

Bosch Video SDK



BOSCH

Table of Contents

1	Introduction	3
1.1	Features Provided by the Video SDK	3
1.2	Device Version Compatibility	3
1.3	Supported Technologies	3
1.4	Intended Users	4
1.5	Sample Applications	4
1.6	System Requirements	5
2	Installing the Video SDK on a Development PC	6
3	Redistributing the Video SDK with a Client Application	6
4	Using the Video SDK	7
4.1	Components of the Video SDK	7
4.2	Discovering Devices on a Network	8
4.3	Connecting to a Device	8
4.4	Discovering Device Capabilities	8
4.5	Working with Relays	9
4.6	Working with Alarm Inputs	9
4.7	Receiving Device Events	9
4.8	Controlling Cameras	9
4.9	Working with Live Media	10
4.9.1	Streaming Live Video from a Device	10
4.9.2	Streaming Live Audio from a Device	10
4.9.3	Streaming Live Audio to a Device	10
4.10	Working with Media Databases	10
4.10.1	Searching Media Databases	10
4.10.2	Accessing Recorded Media	11
4.10.3	Performing Trick Play Operations	11
4.11	Capturing Still Images	11
4.12	Capturing Media Streams	12
4.13	Streaming Video Directly into a Client Application	12
4.14	Diagnostic Logging	12
4.15	Controlling a Device's Video Outputs	12
4.16	MPEG ActiveX	13

1 Introduction

This user guide describes how software applications can leverage the Bosch Video Software Development Kit (Video SDK) to interact with networked CCTV devices.

1.1 Features Provided by the Video SDK

The Video SDK is a reusable software library that exports a high level, object-oriented API based on COM objects and ActiveX controls. It provides the following features. Note that not all features are supported by every device.

- Network device detection.
- Concurrent network connections to multiple devices.
- Live video rendering from multiple devices including in-window pan / tilt / zoom (PTZ) control.
- Playback video rendering from multiple devices including direction, speed, and stepping control.
- Live and playback audio rendering.
- Audio streaming to capable devices.
- Direct audio and video streaming to client applications.
- Recording of live video and rendering of recorded video.
- Still image capture.
- Control of device video and audio.
- Control of relay outputs.
- Event notification from device relays and alarms.
- Device event searching including input alarms and motion alarms.
- Integrated diagnostic logging.

1.2 Device Version Compatibility

The Video SDK is compatible with the Bosch CCTV devices listed below.

VideoJet 10	VideoJet X10	VIP 10	AutoDome IP	DiBos 8.4 and later
VideoJet 100	VideoJet X20	VIP 1000	Dinion IP	Divar 2
VideoJet 8000	VideoJet X40	VIP-X	FlexiDome IP	Divar XF
VideoJet 8008				Video Recording Manager
				VIDOS Lite Monitor Wall
				VIDOS-NVR 4.00

Table 1.1 Bosch CCTV Devices

1.3 Supported Technologies

The Video SDK is designed to be used in the following ActiveX containers:

- Applications developed with:
 - Visual Studio 2003, 2005, and 2008
 - Microsoft Visual C++ 6.0
 - Microsoft Visual Basic 6.0
- Internet Explorer 5.0 and later (including VBScript and JavaScript)



NOTICE! You may also use the Video SDK with ActiveX containers not listed here.

1.4 Intended Users

The Video SDK is intended to be used by software developers with experience in at least one of the technologies listed in Supported Technologies. In addition, developers should have experience writing client-side code for ActiveX Controls and COM objects within their chosen technology. Developers can learn how to use the Video SDK even without previous COM experience by referring to the included sample applications.

1.5 Sample Applications

The Video SDK includes simple and advanced sample applications. The simple applications demonstrate how to perform basic operations like connecting to a device and displaying live video. A simple sample is included for each type of application listed in Supported Technologies. The advanced applications demonstrate how to perform complicated operations like detecting devices, connecting to devices asynchronously, displaying playback video, and searching for events. The Video SDK includes one advanced sample written in C#.

For Windows XP and earlier the samples are located in:

```
<PROGRAM FILES>\Bosch\VideoSDK\Sample Applications
```

For Windows Vista and later the samples are located in:

```
<USERPROFILE>\Documents\VideoSDK\Sample Applications
```



NOTICE! When the SDK is installed to a non-default location or on an operating system other than English or German, a reference path to the Video SDK's installed interop DLLs must be added in order for the C# sample to compile.

