Parameters

Enable	[in] On/Off.

Remarks

Loads a Video Mixing render. VideoEngine support both VideoMixingRender7 and VideoMixingRender9. Not enabling mixing render causes the VideoEngine to uses the generic VideoRender for rendering.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_ConfigureMixer [DirectShow]

This function sets the window in which the stream from a certain conference participant will be placed. This function is analogous to GIPSVideo_ConfigureRender api. The only difference is that GIPSVideo_ConfigureRender() uses directDraw while this api uses DirectShow.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_ConfigureMixer(       int channel,
                                    char streamID,
                                    HWND hWnd,
                                    int zOrder,
                                    float alpha,
                                    float left,
                                    float top,
                                    float right,
                                    float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Video stream index.
hWnd	[in] Window handle.
zOrder	[in] Z-order of video stream.
alpha	[in] Degree of transparency.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

For one stream the streamID would be 0. In mixing, streamID's are incremented from 0 to the number of streams in the signal. If there are 3 streams in a mixed signal, the streamID's should be 0, 1 and 2. `zOrder` is used if several streams are rendered to the same `HWND` overlap. `zOrder` 0 is on top and `zOrder` 1 is one layer behind layer 0. The arguments `left, top, right, bottom,` are the placement within the `HWND`. 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

NOTE: This call require a successful call to GIPSVideo_EnableMixingRender(true)

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_GetSnapShot [DirectShow]

This function gets a snapshot of the picture viewed in the `Window hWnd.`

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_GetSnapShot( HWND hWnd,
                           LPBITMAPINFOHEADER header,
                           int size)
```

Parameters

hWnd	[in] Window handle.
header	[out] Bitmap header allocate by caller.
size	[in] Size of bitmap header.

Remarks

`hWnd` must be one of the Windows previously set by `AddLocalRenderer` or `AddRemoteRenderer.`

NOTE: This call require a successful call to GIPSVideo_EnableMixingRender(true)

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_OnPaint [DirectShow]

When a window from another application's stop covers the window in which the video is rendered, the application gets an `OnPaint` message from Windows that needs to be sent down to GIPS VideoEngine to re-start painting the window correctly.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_OnPaint(HDC hdc)
```

Parameters

hdc	[in] Handle to device context,

NOTE: This call require a successful call to GIPSVideo_EnableMixingRender(true)

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_OnSize [DirectShow]

When a `HWND` is resized, the application gets an `OnSize` message from Windows and this need to be sent down to GIPS VideoEngine to scale the rendering of the video to the new window size.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_OnSize(HWND hWnd)
```

Parameters

hWnd	[in] Window handle.

NOTE: This call require a successful call to `GIPSVideo_EnableMixingRender`(true).

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_OnDisplayMode [DirectShow]

When the resolution or color depth is changed for a screen, the application gets a `WM_DISPLAYCHANGE` message from Windows, this need to be sent down to GIPS VideoEngine to re-start painting the window correctly.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_OnDisplayMode()
```

Parameters

hWnd	[in] Window handle.

NOTE: This call require a successful call to `GIPSVideo_EnableMixingRender(true).`

GIPSVideo_GetAssociatedRenderFilter [DirectShow]

This function returns a reference to the render filter associated with the given window.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_GetAssociatedRenderFilter(HWND hWnd,
                                         IBaseFilter** filter)
```

Parameters

hWnd	[in] Window handle.
filter	[out] Pointer to render filter.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_ChangeHWND [DirectShow]

This function changes the HWND for a channel.

NOTE: DirectShow rendering will be deprecated in future releases of GIPS VideoEngine.

Syntax

```
int GIPSVideo_ChangeHWND( int channel,
                          HWND hWnd,
                          HWND oldhWnd)
```

Parameters

channel	[in] The channel ID number.
hWnd	[in] New window handle.
oldhWnd	[in] Old window handle.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddLocalRenderer [DirectShow]

This function allows for use of a different render filter than the default render filter.

Syntax

```
int GIPSVideo_AddLocalRenderer(REFCLSID)
```

Parameters

REFCLSID	[in] GUID (global unique identifier) of the render filter.

Syntax

```
int GIPSVideo_AddLocalRenderer(IBaseFilter*)
```

Parameters

IBaseFilter	[in] Pointer to a render filter implemented on top of an IBaseFilter.

Remarks

Supports `REFCLSID` or `IBaseFilter` as input.

NOTE: The input pin must support video format I420.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

GIPSVideo_AddRemoteRenderer

This function allows for use of a different render filter than the default render filter.

Syntax

```
int GIPSVideo_AddRemoteRenderer(int channel, REFCLSID)
```

Parameters

channel	[in] The channel ID number.
REFCLSID	[in] GUID (global unique identifier) of the render filter.

Syntax

```
int GIPSVideo_AddRemoteRenderer(int channel, IBaseFilter*)
```

Parameters

channel	[in] The channel ID number.
IBaseFilter	[in] Pointer to a render filter implemented on top of an IBaseFilter.

Remarks

Supports `REFCLSID` or `IBaseFilter` as input.

NOTE: The input pin must support video format I420, RGB24 or ARGB32.

Requirements

Supported platforms	Windows
Header	Declared in GipsVideoEngineWindows.h

Mac OS X Specific Functions

The functions that use Mac OS X specific types as argument are defined in the file GIPSVideoEngineMac.h.

Mac OS X GIPS Video Library supports both Carbon and Cocoa.

Carbon applications can use one of two different methods for rendering the video on the screen: The Carbon and OpenGL renderers. GIPS recommends that all customers using Carbon switch to OpenGL to avoid shortcomings with Carbon. In Mac OS X version 10.5 the Carbon render is no longer thread safe without using the functions `GIPSVideo_RenderLock and GIPSVideo_RenderUnLock.`

Carbon specific calls are listed first, and then Cocoa specific calls are listed at the end of this section.

GIPSVideo_EnableOpenGLRendering [Carbon]

This function enables OpenGL as rendering method instead of Carbon rendering in a Carbon application.

Syntax

```
int GIPSVideo_EnableOpenGLRendering(bool enable)
```

Parameters

Enable	[in] On/Off.

Remarks

The standard rendering APIs are used to add renderers. An Intel Macintosh running OS X version 10.4 or later is required to use OpenGL.

NOTE: Cocoa applications do not need to call this function.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddLocalRenderer [Carbon]

This function sets the window in which the local video stream is rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(  WindowRef renderWindow,
                                 int zOrder,
                                 float left,
                                 float top,
                                 float right,
                                 float bottom)
```

Parameters

renderWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`renderWindow` is a window reference to the window to output the video. The arguments `left, top, right,` and `bottom` are the placement within the `WindowRef.` 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddRemoteRenderer [Carbon]

This function sets the Window in which the remote video stream for a specific channel is rendered.

Syntax

```
int GIPSVideo_AddRemoteRenderer(   int channel,
                                   WindowRef remoteWindow,
                                   int zOrder,
                                   float left,
                                   float top,
                                   float right,
                                   float bottom)
```

Parameters

channel	[in] The channel ID number.
remoteWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`remoteWin` is a window reference to the window to output the video. The arguments `left, top, right, and bottom` are the placement within the WindowRef. 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddLocalRenderer [Carbon]

This function sets the window in which the local video stream is rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(  HIViewRef renderWindow,
                                 int zOrder,
                                 float left,
                                 float top,
                                 float right,
                                 float bottom)
```

Parameters

renderWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`renderWindow` is a window reference to the window to output the video. The arguments `left, top, right,` and `bottom` are the placement within the `WindowRef.` 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddRemoteRenderer [Carbon]

This function sets the Window in which the remote video stream for a specific channel is rendered.

Syntax

```
int GIPSVideo_AddRemoteRenderer(   int channel,
                                   HIViewRef remoteWindow,
                                   int zOrder,
                                   float left,
                                   float top,
                                   float right,
                                   float bottom)
```

Parameters

channel	[in] The channel ID number.
remoteWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.

left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`remoteWin` is a window reference to the window to output the video. The arguments `left, top, right,` and `bottom` are the placement within the WindowRef. 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddRemoteDemuxRender [Carbon]

This function is user to demux one sub stream of an incoming “Brady Bunch” stream.

Syntax

```
int GIPSVideo_AddRemoteDemuxRender(    int channel,
                                       unsigned char streamID,
                                       WindowRef remoteWindow,
                                       int zOrder,
                                       float left,
                                       float top,
                                       float right,
                                       float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Stream index.
remoteWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.

bottom	[in] Bottom border of video stream.

Remarks

The input arguments are similar to `GIPSVideo_AddRemoteRenderer,` with the addition of the id for the sub stream.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_ConfigureMixer [Carbon]

This function sets the window in which the stream from a certain conference participant will be placed.

