exacqVision®

Software Development Kit

Overview

(updated January 20, 2018)

exacq
From Tyco Security Products

Exacq Technologies, Inc.
11955 Exit Five Parkway, Bldg 3
Fishers, IN 46037 USA

Table of Contents

1 System Level Overview

The core of the exacqVision VMS system is the exacqVision server. All devices connected to the security system communicate directly with the server. Exacq Technologies provides a desktop client that communicates directly with the server, and a web-based client (exacqVision Web Service) that communicates with the server using exacqVision SDK. The exacqVision SDK makes it possible to integrate an exacqVision server into their own systems, and seamlessly incorporate all of its features into their own application.

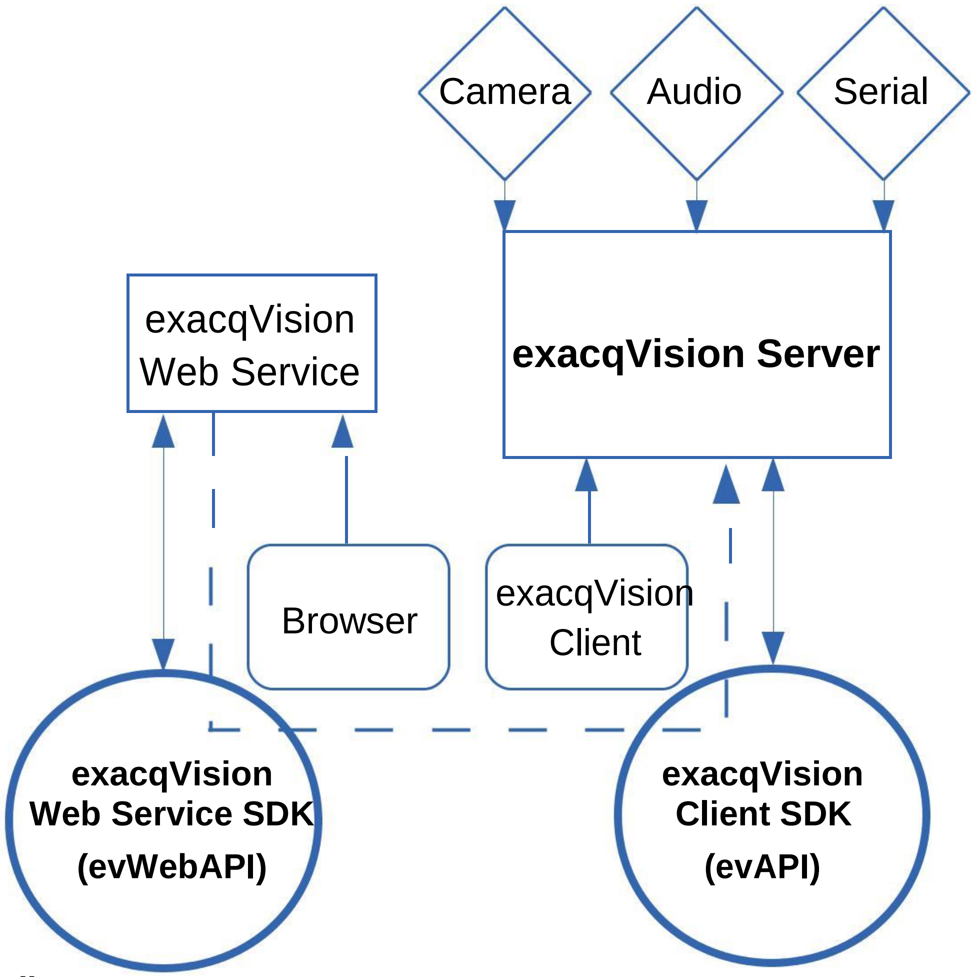


Illustration 1: System Overview

As shown in illustration 1 above, the exacqVision SDK consists of two major components: Client SDK (evAPI) and Web Service SDK (evWebAPI). Which component to choose is dependent on your planned integration. See below for a brief description of each of the two components. Both components are well suited for cross-platform implementations (Windows, OSX, and Linux (Ubuntu)).

1.1 exacqVision Client SDK (evAPI)

The exacqVision Client SDK provides a feature-rich, direct interface to the exacqVision Server that can be used for seamless integration with 3rd party security solutions. The majority of features and functions available through the exacqVision Client are either directly available or are possible to implement using the evAPI. As new features are added to the exacqVision VMS, they are usually made available through the evAPI within one release cycle.