1.6 System Requirements

The following table lists both required and recommended system requirements for end-users or client-application developers.

	End-User	Client Application Developer
Required	<ul style="list-style-type: none"> – Pentium 4, 2GHz CPU (3GHz for Vista and later) – Microsoft Windows 2000 SP4 or later – 256 MB RAM (2GB for Vista and later) – 50 MB hard disk capacity – 100 Mbps Ethernet card – DirectX 9.0c – One of the following video cards that supports DirectX 9.0: <ul style="list-style-type: none"> – NVIDIA GeForce 6600 (128 MB, driver version 93.71) – NVIDIA Quadro FX 4500 PCIe (512 MB, driver version 162.62) – NVIDIA GeForce 7950 GT (512 MB, driver version 93.71) – NVIDIA Quadro FX 3500 PCIe (256 MB, driver version 162.62) – NVIDIA Quadro FX 1500 PCIe (256 MB, driver version 162.62) – NVIDIA Quadro FX 4500 PCIe (512 MB, driver version 162.62) – VER4.30: NVIDIA Quadro NVS 285 PCIe (128 MB, driver version 162.62) – VER4.30: NVIDIA Quadro NVS 440 PCIe (256 MB, driver version 162.62) – ATI FireGL V7200 PCIe (256 MB, driver version 8.323) – ATI FireGL V3300 PCIe (128 MB, driver version 8.323) – ATI Radeon X1300 (256 MB, driver version 6.12) 	
Recommended	<p>Mouse (required for In-Window Pan/Tilt and context menu)</p> <p>Mouse with a mouse wheel (required for In-Window Zoom)</p> <p>512 MB RAM</p>	<p>Microsoft.NET Framework 1.1 is required to run the C# sample applications.</p> <p>Microsoft.NET Framework 3.0 is required to support DiBos8.</p>

Table 1.2 Video SDK System Requirements

2 Installing the Video SDK on a Development PC

The Video SDK installer copies the library files, sample applications, a Video SDK redistributable installer, and documentation to your PC.

3 Redistributing the Video SDK with a Client Application

The Video SDK contains a self-extracting, Video SDK redistributable installer that can be run from third-party installers to copy and configure the Video SDK's runtime libraries on an end-user PC.

The behavior of the redistributable installer can be modified with custom command line arguments. The arguments must be enclosed by double-quotes and must immediately follow the switch /z:

```
/z"<arg 1> ... <arg n>"
```

The redistributable installer recognizes the following custom command line arguments:

Argument	Description
-Quiet	This prevents the installer from displaying its user interface (UI). One exception is the start-up progress bar which appears when the installer is launched. Another exception is self-registration errors. If a DLL fails to self-register, the installer displays an error message.
-MaintMode:###	If the installer runs in maintenance mode, this custom argument selects the mode. The options are: <ul style="list-style-type: none"> - 302 (REPAIR mode) - 303 (REMOVEALL mode). If the installer is running in maintenance mode and this argument is specified, the custom argument "-Quiet" is assumed to be enabled.