Syntax

```
int GIPSVideo_ConfigureMixer(   int channel,
                                int streamID,
                                WindowRef renderWindow,
                                float left,
                                float top,
                                floar right,
                                float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Stream index.
renderWindow	[in] Window handle.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

The arguments `left, top, right`, and `bottom` is the placement in the window given as a float value between 0.0 and 1.0, starting top left in the window.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_RenderLock [Carbon]

Rendering in Carbon windows and handling window events is not thread safe running OS X 10.5. This function can be used to lock the VideoEngine rendering to avoid event handling and rendering being performed in the same time.

Syntax

```
int GIPSVideo_RenderLock()
```

Remarks

Make sure to call `GIPSVideo_RenderUnLock` as soon as possible to avoid locking VideoEngine more than necessary.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_RenderUnlock [Carbon]

This function is called after `GIPSVideo_RenderLock` to let VideoEngine render video again.

Syntax

```
int GIPSVideo_RenderUnLock()
```

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddLocalRenderer [Cocoa]

This function sets the window in which the local video stream is rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(  GIPSCocoaRenderer* renderWindow,
                                 int zOrder,
                                 float left,
                                 float top,
                                 float right,
                                 float bottom)
```

Parameters

renderWindow	[in] Pointer to a GIPSCocoaRenderer.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`renderWindow` is a window reference to the window to output the video. The arguments `left, top, right`, and `bottom,` are the placement within the `WindowRef.` 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddRemoteRenderer [Cocoa]

This function sets the Window in which the remote video stream for a specific channel is rendered.

Syntax

```
int GIPSVideo_AddRemoteRenderer(   int channel,
                                   GIPSCocoaRenderer* remoteWindow,
                                   int zOrder,
                                   float left,
                                   float top,
                                   float right,
                                   float bottom)
```

Parameters

channel	[in] The channel ID number.

remoteWindow	[in] Pointer to a GIPSCocoaRenderer.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

`remoteWin` is a window reference to the window to output the video. The arguments `left, top, right`, and `bottom,` are the placement within the WindowRef. 0.0, 0.0 is upper left corner and 1.0, 1.0 is lower right corner.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

GIPSVideo_AddRemoteDemuxRender [Cocoa]

This function is user to demux one sub stream of an incoming “Brady Bunch” stream.

Syntax

```
int GIPSVideo_AddRemoteDemuxRender(    int channel,
                                       unsigned char streamID,
                                       GIPSCocoaRenderer* remoteWindow,
                                       int zOrder,
                                       float left,
                                       float top,
                                       float right,
                                       float bottom)
```

Parameters

channel	[in] The channel ID number.
streamID	[in] Stream index.
remoteWindow	[in] Pointer to a GIPSCocoaRenderer.
zOrder	[in] Z-order of video stream.
left	[in] Left border of video stream.

top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

The input arguments are similar to `GIPSVideo_AddRemoteRenderer,` with the addition of the id for the sub stream.

Requirements

Supported platforms	Max OSx
Header	Declared in GipsVideoEngineMac.h

Linux Specific Functions

This section describes the API functions using Linux specific arguments. These functions are defined in the GipsVideoEngineLinux.h interface file.

NOTE: GIPS Linux video renderer is designed for 24-bit TrueColor displays and the input Window is expected to have a 24-bit color map. Multiple video streams may be placed in the same window but should not overlap, as limitations of the Xlib draw functionality may make overlapping sections flicker.

GIPSVideo_AddLocalRenderer

This function sets the window in which the local video stream is rendered. `renderWin` is an X11 window reference to the window where the video is to be rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(  Window renderWin,
                                 float left = 0.0,
                                 float top = 0.0,
                                 float right = 1.0,
                                 float bottom = 1.0)
```

Parameters

renderWin	[in] Window handle.
left	[in] Left border of video stream.

top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

The arguments `left, top, right`, and `bottom,` are the placement within the `Window`. 0.0, 0.0 is the upper left corner and 1.0, 1.0 is the lower right corner.

Requirements

Supported platforms	Linux
Header	Declared in GipsVideoEngineLinux.h

GIPSVideo_AddRemoteRenderer

This function sets the window in which the remote video stream for a specific channel is rendered. `remoteWin` is an X11 window reference to the window where the video is to be rendered.

Syntax

```
int GIPSVideo_AddRemoteRenderer( int channel,
                                 Window remoteWin,
                                 float left = 0.0,
                                 float top = 0.0,
                                 float right = 1.0,
                                 float bottom = 1.0)
```

Parameters

channel	[in] The channel ID number.
renderWin	[in] Window handle.
left	[in] Left border of video stream.
top	[in] Top border of video stream.
right	[in] Right border of video stream.
bottom	[in] Bottom border of video stream.

Remarks

The arguments `left, top, right`, and `bottom,` are the placement within the `Window`. 0.0, 0.0 is the upper left corner and 1.0, 1.0 is the lower right corner.

Requirements

Supported platforms	Linux
Header	Declared in GipsVideoEngineLinux.h

GIPSVideo_RegisterXWindowsErrorHandlerCallback

This function registers a callback to get the error messages from XWindows.

Syntax

```
int GIPSVideo_RegisterXWindowsErrorHandlerCallback(
                                   GIPSXWindowsErrorHandler* obj)
```

Parameters

Obj	[in] Handle for error callback.

Requirements

Supported platforms	Linux
Header	Declared in GipsVideoEngineLinux.h

Windows Mobile Specific Functions

This section describes the API functions using Windows Mobile specific arguments. These functions are defined in the GipsVideoEngineWindowsCE.h interface file.

GIPSVideo_AddLocalRenderer

This function sets the window in which the local video stream is rendered. The parameter hWnd is a window reference to the window where the video is to be rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer(  HWND hWnd)
```

Parameters

hWnd	[in] The render window reference.

Requirements

Supported platforms	Windows Mobile
Header	Declared in GipsVideoEngineWindowsCE.h

GIPSVideo_AddRemoteRenderer

This function sets the window in which the local video stream is rendered. The parameter `hWnd` is a window reference to the window where the video is to be rendered.

Syntax

```
int GIPSVideo_AddLocalRenderer( int channel,
                                HWND hWnd)
```

Parameters

channel	[in] The number for the channel to be rendered in the specified window.
hWnd	[in] The render window reference.

Remarks

VideoEngine Mobile does not support multiple channels.

Requirements

Supported platforms	Windows Mobile
Header	Declared in GipsVideoEngineWindowsCE.h

Statistics

GIPSVideo_GetFrameStatistics

This function is called to get the statistics for the codecs in use.

Syntax

```
int GIPSVideo_GetFrameStatistics(int channel,
                                 GIPSVideo_FrameStatistics* inst)
```

Parameters

channel	[in] The channel ID number.
inst	[out] Statistics during the call.

Remarks

See `GIPSVideo_FrameStatistics` under Structures and Types.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetUpdate

This function gets the current incoming frame rate and bit rate.

Syntax

```
int GIPSVideo_GetUpdate(   int channel,
                           int* frameRate,
                           int* bitrate)
```

Parameters

channel	[in] The channel ID number.
frameRate	[out] Incoming frame rate.
bitrate	[out] Incoming bitrate.

Remarks

The bitrate is in bits per second and frameRate in frames per second.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetLocalUpdate

This function gets the current outgoing frame rate and bitrate.

Syntax

```
int GIPSVideo_GetLocalUpdate(    int channel,
                                 int* frameRate,
                                 int* bitrate)
```

Parameters

channel	[in] The channel ID number.

frameRate	[out] Send frame rate
bitrate	[out] Send bitrate

Remarks

The bitrate is in kilobits per second and frameRate is in frames per second.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Trace Settings

VideoEngine can generate trace information to facilitate debugging and troubleshooting.

GIPSVideo_SetTraceFileName

Sets the name of the trace file and enables non-encrypted trace messages.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
int GIPSVideo_SetTraceFileName(char* fileName);
```

Parameters

fileName	[in] Pointer to a zero-terminated character string which contains the name of the trace file.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError().`

Remarks

The type and amount of trace information is set by the `GIPSVideo_SetTraceFilter()` API. See the `GIPS_TraceFilter` enumerator for filter details.

The largest amount of non-encrypted trace information corresponds to the following filters:
`TR_STATE_INO | TR_WARNING | TR_ERROR | TR_CRITICAL | TR_APICALL, or TR_ALL.`

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetDebugTraceFileName

Sets the name of the debug trace file and enables encrypted (internal) trace messages.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
int GIPSVideo_SetDebugTraceFileName(char* fileName);
```

Parameters

fileName	[in] Pointer to a zero-terminated character string which contains the name of the trace file.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError()`.

Remarks

The type and amount of trace information is set by the `GIPSVideo_SetTraceFilter()` API. See the `GIPS_TraceFilter` enumerator for filter details.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetTraceFilter

Specifies the amount and type of trace information, which will be created by the GIPS VideoEngine.

NOTE: trace should only be enabled for debugging purposes. Some trace filters will result in a large amount of generated trace messages.

Syntax

```
int GIPSVideo_SetTraceFilter(unsigned int filter);
```

Parameters

filter	[in] Sets the filter type. See the `GIPS_TraceFilter` enumerator for filter details.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError().`

Remarks

Default filter type is `TR_ALL`.

To disable all traces, use the TR_NONE filter type.

Valid trace file names must have been set before this filter has any effect. See `GIPSVideo_SetTraceFileName()` and `GIPSVideo_SetDebugTraceFileName()` for details.

Requirements

Supported platforms	Windows, MAC OS X, Linux

GIPSVideo_SetTraceCallback

This function gets the trace information via a callback function.

Syntax

```
int GIPSVideo_SetTraceCallback(GIPSTraceCallbackFunction)
```

Parameters

GIPSTraceCallbackFunction	[in] "C" function pointer to a trace callback function.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartRTPDump

This function enables capturing of RTP packets to a binary file on a specific channel and for a given direction. The file can later be replayed using e.g. RTP Tools' `rtpplay` since the binary file format is compatible with the `rtpdump` format.