Recommended Implementation Language: C/C++. Other languages, like C#, Python are also available, but may receive limited support.

1.2 exacqVision Web Service SDK (evWebAPI)

The exacqVision Web Service SDK provides a solid set of features based on HTTP protocol. It's mainly used for developing browser based applications that communicate with the web service. The evWebAPI is usually the correct choice for web based or mobile applications and can be used with cross-site scripting.
Recommended Implementation Language: HTML/JavaScript

2 Documentation and Capabilities

ExacqVision SDK implements a wide spectrum of commands that enable the control of virtually all aspects of exacqServer functionality. Here is the overview of some major functional areas.
Configurations:

- Add and remove cameras, security devices
- Modify camera streaming and recording settings; configure camera OSD
- Configure event linking, notifications, serial/pos, archiving, bookmarks and cases
- Configure web panels, views and tours
- Configure users

Discovery:

- Discover cameras on the network (once discovered, they can be added to the exacqServer)
- Discover servers running exacqServer on the network

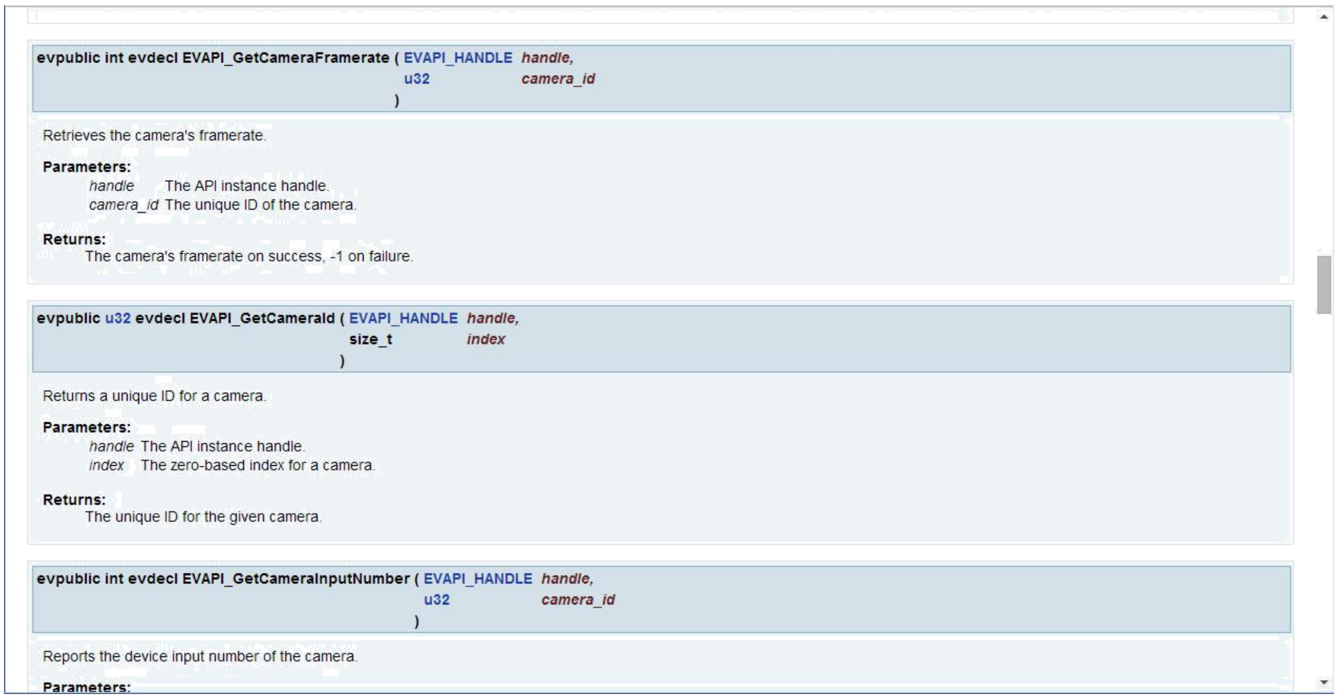
Events and Status:

- All events emitted by the exacqServer are exposed via the SDK, including, for example, motion detection events, trigger events, access control or intrusion sensor events, analytics, camera connection health, archiving health, storage health events, capture board health status, system health status, etc.