Table 3.1 Video SDK Redistributable Installer Command Line Arguments

The redistributable installer writes a return code to the string value "Result" in the registry key:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Bosch\SecuritLibrary"ms\Video SDK Runtime Library"
```

The return codes include:

Code	Description
"CANCELED"	The install was canceled.
"FAILED"	The install failed.
"INSTALLED"	The redistributable was installed for the first time.
"REBOOT"	The redistributable was finished but required the PC to be rebooted.
"REMOVEDALL"	The redistributable was uninstalled.
"REPAIRED"	The redistributable was repaired.
"UPDATED"	The redistributable was updated.

Table 3.2 Video SDK Redistributable Installer Return Codes

4 Using the Video SDK

This section describes using the main functions of the Video SDK. Each subsection contains details about how to perform a major function and its related functions.

4.1 Components of the Video SDK

The Video SDK consists of ten main components. Each component is an ATL COM component – either an ActiveX Control with a user interface (UI) or a non-UI COM object.

Component	Type	High-Level Description
AudioReceiver	COM object	Renders audio from a network device.
AudioSource	COM object	Captures audio for streaming to a network device.
Cameo	ActiveX Control	Renders a live or playback video stream from a network device.
DeviceConnector	COM object	Establishes a connection to a network device.
DeviceFinder	COM object	Detects network devices.
DeviceProxy	COM object	Manipulates a network device.
DiagnosticLog	COM object	Logs messages from client applications and the SDK itself.
MediaFileReader	COM object	Renders a media file recorded with the MediaFileWriter.
MediaFileWriter	COM object	Records a media file to the local file system.
PlaybackController	COM object	Controls playback operations and trick play operations for playback streams.

Table 4.1 Major Video SDK Components

The following diagram depicts the Video SDK’s major components and their relationship to the client application.

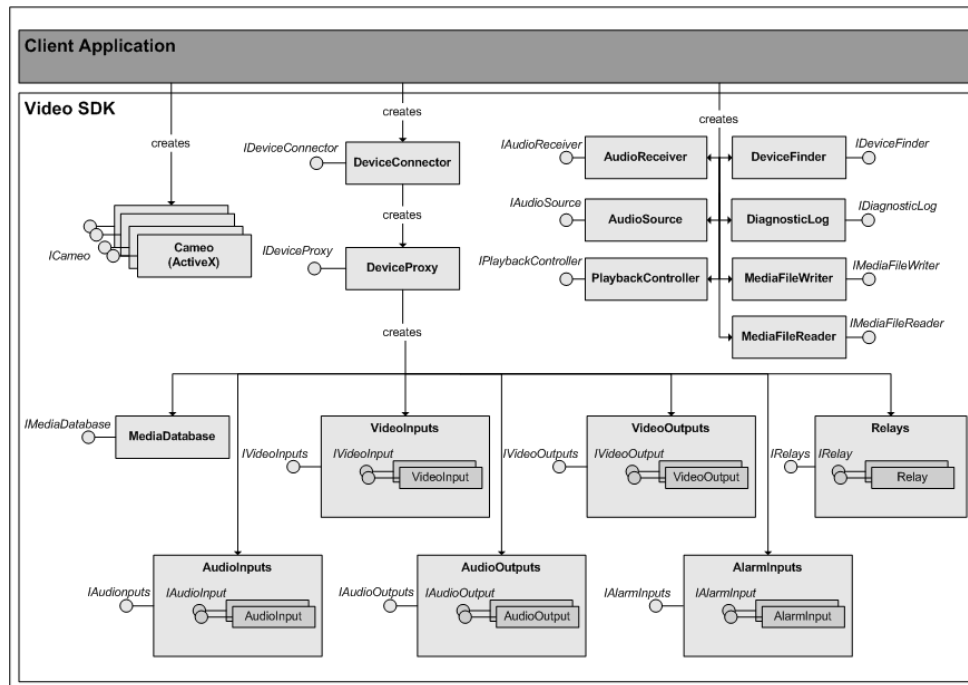


Figure 4.1 Major Video SDK Components

4.2 Discovering Devices on a Network

A client application detects devices on the network using the `DeviceFinder` component's `IDeviceFinder` interface and `_IDeviceFinderEvents` event interface. Detection is started with `IDeviceFinder::StartDetect`. The `DeviceFinder` responds with an asynchronous event for each detected device via `_IDeviceFinderEvents::DeviceDetected`. The `IDeviceFinder::DeviceInfos` property simultaneously maintains a list of all currently detected devices. `DeviceFinder` will also send an `_IDeviceFinderEvents::DeviceRemoved` event for any device that is removed from the network. Detection continues until stopped with `IDeviceFinder::StopDetect`.

4.3 Connecting to a Device

A client application connects to a device using the `DeviceConnector` component. It can be connected to multiple devices simultaneously. Asynchronous connections are established with `IDeviceConnector::ConnectAsync` using a URL describing the communication protocol, IP address or hostname, port number, user name (if required), and password (if required). The `DeviceConnector` responds with an asynchronous event via `_IDeviceConnectorEvents::ConnectResult`. Synchronous connections are established with `IDeviceConnector::Connect` which also uses a URL.

When connection establishment is attempted, the client application receives a `DeviceProxy` for the device. Its `IDeviceProxy::ConnectionState` property indicates if connection establishment was successful. If so, the `DeviceProxy` can be used to interact with the device. If not, it should be immediately discarded.