NOTE: It is recommended that you use this API for debugging purposes only since the created files can become very large.

Syntax

```
int GIPSVideo_StartRTPDump(int channel, const char* fileNameUTF8, bool
inPacket);
```

Parameters

channel	[in] The channel ID number.
fileNameUTF8	[in] A pointer to an array containing the name of the file as a null-terminated and UTF-8 encoded string.
inPacket	[in] True for incoming packet, false for outgoing packet.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError().`

Remarks

It is possible to enable this functionality before any RTP media is received or transmitted. As soon as the RTP session starts, packets will be stored in the already opened file.

This API allows the user to capture RTP sessions without using an external tool like Wireshark (http://www.wireshark.org/).

Both RTP and RTCP packets are captured in both directions.

If RTP dump is activated on the incoming side, the packets are captured *after* decryption (e.g. SRTP).

If RTP dump is activated on the outgoing side, the packets are captured *before* encryption (e.g. SRTP).

See http://www.cs.columbia.edu/irt/software/rtptools/ for details on how to use the command-line tool `rtpplay`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopRTPDump

This function disables capturing of RTP packets to a binary file on a specific channel and for a given direction.

Syntax

```
int GIPSVideo_StopRTPDump(int channel, bool inPacket);
```

Parameters

channel	[in] The channel ID number.

inPacket	[in] True for incoming packet, false for outgoing packet.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError().`

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_RTPDumpIsActive

This function retrieves the current RTP capturing state for the specified channel and direction.

Syntax

```
int GIPSVideo_RTPDumpIsActive(int channel, bool inPacket);
```

Parameters

channel	[in] The channel ID number.
inPacket	[in] True for incoming packet, false for outgoing packet.

Return Values

The return value is `0` if RTP dump is disabled and `1` if RTP dump is enabled. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Registering Callbacks

GIPSVideo_SetCallback

This function registers the generic callback functions.

Syntax

```
int GIPSVideo_SetCallback(GIPSVideoCallback * obj);
```

Parameters

obj	[in] Callback object.

Example Code

The example shows the usage of GIPSVideo_SetCallback() with GIPSVideo_EnableBrightnessAlarm() function. The code assumes that VideoEngine has been initialized and the camera is running.

```
class MyGIPSVideoCallback: public GIPSVideoCallback
{
    virtual void PerformanceAlarm(int value) {};
    virtual void LocalFrameRate(int frameRate) {};
    virtual void MotionUpdate(unsigned char value) {};
    virtual void NoPictureAlarm(bool ) {}
    virtual void BrightnessAlarm(int value)
    {
        if (value == 1)
            printf("\n\t\tBrightnessAlarm - dark image.\n");
        else if (value == 2)
            printf("\n\t\tBrightnessAlarm - light image.\n");
    };
};

void main()
{

    MyGIPSVideoCallback* callback = new MyGIPSVideoCallback();

    _ptrViE->GIPSVideo_SetCallback(callback);
    _ptrViE->GIPSVideo_EnableBrightnessAlarm(true);

    _ptrViE->GIPSVideo_SetCallback(NULL);
    _ptrViE->GIPSVideo_EnableBrightnessAlarm(false);
    delete callback;

}
```

Remarks

See GIPSVideoCallback for details on the callback events.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableBrightnessAlarm

This function enables or disables the `BrightnessAlarm` callback in the `GIPSVideoCallback` class.

Syntax

```
int GIPSVideo_EnableBrightnessAlarm(bool enable)
```

Parameters

enable	[in] Set to true to enable brightness alarm.

Remarks

See GIPSVideoCallback for details on the callback events.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableMotionUpdate

This function enables or disables the `MotionUpdate` callback in the `GIPSVideoCallback` class.

Syntax

```
int GIPSVideo_EnableMotionUpdate(bool enable)
```

Parameters

enable	[in] Set to true to enable motion update callback and false to disable the motion update.

Remarks

See GIPSVideoCallback for details on the callback events.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetChannelcallback

This function registers the channel callback functions which are defined in the GIPSVideoChannelCallback class.

Syntax

```
int GIPSVideo_SetChannelCallback( int channel, GIPSVideoChannelCallback *
obj);
```

Parameters

channel	[in] The channel ID number.
obj	[in] Callback object.

Remarks

See GIPSVideoCallback for details on the callback events.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_RegisterRenderCallback

This function registers the `GIPSEffectFilter` class to VideoEngine to enable adding effects to the incoming video streams before the frame is sent to the render.

Syntax

```
int GIPSVideo_RegisterRenderCallback(int channel, GIPSEffectFilter* obj)
```

Parameters

channel	[in] The channel ID for the render callback.
obj	[in] Callback object.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_RegisterSendCallback

This function registers the `GIPSEffectFilter` class to VideoEngine to enable adding effects to the sending stream before encoding the stream.

NOTE: For future use, not implemented yet.

Syntax

```
int GIPSVideo_RegisterSendCallback(int channel, GIPSEffectFilter* obj)
```

Parameters

channel	[in] The channel ID for the render callback
obj	[in] Callback object

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_RegisterCaptureCallback

This function registers the `GIPSEffectFilter` class to VideoEngine to enable adding effects to the incoming camera frame.

Syntax

```
int GIPSVideo_RegisterCaptureCallback(GIPSEffectFilter* obj)
```

Parameters

obj	[in] Callback object.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableKeyFrameRequestCallback

This function enables or disables the RequestNewKeyFrame callback in the GIPSVideoChannelCallback class. The RequestNewKeyFrame callback is called when VideoEngine is not in sending mode or RTCP not enabled. With this API, the callback will be issued all the time.

Syntax

```
int GIPSVideo_EnableKeyFrameRequestCallback(bool enable)
```

Parameters

enable	[in] Set to true to enable `RequestNewKeyFrame` callback and false to the default behavior.

Remarks

See GIPSVideoChannelCallback for details on the callback events.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Version Handling

GIPSVideo_GetVersion

This function copies the version into the given buffer with length `buflen`.

Syntax

```
int GIPSVideo_GetVersion(char *version, int buflen)
```

Parameters

version	[out] Char vector where the version information is copied to.
buflen	[in] Size in bytes of version vector.

Return Values

The function returns 0 if the copy was performed successfully and –1 otherwise.

Remarks

A buffer size of 1024 characters is sufficient for this call.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

File Functions

The file functions supports 2 formats, the standard AVI file format and a GIPS media file format. A file recorded in the AVI format can be played in most available media players depending on the codec supported in the media player. The GIPS media file format can only be handled by the GIPS Video Engines; however the function `GIPSVideo_ConvertGMFToAVI` can be used to convert from the GIPS media file format to the AVI file format. The benefit of the GIPS media file format is that it requires extremely low CPU usage at the time of the recording since it does not require transcoding the incoming stream.

The AVI file format is default, to use the GIPS file format replace `FILE_FORMAT_AVI_FILE` with `FILE_FORMAT_PREENCODED_FILE`.

NOTE: The available codecs for the AVI file encoding are I420 and MPEG-4 currently.

GIPSVideo_StartPlayFile

This function starts playing a file on a channel.

Syntax

```
int GIPSVideo_StartPlayFile(    int channel,
                                const char * fileName,
                                bool loop = false,
                                int file_format = FILE_FORMAT_AVI_FILE,
                                bool tryToSendPreEncoded = false)
```

Parameters

channel	[in] The channel ID number.
filename	[in] Name of the file to play.
loop	[in] Play the file over and over again.
file_format	[in] Format of the file.
tryToSendPreEncoded	[in] true if data is already encoded, otherwise false.

Remarks

The sending codec must match to the codec used during recording of the file. Application would need to call `GIPSVideo_SetSendCodec()` with correct height and width before you play the file out.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_IsPlayingFile

This function returns true if we currently are playing a file on this channel.

Syntax

```
bool GIPSVideo_IsPlayingFile(int channel)
```

Parameters

channel	[in] The channel ID number.

Return Values

This function returns true if any file is playing on the channel.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopPlayingFile

This function stops playing a file on a channel.

Syntax

```
int GIPSVideo_StopPlayingFile(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartPlayFileAsCamera

This function starts playing a file as a camera.

Syntax

```
int GIPSVideo_StartPlayFileAsCamera(const char * fileName,
                                    bool loop = false,
                                   int file_format = FILE_FORMAT_AVI_FILE)
```

Parameters

filename	[in] File name.
loop	[in] Play the file in an infinite loop.
file_format	[in] Format of the file.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_IsPlayingFileAsCamera

This function returns true if we currently are playing a file instead of the camera

Syntax

```
bool GIPSVideo_IsPlayingFileAsCamera()
```

Return Values

This function returns true if any file is playing as camera.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopPlayingAsCamera

This function stops playing the file as a camera.

Syntax

```
int GIPSVideo_StopPlayingFileAsCamera()
```

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartRecording

This function starts recording the incoming channel to file using the encoder specified in `inst`.

Syntax

```
int GIPSVideo_StartRecording(    int channel,
                                 const char* fileName,
                                 GIPSVideo_CodecInst* inst = NULL,
                                 int file_format = FILE_FORMAT_AVI_FILE,
                                 bool wide_band = true)
```

Parameters

channel	[in] The channel ID number.
fileName	[in] File name.
inst	[in] Codec to use for the file encoding.
file_format	[in] Format of the file.
wide_band	[in] Record the audio in wide band (16KHz).

NOTE: The channel in use must not use the default send codec. Refer to the def parameter in GIPSVideo_SetSendCodec() api. The def parameter should be set to false when using this api. For AVI, currently it is possible to record using I420 or MPEG-4 codecs only. You can set the `file_format` to `FILE_FORMAT_PREENCODED_FILE` to record using any other codec. This will output the incoming compressed stream directly to file.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopRecording

This function stops recording the incoming channel.

Syntax

```
int GIPSVideo_StopRecording(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartRecordingCamera

This function starts recording the stream from the camera to the file filename using encoder specified in `codec_inst`.

Syntax

```
int GIPSVideo_StartRecordingCamera(
                          const char* fileName,
                          GIPSVideo_CodecInst *codec_inst,
                          bool wide_band = true)
```

Parameters

fileName	[in] File name.
codec_inst	[in] Codec to use for the file encoding.
wide_band	[in] Record the audio in wide band (16KHz).

NOTE: The API only supports recording to avi file format currently. You must use I420 or MPEG-4 to record to AVI files.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopRecordingCamera

This function stops recording the local video from the camera.

Syntax

```
int GIPSVideo_StopRecordingCamera()
```

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StartRecording

This function starts recording the outgoing stream to a file `fileName` using the encoder specified in `inst`.

Syntax

```
int GIPSVideo_StartRecording(    const char* fileName,
                                 GIPSVideo_CodecInst* inst = NULL,
                                 int file_format = FILE_FORMAT_AVI_FILE,
                                 bool wide_band = true)
```

Parameters

fileName	[in] File name.
inst	[in] Codec to use for the file encoding.
file_format	[in] Format of the file.
wide_band	[in] Record the audio in wide band (16KHz).

NOTE: For AVI, currently it is possible to record using I420 or MPEG-4 codecs only. You can set the`file_format` to `FILE_FORMAT_PREENCODED_FILE` to record using any other codec.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_StopRecording

This function stops recording the outgoing stream.

Syntax

```
int GIPSVideo_StopRecording()
```

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetFileInfo

This function returns the duration and codec information of a specified file.

Syntax

```
int GIPSVideo_GetFileInfo (const char* fileName,
                                  int& durationMs,
                                 int file_format,
                                 GIPSVideo_CodecInst *codecInst = NULL);
```

Parameters

fileName	[in] File name.
durationMs	[out] The duration of the file (in milliseconds).
file_format	[in] Format of the file.
codecInst	[out] Codec information of the recorded file.

Return Values

The return value is `0` if the function succeeds. If the function fails, the return value is `-1` and a specific error code can be retrieved by calling `GIPSVideo_GetLastError()`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

File Conversion

This section addresses functions used to convert files from one format to another.

GIPSVideo_ConvertGMFToAVI

This function converts a GIPS media file to AVI file format.

Syntax

```
int GIPSVideo_ConvertGMFToAVI(
                          const char* fileNameGMF,
                          const char* fileNameAVI,
                          GIPSVideo_CodecInst *codec_inst)
```

Parameters

fileNameGMF	[in] File name of GIPS Media file.
fileNameAVI	[out] File name of AVI file.
codec_inst	[in] Codec used during recording of fileNameGMF.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Snapshot (JPEG Files)

This section handles JPEG images

GIPSVideo_GetSnapshotCamera

This function gets a JPEG Snapshot of the current camera image.

Syntax

```
int GIPSVideo_GetSnapshotCamera(const char* fileName)
```

Parameters

fileName	[in] name of the JPEG file where the image is stored.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSnapshot

This function gets a JPEG Snapshot of the current incoming image on a channel.

Syntax

```
int GIPSVideo_GetSnapshot(int channel, const char* fileName)
```

Parameters

channel	[in] The channel ID number.

filename	[in] Name of the JPEG file where the image is stored.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Snapshot (GIPSVideo_Picture)

This section handles snapshot taken using GIPSVideo_Picture.

GIPSVideo_GetSnapshotCamera

This function gets a GIPSVideo_Picture Snapshot of the current camera image.

Syntax

```
int GIPSVideo_GetSnapshotCamera(GIPSVideo_Picture& picture)
```

Parameters

picture	[in] Reference to the structure to store the snapshot in.

Remarks

The `data` field in `picture` must be a `NULL` pointer and `size` must be zero. `type` must be set to the video format the snapshot shall be stored in. Supported video formats are `GIPSVideo_I420` and `GIPSVideo_RGB24`. `GIPSVideo_GetSnapshot` will allocate the required memory. To free the memory, call `GIPSVideo_FreeImage`.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetSnapshot

This function gets a `GIPSVideo_Picture` Snapshot of the current incoming image on a channel.

Syntax

```
int GIPSVideo_GetSnapshot(int channel, GIPSVideo_Picture& picture)
```

Parameters

channel	[in] The channel ID number.
picture	[in/out] Reference to the structure to store the snapshot in.

Remarks

The `data` field in `picture` must be a `NULL` pointer and `size` must be zero. `type` must be set to the video format the snapshot shall be stored in. Supported video formats are `GIPSVideo_I420` and `GIPSVideo_RGB24`. `GIPSVideo_GetSnapshot` will allocate the required memory. To free the memory, call `GIPSVideo_FreeImage`.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_FreePicture

This function frees the memory allocated by `GIPSVideo_GetSnapshot` and `GIPSVideo_GetSnapshotCamera`.

Syntax

```
int GIPSVideo_FreePicture(GIPSVideo_Picture& picture)
```

Parameters

picture	[in] Pointer to the structure to free.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Background

GIPSVideo_SetBackgroundImage (JPEG Files)

This function sets a JPEG image as the background image.

Syntax

```
int GIPSVideo_SetBackgroundImage(int channel,
                                 const char* fileName,
                                 unsigned int timeInMS = 0)
```

Parameters

channel	[in] The channel ID number, use -1 for local renderer.
filename	[in] Name of the JPEG file.

timeInMS	[in] Time in milliseconds before this image is started to render.

Remarks

If `timeInMS` is larger than zero, the image will be displayed for that channel when there has been no decoded packet for `timeInMS` milliseconds.

For local renderer, `timeInMS` must be 0 milliseconds or equal to or higher than 1000 milliseconds.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetBackgroundImage (GIPSVideo_Picture)

This function sets a `GIPSVideo_Picture` image as the background image.

Syntax

```
int GIPSVideo_SetBackgroundImage(int channel,
                                 GIPSVideo_Picture& picture,
                                 unsigned int timeInMS = 0)
```

Parameters

channel	[in] The channel ID number, use -1 for local renderer.
picture	[in] Pointer to the GIPSVideo_Picture structure.
timeInMS	[in] Time in milliseconds before this image is started to render.

Remarks

If `timeInMS` is larger than zero, the image will be displayed for that channel when there has been no decoded packet for `timeInMS` milliseconds.

For local renderer, `timeInMS` must be 0 milliseconds or equal to or higher than 1000 milliseconds.

Supported video formats are `GIPSVideo_I420` and `GIPSVideo_RGB24`.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Conferencing (Multi-party Calling)

This section describes the functions used for setting up and controlling a multi-party call.

GIPSVideo_Conferencing

This function adds or removes a channel from conference.

Syntax

```
int GIPSVideo_Conferencing(int channel, bool enable);
```

Parameters

channel	[in] The channel ID number.
enable	[in] True adds channel to conferencing.

Remarks

The conference mix is created as a "Brady bunch", hence the video window is divided into 4 quadrants with one channels rendered in a quadrant.

In quadrant 0 we put our own image. In quadrant 1 we put the lowest channel which has conferencing enabled. In quadrant 2 we put the second lowest channel that has conferencing enable. In quadrant 3 we put the third lowest channel that has conferencing enable.

NOTE: In this version, incoming channels must have the same size or 1/4 of the size to be added to the conference. I.e. if conference is 320*240 incoming channels can be of size 320*240 or 160*120.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_ConferenceDemuxing

This function enables demuxing of the input for one channel.

Syntax

```
int GIPSVideo_ConferenceDemuxing(int channel, bool enable);
```

Parameters

channel	[in] The channel ID number.
enable	[in] True enables demuxing.

Remarks

The incoming stream is assumed to be a conference mix of four different streams. The four quadrants of the incoming conference mix will be separated into four separate streams, which will be sent to the render separately.

NOTE: This function is not required for DirectDraw on Windows.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetRenderStreamID

This function gets the stream identifier given the source identifier

Syntax

```
int GIPSVideo_GetRenderStreamID(int channel, unsigned int csrc);
```

Parameters

channel	[in] The channel ID number.
csrc	[in] Source identifier for a stream.

Return Values

The return value is the id of the stream with the specified CSRC for a certain channel. For example the render stream id used when rendering a conference mixed video stream. The value of csrc in a mixed stream can be obtained using `GIPSVideoChannelCallback` function `IncomingCSRCChanged.`

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetLayout

This function sets a video layout to be used when mixing several channels into one stream

Syntax

```
int GIPSVideo_SetLayout(GIPSVideoLayouts layout);
```

Parameters

layout	[in] Chosen video layout.