If an established connection is lost or re-established, the `DeviceProxy` issues an asynchronous event via `_IDeviceProxyEvents::ConnectionStateChanged`.

4.4 Discovering Device Capabilities

When a connection has been established to a device, its `DeviceProxy` can be used to determine its capabilities. The Video SDK supports the device capabilities listed below.

Several of `DeviceProxy`'s properties are collections. If a collection has a count of zero, the capability is not supported. Capabilities are accessed through a collection's individual members. The table below summarizes the capabilities; an "X" denotes that a product supports the capability while an "O" denotes that the capability is an optional feature.

Capability	Collection or Property	DiBos	Dinion IP	Divar 2	DivarXF	Gen-4 AutoDome IP	NVR	Video Jet 10	VideoJet 100	VideoJet 1000	VideoJet 8000	VideoJet 8004	VideoJet 8008	VIDOS Lite Monitor	VIP-X	VIP 10	VIP 1000	VIP 1600
Alarm inputs	AlarmInputs collection	x	x	x	x	x		x	x	x	x	x	x		x	x	x	x
Audio inputs	AudioInputs collection	x			x	x		x	x	x	o	o	o		o	x	x	x
Audio outputs	AudioOutputs collection			x				x	x	x	o	o	o		o	x	x	
Media database	MediaDatabase property	x		x	x		x	o	o	o	o	o	o		o			o
Relay control	Relays collection	x	x	x	x	x		x	x	x	o	o	o		x	x	x	x
Video inputs	VideoInputs collection	x	x	x	x	x		x	x	x	x	x	x		x	x	x	x
Video outputs	VideoOutputs collection			x				x	x	x				x	x	x	x	

Table 4.2 Device Capabilities (x = supported, o = optional)

4.5 Working with Relays

A client application manipulates relays using items from the `DeviceProxy` component's `IDeviceProxy::Relays` collection. It can also receive asynchronous relay state events from `_IRelayEvents::StateChanged`.

4.6 Working with Alarm Inputs

A client application queries alarms using items from the `DeviceProxy` component's `IDeviceProxy::AlarmInputs` collection. It can also receive asynchronous alarm events from `_IAlarmInputEvents::StateChanged`.

4.7 Receiving Device Events

Each device supports a particular set of device events. A client application can receive such asynchronous events from the `DeviceProxy` component's `_IDeviceProxyEvents` event interface via `_IDeviceProxyEvents::DeviceEventReceived`. Events received on a more specific event interface (e.g., `_IRelayEvents`) will not also be received on `_IDeviceProxyEvents`.

4.8 Controlling Cameras

A client application controls cameras using either the Camera Control API or the In-Window PTZ feature.

The Camera Control API is accessed with the `DeviceProxy` component's `IDeviceProxy::VideoInputs` collection. It contains `IVideoInputs` which provide access to camera functions via `IVideoInput::CameraController`.

The In-Window PTZ feature is provided by the `Cameo` ActiveX Control and enabled by default. When enabled, cameras can be operated using the mouse. The `Cameo` also supports a context menu for camera operations.

4.9 Working with Live Media

4.9.1 Streaming Live Video from a Device

A client application renders live video using the `Cameo ActiveX Control`. Video streams are selected from a `DeviceProxy` component's `IDeviceProxy::VideoInputs` collection. A stream is rendered by assigning its `IVideoInput` to the `Cameo` via `ICameo::SetVideoStream`.

4.9.2 Streaming Live Audio from a Device

A client application renders live audio using the `AudioReceiver` component. Audio streams are selected from a `DeviceProxy` component's `IDeviceProxy::AudioInputs` collection. A stream is rendered by assigning its `IAudioInput` to the `AudioReceiver` via `IAudioReceiver::AddStream`.

4.9.3 Streaming Live Audio to a Device