Remarks

When in a conference there are several layouts to choose between. `GIPS_LAYOUT_DEFAULT` use black fields when the number of channels doesn't add up to a natural layout such as a 4 participant conference. `GIPS_LAYOUT_ADVANCED1` to `GIPS_LAYOUT_ADVANCED4` stretch and cut the incoming images in different ways to minimize the black areas of the mixed signal. `GIPS_LAYOUT_ADVANCED1` to `GIPS_LAYOUT_ADVANCED4` cannot be used for demuxing since all fields have different size and layout.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

RTCP Functions

RTCP enables automatic bandwidth management of the individual video streams and provides statistics about the session.

GIPSVideo_EnableIntraRequest

This function enables key frame request signaling according to RFC 2032. RTCP packets over the RTP channel are used.

Syntax

```
int GIPSVideo_EnableIntraRequest(int channel, bool enable)
```

Parameters

channel	[in] The channel ID number.
Enable	[in] On/Off.

Remarks

Deprecated by RFC 4585. Use `GIPSVideo_EnablePLI` when writing a new application.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableRTCP

This function enables the transmission of RTCP sender report (See RFC 3550 for details).

Syntax

```
enum GIPSRTCPMode
{
   GIPS_RTCP_NONE = 0,
   GIPS_RTCP_COMPOUND_RFC4585 = 1,
   GIPS_RTCP_NON_COMPOUND_RFC5506 =2
}
int GIPSVideo_EnableRTCP(int channel, GIPSRTCPMode rtcpMode)
```

Parameters

channel	[in] The channel ID number.
rtcpMode	[in] RTCP mode. Can be `none`, `compound,` described in RFC 4585 or `none compound` described in RFC 5506.

Remarks

RTCP is enabled by default and the default mode is `compound`.

RTCP BYE is send automatically when `GIPSVideo_StopSend` is called.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_RTCPStat

This function retrieves the RTCP statistics computed by the GIPS VideoEngine. The statistics are computed according to RFC 3550.

Syntax

```
int GIPSVideo_RTCPStat(    int channel,
                           GIPSVideo_CallStatistics * incomingStats,
                           GIPSVideo_CallStatistics * outgoingStats)
```

Parameters

channel	[in] The channel ID number.
incomingStats	[out] Incoming stream call statistics.
outgoingStats	[out] Outgoing stream call statistics.

Return Values

The function returns 0 if the information was retrieved successfully and -1 otherwise.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetRTCPCNAME

This function sets the CNAME parameter for RTCP reports. Default name is user1@undefined.

Syntax

```
int GIPSVideo_SetRTCPCNAME(int channel, char  str[256])
```

Parameters

channel	[in] The channel ID number.
str	[in] Name of the local party.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetRemoteRTCPCNAME

This function retrieves the CNAME parameter from the remote party RTCP report.

Syntax

```
int GIPSVideo_GetRemoteRTCPCNAME(int channel, char str[256])
```

Parameters

channel	[in] The channel ID number.
str	[out] Name of the remote party.

Return Values

If no RTCP report has been received, or if the report does not contain any CNAME field, this will be an empty string. The returned string can be up to 255 bytes long.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableNACK

This function enables NACK (Negative ACKnowledgement) using RTCP, implemented based on RFC4585. NACK retransmits RTP packets if lost by the network; this creates a lossless transport at the expense of delay.

Syntax

```
int GIPSVideo_EnableNACK(int channel, bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] On/Off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableFEC

This function enables forward error correction (FEC), which improves packet loss robustness. Enabling will cause protecting FEC packets to be sent along with the usual media packets. The FEC packets are used at the receive side in an attempt to recover any lost media packets. The implementation is based on RFC 5109.

Syntax

```
int GIPSVideo_EnableFEC(   int channel,
                           bool enable,
                           unsigned char payloadTypeRED,
                           unsigned char payloadTypeFEC
                          )
```

Parameters

channel	[in] The channel ID number.
enable	[in] On/Off.
payloadTypeRED	[in] RTP payload number used for redundant (or RED) data which will be applied to the entire stream.
payloadTypeFEC	[in] RTP payload number used to identify FEC packets.

Remarks

NACK and FEC both should not be enabled on the same channel.

The receiver must have registered corresponding RED and FEC payload types. With VideoEngine, this can be done by calling `GIPSVideo_SetReceiveCodec` with `codec_inst.plname` set to `“red”` and `“ULPFEC”` respectively and `codec_inst.pltype` set appropriately.

Currently, this standard FEC is only supported when using H.264 AVC. LSVX has a proprietary FEC which is not enabled by this function, but by specifying `GIPS_LSVX_FEC` in the `codec_inst.codecSpecific` field. Refer to the `GIPSVideoSignalingLSVX` enum.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnablePLI

This function enables PLI (picture loss indication) using RTCP, implemented based on RFC4585. If the decoder enters a state from which it can’t recover it sends a PLI to the sender requesting a new key frame to get a new start point.

Syntax

```
int GIPSVideo_EnablePLI(int channel,  bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] On/Off.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableTMMBR

This function enables signaling of temporary bitrate constraints using RTCP, implemented based on RFC4585.

Syntax

```
int GIPSVideo_EnableTMMBR(int channel, bool enable)
```

Parameters

channel	[in] The channel ID number.
enable	[in] On/Off.

Requirements

Supported platforms	Windows, MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetAvailableBandwidth

This function returns the estimated maximum capacity of a channel. The estimate is only returned if a maximum available bitrate has been found during an ongoing call.

Syntax

```
GIPSVideo_GetAvailableBandwidth(int channel, int& bandwidth)
```

Parameters

channel	[in] The channel ID number.
bandwidth	[out] The estimated available bandwidth.

Return Values

If no maximum bandwidth has been found during an ongoing call the function returns -1.The function returns 0 on success.

Requirements

Supported platforms	Windows(incl Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_SetRTPKeepaliveStatus

This function enables or disables an RTP keepalive mechanism which can be used to maintain an existing Network Address Translator (NAT) mapping while regular RTP is no longer transmitted.

NOTE: See Section 4.6 (RTP Packet with Unknown Payload Type) at http://www.ietf.org/id/draft-ietf-avt-app-rtp-keepalive-07.txt for more details.

Syntax

```
GIPSVideo_SetRTPKeepaliveStatus(int channel,
                                bool enable,
                                int unknownPayloadType,
                                int deltaTransmitTimeSeconds = 15)
```

Parameters

channel	[in] the channel ID number**.**
enable	[in] If this parameter is true, RTP keepalive is enabled. If false, RTP keepalive is disabled.
unknownPayloadType	[in_opt] Dynamic payload type that has not been negotiated by the peers (e.g. not negotiated within the SDP offer/answer). Valid input range is [0,127].
deltaTransmitTimeSeconds	[in_opt] Specifies the time, in seconds between two successive RTP keepalive packets. Default valu is 15 seconds. Valid input rangeis [1,60] seconds.

Return Values

The return valis is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

Remarks

RTP keepalive packets are all of length 0 (contains no RTP payload).

RTP keepalice packets will only be transmitted when *all* of the following conditions are met:

1. RTP keepalive is enabled.
2. The ViE is in a listening state.
3. A destination address is defined using the GIPSVideo_SetSendDestination() API.
4. The VE is not sending or is in an on-hold state.
5. Regular RTP packets are not transmitted.

GIPS VideoEngine will silently discard incoming RTP keepalive packets.

Requirements

Supported platforms	Windows(incl Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_GetRTPKeepaliveStatus

This function returns the RTP keepalive status for a specific channel.

Syntax

```
GIPSVideo_GetRTPKeepaliveStatus(int channel,
                                Bool& enabled,
                                Int& unknownPayloadType,
                                Int& deltaTransmitTimeSeconds)
```

Parameters

channel	[in] the channel ID number**.**
enabled	[out] A binary reference output which is set to true if RTP keepalive is enabled and false otherwise
unknownPayloadType	[opt] Contains the dynampic payload type as output
deltaTransmitTimeSeconds	[opt] Contains the delta time, in seconds, between transmission of two successive RTP keepalive packets as output.

Return Values

The return valis is 0 if the function succeeds. If the function fails, the return value is -1 and a specific error code can be retrieved by calling GIPSVideo_GetLastError().

Remarks

A returned status of true, does not guarantee that RTP keepalive packets are actually being transmitted. All conditions given above (see GIPSVideo_SetRTPKeepaliveStatus()) must be fulfilled before transmission stats.

Requirements

Supported platforms	Windows(incl Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Secure RTP

NOTE: This is an optional feature. It relies on a GIPS product that may not be included in your VideoEngine configuration.

VideoEngine can be delivered with a reference implementation of Secure RTP (SRTP) using the open source libSRTP available at http://srtp.sourceforge.net/srtp.html. The following functions control SRTP settings. A brief description of the functionality is given here; for complete information please refer to the libSRTP webpage and RFC 3711.

There are two types of protection, encryption and authentication. Both, one or none of them can be used, creating four combined types of protection. *It is strongly recommended to use both.* Encryption is done on the payload; authentication is applied to header and payload.

There are two kinds of ciphers (encryption algorithms) available, AES 128 counter mode and null. There are two kinds of authentication available, HMAC-SHA1 and null. Null cipher and null authentication provide no protection and are available for compliance with RFC 3711.

A session encryption key, a session authentication key and a session salt key are derived from a master key and master salt. The master key is assumed to be 128 bits (16 bytes) and the master salt is assumed to be 112 bits (14 bytes) for the supported cipher and authentication types. The key input (`key`) to the API calls below is a pointer to a buffer that contains both the master key (first 128 bits) and master salt (the following 112 bits) and is consequently assumed to be 240 bits (30 bytes). For information on master key handling, please refer to the libSRTP webpage.

For AES 128 CM, the cipher key length (`cipher_key_len`) is the session encryption key length plus session salt key length. The session encryption key length is always 128 bits (16 bytes). For null cipher, the cipher key length is the cipher key length.

Authentication key length (`auth_key_len`) is the session authentication key length. Authentication tag length is the length of the authentication tag added to the packet.

NOTE: In the current implementation, cipher key length input determines the number of bytes to copy from the buffer pointed to by `key`. This means that the cipher key length must always be at least 30 bytes, otherwise the master key/salt will be truncated.

The two kinds of ciphers (`cipher_type`):

```
#define CIPHER_NULL 0
#define CIPHER_AES_128_COUNTER_MODE 1
```

The two types of authentication (`auth_type`):

```
#define AUTH_NULL 0
#define AUTH_HMAC_SHA1 3
```

The four combined types of protection (`security`):

```
#define NO_PROTECTION 0
#define ENCRYPTION 1
#define AUTHENTICATION 2
#define ENCRYPTION_AND_AUTHENTICATION 3
```

Recommended parameters:

Desired protection	Both (recommended)	Encryption only	Authentication only
Cipher type	AES 128 CM	AES 128 CM	NULL
Cipher length	30	30	30
Authentication type	HMAC-SHA1	NULL	HMAC-SHA1
Authentication key length	20	0	20
Authentication tag length	4 or 10[1]	0	4 or 10[1]
Protection (`security`)	Encryption and auth	Encryption	Authentication

Valid parameter values:

Cipher length	30 - 256
Authentication key length	0 – 20 (HMAC-SHA1) / 0 – 256 (NULL)
Authentication tag length	0 – 20 (HMAC-SHA1) / 0 – 12 (NULL)

GIPSVideo_EnableSRTPSend

This function enables SRTP on the transmitted data for a given channel. The key parameter has all the master key and salt for both cipher and authentication. For AES128 you have 16 bytes key and 14 bytes salt.

Syntax

```
int GIPSVideo_EnableSRTPSend(    int channel,
                                 int cipher_type,
                                 int cipher_key_len,
                                 int auth_type,
                                 int auth_key_len,
                                 int auth_tag_len,
                                 int security,
                                 const unsigned char* key,
                                  bool applyToRTCP)
```

Parameters

channel	[in] The channel ID number.
cipher_type	[in] Cipher type.
cipher_key_len	[in] Cipher key length plus salt key length in bytes. Must be at least 30.
auth_type	[in] Authentication type.
auth_key_len	[in] Authentication key length in bytes.
auth_tag_len	[in] Authentication tag length in bytes.
security	[in] Security type.
key	[in] Master key and master salt.
applyToRTCP	[in] Encrypt RTPCP as well as RTP packets.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_DisableSRTPSend

This function disables SRTP on the transmitted data for a given channel.

Syntax

```
int GIPSVideo_DisableSRTPSend(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableSRTPReceive

This function enables SRTP on the received data for a given channel.

Syntax

```
int GIPSVideo_EnableSRTPReceive(       int channel,
                                       int cipher_type,
                                       int cipher_key_len,
                                       int auth_type,
                                       int auth_key_len,
                                       int auth_tag_len,
                                       int security,
                                       const unsigned char* key,
                                        bool applyToRTCP)
```

Parameters

channel	[in] The channel ID number.
cipher_type	[in] Cipher type.
cipher_key_len	[in] Cipher key length plus salt key length in bytes. Must be at least 30.
auth_type	[in] Authentication type.
auth_key_len	[in] Authentication key length in bytes.
auth_tag_len	[in] Authentication tag length in bytes.
security	[in] Security type.
key	[in] Master key and master salt.
applyToRTCP	[in] decrypt RTPCP as well as RTP packets.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideoDisableSRTPReceive

This function disables SRTP on the received data for a given channel.

Syntax

```
int GIPSVideo_DisableSRTPReceive(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

External Encryption

VideoEngine provides an interface that enables the application writer to add their own encryption scheme to the RTP stream. What needs to be done is to implement a C++ class that overloads the `GIPS_encryption` class defined in the API header file:

Class GIPS_encryption

Syntax

```
class GIPS_encryption
{
public:
    virtual void encrypt(  int channel_no,
                           unsigned char * in_data,
                           unsigned char * out_data,
                           int bytes_in,
                           int *bytes_out) = 0;

    virtual void decrypt(  int channel_no,
                           unsigned char * in_data,
                           unsigned char *out_data,
                           int bytes_in,
                           int *bytes_out) = 0;

    virtual void encrypt_rtcp(   int channel_no,
                                 unsigned char *in_data,
                                 unsigned char * out_data,
                                 int bytes_in,
                                 int * bytes_out) = 0;

    virtual void decrypt_rtcp(   int channel_no,
                                 unsigned char * in_data,
                                 unsigned char *out_data,
```

Global IP Solutions

```
                                  int bytes_in,
                                  int *bytes_out) = 0;
};
```

The `encrypt` call will be called for every RTP packet that is sent with the whole RTP packet as argument `in_data` including the RTP header. If the RTP header should not be encrypted the first 12 bytes should be left un-touched. For every call the function must set the value pointed to by `bytes_out` to the number of bytes to be sent. If the packet does not change its size by the encryption `bytes_out` should be set to `bytes_in`.

The same methodology is used for the `decrypt` call that will be called for every incoming RTP packet. Besides implementing the encryption class the following calls are used to enable the encryption:

`encrypt_rtcp` and `decrypt_rtcp` are the corresponding calls for RTCP packets.

GIPSVideo_InitEncryption

This function initializes VideoEninge with the encryption object that should be used for the encryption.

Syntax

```
int GIPSVideo_InitEncryption (GIPS_encryption *encr_obj)
```

Parameters

encr_obj	[in] External encryption object

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_EnableEncryption

This function turns on the encryption and the decryption for the specified channel.

Syntax

```
int GIPSVideo_EnableEncryption(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

GIPSVideo_DisableEncryption

This function turns off the encryption and the decryption for the specified channel.

Syntax

```
int GIPSVideo_DisableEncryption(int channel)
```

Parameters

channel	[in] The channel ID number.

Requirements

Supported platforms	Windows (incl. Mobile), MAC OS X, Linux
Header	Declared in GipsVideoEngine.h

Chapter 4: Example Code

The code below creates a small video autoTest program which will start the camera and do local preview for approximately 10 seconds and then send H264 encoded stream in loopback for 10 seconds. It is a windows console test application showing usage of h264 codec only.

```
#include <stdio.h>
#include "GipsVideoEngineWindows.h"
#include "GipsVideoEngine.h"
#define SLEEP_10_SEC ::Sleep(10000)
#define GET_TIME_IN_MS timeGetTime

LRESULT CALLBACK GIPSWinProc( HWND hWnd,UINT uMsg,WPARAM wParam,LPARAM
lParam)
{
    switch(uMsg)
    {
    case WM_DESTROY:
          break;
    case WM_COMMAND:
          break;
    }
    return  DefWindowProc(hWnd,uMsg,wParam,lParam);
}

int GIPS_CreateWindow(HWND &hwndMain,int winNum, int width, int height)
{
    HINSTANCE hinst = GetModuleHandle(0);
    WNDCLASSEX wcx ;
    wcx.hInstance     = hinst;
    wcx.lpszClassName = "GIPSAutoTest" ;
    wcx.lpfnWndProc   = (WNDPROC)GIPSWinProc;
    wcx.style         = CS_DBLCLKS;
    wcx.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
    wcx.hIconSm       = LoadIcon (NULL, IDI_APPLICATION);
    wcx.hCursor       = LoadCursor (NULL, IDC_ARROW);
    wcx.lpszMenuName  = NULL;
    wcx.cbSize        = sizeof (WNDCLASSEX);
    wcx.cbClsExtra    = 0;
    wcx.cbWndExtra    = 0;
    wcx.hbrBackground = GetSysColorBrush(COLOR_3DFACE) ;

    // Register our window class with the operating system.
    // If there is an error, exit program.
    if ( !RegisterClassEx (&wcx) )
    {
          MessageBox( 0, TEXT("Failed to register window
          class!"),TEXT("Error!"), MB_OK|MB_ICONERROR ) ;
```

```
            return 0;
      }

      // Create the main window.
      hwndMain = CreateWindowEx(
            0,                        // no extended styles
            "GIPSAutoTest",             // class name
            "GIPSAutoTest Window",          // window name
            WS_OVERLAPPED |WS_THICKFRAME,   // overlapped window
            800,                      // horizontal position
            0,                        // vertical position
            width,                    // width
            height,                   // height
            (HWND) NULL,              // no parent or owner window
            (HMENU) NULL,             // class menu used
            hinst,                    // instance handle
            NULL);                    // no window creation data

      if (!hwndMain)
      {
            int error = GetLastError();
            return -1;
      }

      // Show the window using the flag specified by the program
      // that started the application, and send the application
      // a WM_PAINT message.