A client application streams PC microphone audio to a device by using the `AudioSource` component. Audio outputs are selected from a `DeviceProxy` component's `IDeviceProxy::AudioOutputs` collection. Audio is streamed by assigning the stream from the `IAudioSource` to the device's `AudioOutput` via `IAudioOutput::SetStream`.

4.10 Working with Media Databases

A client application accesses device recordings using the `DeviceProxy` component. The `IDeviceProxy::MediaDatabase` property supports search and playback operations.

4.10.1 Searching Media Databases

A client application searches device recordings using `ISearchSessions` which are acquired via `IMediaDatabase::CreateSearchSession`. Searches are started via `ISearchSession::Start`. Search results are returned by asynchronous events via `_ISearchSessionEvents`. Searches end when an asynchronous event `_ISearchSessionEvents::Progress` is received with a `Progress` parameter equal to 100.

Searching for Tracks

A media database stores video and audio data in tracks. A client application specifies a track search by passing `steTrack` to `IMediaDatabase::CreateSearchSession`. A search can be constrained by a time interval or a particular set of track identifiers. Asynchronous `_ISearchSessionEvents::TrackAvailable` events return an `ITrack` for each track found.

Searching for Events

A media database stores events that record alarm inputs, video loss, etc. A client application specifies an event search by passing `steEvent` to `IMediaDatabase::CreateSearchSession`. A search can be constrained by a time interval or a particular set of event types. Asynchronous `_ISearchSessionEvents::ResultAvailable` events return information for each event found.

Searching for Timeline Information

A timeline is an event history for a set of tracks. It is represented as a table. A row represents a combination of a track identifier and an event type. A column represents a fraction of the timeline measured in seconds.

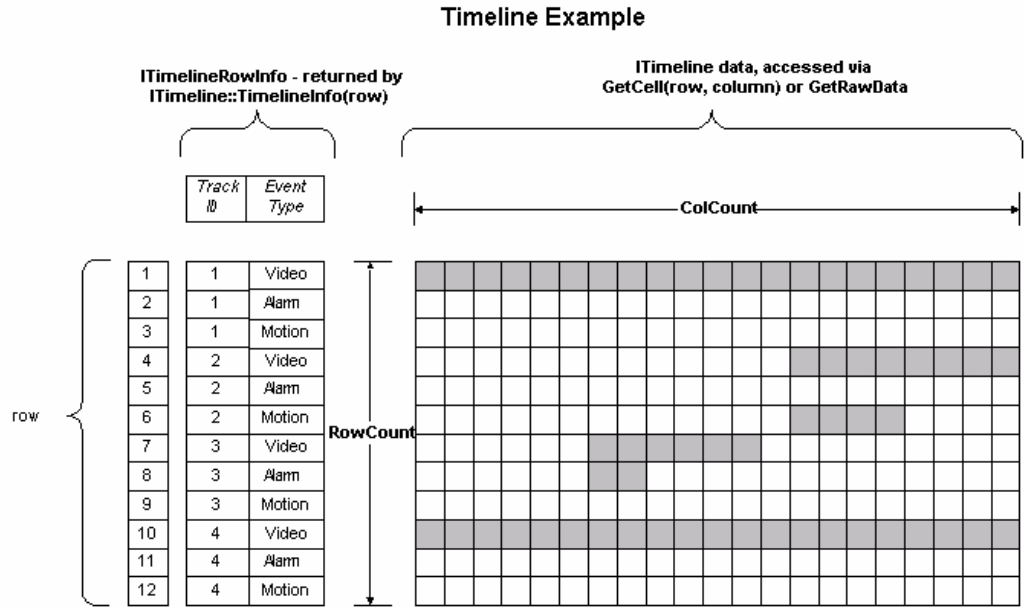


Figure 4.2 Timeline Example

A client application specifies a timeline search by passing `steTimeline` to `IMediaDatabase::CreateSearchSession`. A search can be constrained by a time interval, granularity (how many seconds each column represents), a particular set of track identifiers, and/or a set of events to locate on the tracks. An asynchronous `_ISearchSessionEvents::TimelineAvailable` event returns an `ITimeline`.

4.10.2 Accessing Recorded Media

A client application accesses tracks using an `IMediaSession` acquired from `IMediaDatabase::GetMediaSession` or from `ITrack::GetMediaSession`. Streams provided by the `IMediaSession` can be rendered by assigning them to a `Cameo` or `AudioReceiver`.

4.10.3 Performing Trick Play Operations

A client application controls trick play operations on a video playback stream by using the `IPlaybackController` passed to `IMediaDatabase::GetMediaSession` or `ITrack::GetMediaSession` when the stream was acquired. Trick play operations include changing playback direction or speed, seeking within the stream, and performing single-step operations. Single-step operations require `IPlaybackController::SmoothReverseEnabled` to be set to `TRUE`. If the `IPlaybackController` is shared by several media sessions, all streams will be controlled in concert.

4.11 Capturing Still Images

A client application captures still images using the `Cameo` ActiveX control method `ICameo::CaptureSingleFrame`. Image formats include JPEG and BMP which are selected via `ICameo::CaptureFormat`.

4.12 Capturing Media Streams

A client application records `IDataStream` media streams to a file using the `MediaFileWriter` component. Rendering the file requires a viewer application created with the `MediaFileReader` component and `Cameo ActiveX Control`.

4.13 Streaming Video Directly into a Client Application