      ShowWindow(hwndMain, SW_SHOWDEFAULT);
      UpdateWindow(hwndMain);
      return 0;
}

int main()
{
      printf("\n\n\n\nStarting GIPS video test\n");

      int err = 0;

      GipsVideoEngineWindows* _ptrViE = &GetGipsVideoEngine();


      _ptrViE->GIPSVideo_SetDebugTraceFileName("debugTrace.txt");
      _ptrViE->GIPSVideo_SetTraceFileName("trace.txt");

      err |= _ptrViE->GIPSVideo_Init((GIPSVoiceEngine*)NULL, 0, 0, 0);

      if (err)
      {
            int error = _ptrViE->GIPSVideo_GetLastError();
            printf("GIPSVideo_Init error:%d",error);
            SLEEP_10_SEC;
            return -1;
      }
```

```
int channel1 = _ptrViE->GIPSVideo_CreateChannel();
printf("\tCreated channel: %d\n", channel1);

ptrinf("\nEnumerating cameras, make sure that all connected are
listed\n");


char str[64];
bool found = false;
int captureIdx = 0;
while (-1 != _ptrViE->GIPSVideo_GetCaptureDevice(captureIdx,
                  str, sizeof(str)))
{
      int length = (int)strlen(str);
      if(length > 4 && strncmp(str+(length-5),"(VFW)",5) !=0)
      {
            printf("\tFound camera: %s\n",str);
            found = true;
      }
      captureIdx++;
      memset(str, 0, 64);
}
if(!found)
{
      printf("ERROR no camera connected, required for test\n");
      SLEEP_10_SEC;
      return 0;
}
if(-1 == _ptrViE->GIPSVideo_GetCaptureDevice(0,str,sizeof(str)))
{
      printf("ERROR in GIPSVideo_GetCaptureDevice\n");
      SLEEP_10_SEC;
      return 0;
}
printf("\tUsing camera: %s\n", str);

if(-1 == _ptrviE->GIPSVideo_SetCaptureDevice(str, sizeof(str)))
{
      printf("ERROR in GIPSVideo_SetCaptureDevice\n");
      SLEEP_10_SEC;
      return 0;
}
GIPSCameraCapability cap;
int i =0;
while (_ptrViE->GIPSVideo_GetCaptureCapabilities(i,&cap) != -1)
{
      i++;
}

printf("\n\nTest local camera; \n\tYou should see your local
preview for approximately 10 seconds\n\n");
```

```
HWND myHwnd;
GIPS_CreateWindow(myHwnd,0, 352,288);

if(-1 == _ptrViE->GIPSVideo_AddLocalRenderer(myHwnd, 0,
                  0.0f, 0.0f,1.0f,1.0f))
{
      printf("ERROR in GIPSVideo_AddLocalRenderer\n");
      SLEEP_10_SEC;
      return 0;
}

printf("\tStart the camera\n");
if(-1 == _ptrViE->GIPSVideo_Run())
{
      printf("ERROR in GIPSVideo_Run\n");
      return 0;
}
printf("\tThe camera is started\n");

SLEEP_10_SEC;

GIPSVideo_CodecInst codec;

int numOfCodecs = _ptrViE->GIPSVideo_GetNofCodecs();
for(int i=0; i<numOfCodecs;++i)
{
      if(-1 != _ptrViE->GIPSVideo_GetCodec(i,&codec))
      {
            if(strncmp(codec.plname, "H264", 4) == 0)
            {
                  _ptrViE->GIPSVideo_EnablePLI(channel1,
                                true);
                  codec.pltype = 126;
                  break;
            }

      }
}
printf("\n\nTest H264 codec default settings in CIF 30fps
300kbit/s\n\tYou should see each encoded/decoded codec for
approximately 10 seconds\n\n ");

int bitrate = 300;
strcpy(codec.plname, "H264");
codec.pltype = 126;
codec.bitRate = bitrate;
codec.maxBitRate = bitrate;
codec.width = 352;
codec.height = 288;
codec.frameRate = 30;
codec.codecSpecific = GIPS_H264SingleMode;
codec.quality =4; //default
codec.level = 51; // setting to maximum codec level.
```

```
if(-1 == _ptrViE->GIPSVideo_SetReceiveCodec(channel1,&codec))
{
      printf("ERROR in GIPSVideo_SetReceiveCodec\n");
      SLEEP_10_SEC;
      return 0;
}
if(-1 == _ptrViE->GIPSVideo_SetSendCodec(channel1, &codec,false))
{
      printf("ERROR in GIPSVideo_SetSendCodec\n");
      SLEEP_10_SEC;
      return 0;
}


printf("\tTesting GIPS sockets\n");

if( -1 == _ptrViE->GIPSVideo_SetSendDestination(channel1,
                   12345, "127.0.0.1"))
{
      printf("ERROR in GIPSVideo_SetSendDestination\n");
      SLEEP_10_SEC;
      return 0;
}

printf("\tSend destination 127.0.0.1:12345\n");
if( -1 == _ptrViE->GIPSVideo_SetLocalReceiver(
                      channel1, 12345))
{
      printf("ERROR in GIPSVideo_SetLocalReceiver\n");
      SLEEP_10_SEC;
      return 0;
}
printf("\tLocal receiver 127.0.0.1:12345\n");

printf("\tAdding RemoteRenderer window\n");
if(-1 == _ptrViE->GIPSVideo_AddRemoteRenderer(channel1,
               myHwnd, 0, 0.0f, 0.0f,1.0f,1.0f))

{
      printf("ERROR in GIPSVideo_AddRemoteRenderer\n");
      SLEEP_10_SEC;
      return 0;
}


if(-1 == _ptrViE->GIPSVideo_StartRender(channel1))
{
      printf("ERROR in GIPSVideo_StartRender\n");
      SLEEP_10_SEC;
      return 0;
}
```

```
        if(-1 == _ptrViE->GIPSVideo_StartSend(channel1))
        {
              printf("ERROR in GIPSVideo_StartSend\n");
              SLEEP_10_SEC;
              return 0;
        }
        printf("\tStart Testing: %s\n", codec.plname);

        SLEEP_10_SEC;

        printf("\tStop Testing: %s\n", codec.plname);

        if(-1 == _ptrViE->GIPSVideo_StopSend(channel1))
        {
              printf("ERROR in GIPSVideo_StopSend\n");
              SLEEP_10_SEC;
              return 0;
        }
        if(-1 == _ptrViE->GIPSVideo_StopRender(channel1))
        {
              printf("ERROR in GIPSVideo_StopRender\n");
              SLEEP_10_SEC;
              return 0;
        }


        printf("\tDelete channel:%d\n", channel1);
        if(-1 == _ptrViE->GIPSVideo_DeleteChannel(channel1))
        {
              printf("ERROR in GIPSVideo_DeleteChannel\n");
              SLEEP_10_SEC;
              return 0;
        }
        printf("\tTerminate engine\n");
        if(-1 == _ptrViE->GIPSVideo_Terminate())
        {
              printf("ERROR in GIPSVideo_Terminate\n");
              SLEEP_10_SEC;
              return 0;
        }

        printf("Testing done.\n");
        SLEEP_10_SEC;
        ::DestroyWindow(myHwnd);

        printf("\n\nGIPS video test done\n");
        SLEEP_10_SEC;
        return 0;
}
```

Chapter 5: Error handling

All functions return 0 if the task was successfully performed and –1 otherwise. So when a function returns –1, it does not necessarily mean that there is a major error but rather that the specific task could not be performed. For example the function

```
int GIPSVideo_StartSend(int channel);
```

Will return –1 if the input value, channel, contains a negative number. But it will also return –1 if the channel has not been properly initiated. To find out the specific reason to why the function returned –1, the following function must be called