Note: This feature requires knowledge of Microsoft DirectShow custom filter development.

A client application can directly access decompressed video stream images using a custom filter as follows:

1. Access an `IDataStream::Filter` to acquire an `IUnknown` pointer to the upstream filter.
2. Call `QueryInterface` on the pointer to acquire the Microsoft DirectShow interface `IBaseFilter`.
3. Call `IBaseFilter::QueryFilterInfo` to acquire the Video SDK's filter graph.
4. Stop the filter graph.
5. Add the custom filter to the filter graph.
6. Run the filter graph. The custom filter will directly receive decoded images from the Video SDK's filter graph.

4.14 Diagnostic Logging

The Video SDK supports diagnostic logging which is disabled by default. The output logs are used by Bosch to diagnose the VSDK's behavior. Log files are circular, fixed length, tab-delimited text files with a default size of 2MB. They can be viewed with spreadsheet applications.

A client application controls logging with the `DiagnosticLog` component's `IDiagnosticLog` interface. Logging is enabled via `IDiagnosticLog::SetLoggingLevel`. A client application can add its own runtime information to the log via `IDiagnosticLog::LogMessage`.

4.15 Controlling a Device's Video Outputs

Devices typically have video outputs. A video output displays video in a region called a cameo (not to be confused with services provided by the `Cameo ActiveX Control`). An arrangement of one or more cameos is called a layout. Some video outputs support several layouts. Video assigned to a cameo is provided by a channel. A channel's video is provided by a video input or an internal video decoder. A video decoder's video is provided by a connection to another device's video encoder.

A client application controls video outputs using items from the `DeviceProxy` component's `IDeviceProxy::VideoOutputs` collection. An `IVideoOutput` provides access to layouts, channels, and the video output's display configuration.

A client application assigns a channel (video stream) to a video output cameo by setting an `IDisplayConfiguration::CameoMapping` array item to an `IVideoOutputChannel::Index` value, then calling `IVideoOutput::SetDisplayConfiguration`.

Cameos in the `IDisplayConfiguration::CameoMapping` array appear in a left-right, top-down (Z) order. For example, if the layout is a 2x2 grid of cameos, the `CameoMapping` array

items, from first to last, would be the left-top, right-top, left-bottom, and right-bottom cameos.

Note: not all devices' video inputs and outputs are compatible. Consult the device documentation for more information.

4.16 MPEG ActiveX

The MPEG ActiveX control only supports the VideoJet, VIP, and NVR devices. It is included so existing client applications can access the Video SDK's functionality with minimal development. It supports the same features as the version 3.0 MPEG ActiveX control with the exception of the following APIs:

- `CameraFilter`
- `ConnectPPP`
- `DisconnectPPP`
- `OpenPlayback` (only works for local files, not HTTP server files)
- `SetMPEG4Overlay`
- `SetLocalMuxMode`

Stubs of the APIs are provided for backwards compatibility, but do nothing.

The MPEG ActiveX control extends the version 3.0 MPEG ActiveX control API with a method called `GetCameo` to provide access to its internal `ICameo` interface. This facilitates additional Cameo-related functionality without forcing the client application to directly integrate the core Video SDK.

Bosch Security Systems, Inc.

850 Greenfield Road
Lancaster, PA 17522

USA

Telephone +1 800-366-2283

Fax +1 800-366-1329

www.boschsecurity.com

© Bosch Security Systems, Inc., 2008; F01U082397 | 4.0 | 2008.04; Data subject to change without notice.