```
int GIPSVideo_GetLastError();
```

This function returns a positive integer corresponding to the last error that occurred in the GIPS VideoEngine or 0 if no error has occurred. These values are referred to as error codes.

For a description of specific error codes please see appendix A.

Recommended handling of the error codes

The following table shows how the error codes are categorized according to the severity of the error and according to what action that we recommend are to be taken.

Appendix A shows the error code and gives a possible reason to why the problem occurs. This table can be used to create the proper message-box to inform the end-user of the problem and request him/her to check the specific hardware/software not functioning.

Appendix A: Error Codes

The tables in Appendix A describe all the possible error codes that can be returned when calling `int GIPSVideo_GetLastError().`

Generic Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12000	GIPSVideo_SetSendCodec GIPSVideo_SetReceiveCodec	The codec input argument to the function is not initialized.	The input codec instance has not been initialized.
12001	GIPSVideo_GetVersion	The buffer in the function call is too small.	There is not enough space in the input buffer.
12002	GIPSVideo_SetTrace GIPSVideo_SetTraceFileName GIPSVideo_StartRender	An internal error in Video Engine.	Problem allocating memory.
12003	GIPSVideo_Init	Authentication error.	GIPSVideo_Authenticate has not been called with the correct key.
12004	Various functions	The input argument to the function is not valid.	
12005	Various functions	Failed to find one of the input arguments.	GIPSVideo_ConfigureMixer is called for a HWND that is not in use.
12006	Various functions	This is a time limited version of VideoEngine and the time has expired.	
12007	Various functions	Function not available in this build.	
12008	GIPSVideo_StartRecording	VideoEngine requires a voiceEngine for this recording.	No VoiceEngine registered to the video Engine instance.
12009	Various functions	Failed to allocate enough memory.	

Channel Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12010	GIPSVideo_CreateChannel	Problem creating a channel.	Trying to create a channel that is already created.
12011	All functions with channel as input.	The channel cannot be initialized.	A channel is being used without being initialized.
12012	GIPSVideo_CreateChannel	Cannot create a channel.	The channel id could be wrong, e.g. higher than the maximum number of channels.
12013	GIPSVideo_CreateChannel	Failed to create the channel.	The maximum number of channels is exceeded.

Render Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12053	GIPSVideo_SetLocalReceiver	Cannot change settings for the receiver.	The receiver is already rendering. The local renderer is already added.
12054	GIPSVideo_StopRender	Cannot stop render.	The receiver is not rendering.
12055		Failed to change rendering size	
12056	GIPSVideo_AddLocalRenderer	Failed to add the render filter to the filter graph.	
12057	GIPSVideo_ConfigureMixer	The stream id is invalid.	
12058	GIPSVideo_AddRemoteRenderer	Can’t add this window for this channel.	This window has already been added as renderer for this channel.
12059	FrameSizeChange in GIPSVideoRenderCallback	FrameSizeChange returned an error.	

Transport Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12060	GIPSVideo_Init GIPSVideo_Terminate GIPSVideo_StartRender GIPSVideo_StartSend	Cannot open or close the socket manager or a socket.	There is something wrong with the network device, the IP-address or the port number.
12061	GIPSVideo_SetLocalReceiver	Cannot bind the socket.	Trying to use wrong IP-address or port number.
12062	GIPSVideo_SetLocalReceiver GIPSVideo_GetLocalReceiver GIPSVideo_SetSendDestination GIPSVideo_GetSendDestination GIPSVideo_SetSendTOS GIPSVideo_GetSendTOS GIPSVideo_SetSendGQOS GIPSVideo_GetSendGQOS GIPSVideo_GetFromPort GIPSVideo_SetSrcPort	Cannot change the ip-address, port number, ToS or QoS.	External transport is used and these settings should be taken care of by the transport.
12063	GIPSVideo_SetSendDestination GIPSVideo_StartSend	Cannot start sending or set the send destination.	The sender is already running.
12064	GIPSVideo_GetSendDestination	There is no send destination.	Trying to use the send destination parameters before they have been set.
12065	GIPSVideo_StopSend	Cannot stop sending.	The sender is not sending.
12066	GIPSVideo_EnableIPv6	Error enabling IPv6.	Cannot enable IPv6 as the sockets have already been initialised.
12067	GIPSVideo_SetSendDestination	Invalid IP address.	The format of the address is incorrect.
12068	GIPSVideo_SetMTU	Invalid MTU size.	The MTU is either too low or too high, see API description for valid values.

Encryption Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12070	GIPSVideo_EnableSRTPSend GIPSVideo_DisableSRTPSend GIPSVideo_EnableSRTPReceive GIPSVideo_EnableSRTPReceive	Cannot use SRTP.	SRTP is not available.
12071	GIPSVideo_Init	Cannot initialize SRTP.	
12072	GIPSVideo_EnableEncryption	Cannot use encryption.	The encryption has not been initialized.

Codec Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12080	GIPSVideo_GetCodec	Cannot get the codec.	The input argument is wrong, e.g. listnumber is greater than the number of codecs.
12081	GIPSVideo_GetSendCodec	Cannot get the send codec.	The send codec has not been initialized.
12082	GIPSVideo_SetSendCodec	Cannot set the send codec.	The send codec is already in use.
12083	GIPSVideo_SetSendCodec	Cannot set the send codec.	Codec does not support the size.
12084	All functions with a codec instance as input argument.	Invalid codec payload name.	The codec is not supported.
12085	Not used		

RTCP Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12090	GIPSVideo_EnableRTCP	Cannot enable RTCP.	RTCP is already enabled.

12091	GIPSVideo_EnableRTCP	Cannot disable RTCP.	RTCP is not enabled.

Capture Error Codes

Error Code	**Function Name/s**	**Problem Description**	**Possible Reason**
12100	GIPSVideo_SetSendCodec	No capture device selected.	No device for capturing video has been selected.
12101	GIPSVideo_GetCaptureDevice GIPSVideo_SetCaptureDevice	No such capture device.	Trying to set/get a capture device that not exists.
12102	GIPSVideo_GetCaptureDevice GIPSVideo_SetCaptureDevice	Problem with the capture device.	
12103	GIPSVideo_GetCaptureCapabilities	No such list number.	
12104	GIPSVideo_AddCaptureFilter	The capture filter cannot be added.	The filter has wrong input/output format.
12105	GIPSVideo_SetSendCodec	The capture device is busy.	The capture device is used by another application.
12106	Not used		
12107	GIPSVideo_GetSnapshot	There is pending call to the GIPSVideo_GetSnapshot method.	

Recording File Error Codes

Error Code	**Function Name/s**	**Problem Description**	**Possible Reason**
12110	GIPSVideo_StartRecording	Failed to start recording.	Already recording the channel. The codec is not supported for recording.
12111	GIPSVideo_StopRecording	Failed to stop recording.	The channel is not recording.
12112	GIPSVideo_StartRecordingCamera	Failed to start recording camera.	The camera is already recording.

12113	GIPSVideo_StopRecording	Failed to stop recording camera.	Camera is not recording.

Play File Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12120	GIPSVideo_StartPlayFile	Failed to start playing file.	The channel is already playing a file.
12121	GIPSVideo_StopPlayingFile	Failed to stop playing file.	The channel is not playing a file.
12122	GIPSVideo_StartPlayFile GIPSVideo_StartPlayFileAsCamera	The file format is invalid.	The input file is not an AVI file.
12123	GIPSVideo_StartPlayFile	No file decoder is specified.	The channel must not use the default decoder.
12124	GIPSVideo_StartPlayFileAsCamera	Already playing a file as camera.	
12125	GIPSVideo_StopPlayingFileAsCamera	No file is played as camera.	
12126	GIPSVideo_StartPlayingFileAsCamera	Incorrect video size in file.	The video size in file does not match sent codec video size.

Image Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12130	GIPSVideo_SetBackgroundImage	There is already a background image set.	
12131	GIPSVideo_SetBackgroundImage	Image format not supported.	
12132	GIPSVideo_SetBackgroundImage	The image size is not supported.	

Windows Graph Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12300	GIPSVideo_CreateChannel	Cannot create a direct show graph.	
12301	GIPSVideo_CreateChannel	Cannot create a media control for the direct show graph.	
12302	GIPSVideo_StartRender GIPSVideo_Run	Cannot start a direct show graph.	
12303	GIPSVideo_SetReceiveCodec GIPSVideo_AddRenderFilter GIPSVideo_AddLocalRenderer GIPSVideo_AddRemoteRenderer	There is no direct show graph.	Can be using the wrong API call order.
12304	GIPSVideo_EnableMixingRenderer GIPSVideo_AddLocalRenderer GIPSVideo_AddRemoteRenderer	The direct show graph is running.	
12305	GIPSVideo_Stop	Failed to stop the graph.	

Direct Show GIPS Filter Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12310	GIPSVideo_SetSendCodec	Cannot set the send codec.	Problem with the direct show send filter.
12311	GIPSVideo_SetReceiveCodec	Cannot set the receive codec.	Problem with the direct show receive filter.
12312	GIPSVideo_SetSendCodec	Cannot set the send filter.	The send filter and the webcam can't agree on the same media type.
12313	GIPSVideo_SetSendCodec	Cannot connect the send codec.	The send filter and the capture device cannot connect.

Direct Show Filter Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12320	GIPSVideo_SetSendCodec	Different direct show filters cannot use the same media format.	The capture device and send codec cannot agree on the video format.
12321	GIPSVideo_SetReceiveCodec GIPSVideo_SetCaptureDevice	Cannot get a pin on the capture device or the receive codec.	
12322	GIPSVideo_SetSendCodec	Errors creating the direct show filter.	

Direct Show Rendering Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12350	GIPSVideo_AddLocalRenderer GIPSVideo_AddRemoteRenderer	Cannot create VMR.	
12351	GIPSVideo_AddExternalRender GIPSVideo_SetRemoteDisplay	Cannot configure VMR.	
12352	GIPSVideo_SetLocalDisplay GIPSVideo_SetRemoteDisplay	Cannot configure WindowlessControl in VMR.	
12355	GIPSVideo_AddRemoteRenderer	Failed to connect to rendering filter.	

Direct Draw Rendering Error Codes

Error Code	Function Name/s	Problem Description	Possible Reason
12362	All DirectDraw functions	DirectDraw failure.	
12363	GIPSVideo_AddRemoteDemuxRend er GIPSVideo_AddRenderText	DirectDraw failed to find HWND.	

12364	GIPSVideo_AddFullScreenRender	DirectDraw failed to enter true full screen mode.	
12366	GIPSVideo_AddRenderText GIPSVideo_AddRenderBitmap	Invalid argument.	Left, Top, Right or Bottom are outside the picture.
12367	GIPSVideo_AddLocalRender (Windows) GIPSVideo_AddRemoteRender(Windows)	Direct draw hardware acceleration is not enabled on the display card.	