Live and Search:

- Search previously recorded and archived video and audio, serial/pos data, events (from event linking configurations)
- Search previously stored analytics metadata for cameras and intrusion panels
- Subscribe to/unsubscribe from live streams of video and audio, serial
- Export recorded or live video, audio and serial using popular formats (avi, mov, mp4, ps)

Thorough documentation, along with code samples, is provided with both SDK components (evAPI and WebAPI). Snippets of the included documentation are shown in the illustrations below:



```
evpublic int evdecl EVAPI_GetCameraFramerate ( EVAPI_HANDLE  handle,
                                               u32           camera_id
                                             )
```

Retrieves the camera's framerate.

Parameters:
handle The API instance handle.
camera_id The unique ID of the camera.

Returns:
The camera's framerate on success, -1 on failure.

```
evpublic u32 evdecl EVAPI_GetCameraId ( EVAPI_HANDLE  handle,
                                        size_t        index
                                      )
```

Returns a unique ID for a camera.

Parameters:
handle The API instance handle.
index The zero-based index for a camera.

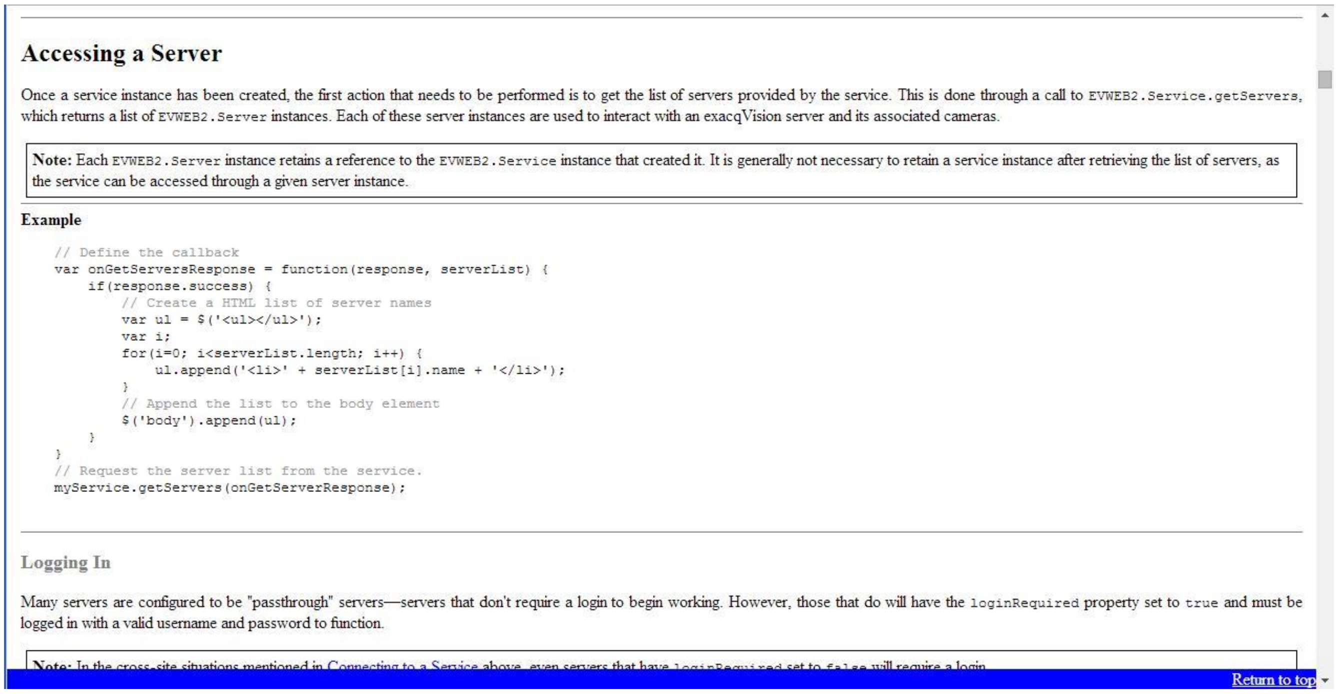
Returns:
The unique ID for the given camera.

```
evpublic int evdecl EVAPI_GetCameraInputNumber ( EVAPI_HANDLE  handle,
                                                 u32           camera_id
                                               )
```

Reports the device input number of the camera.

Parameters:

Illustration 2: exacqVision Client SDK (evAPI) Documentation Snippet



Accessing a Server

Once a service instance has been created, the first action that needs to be performed is to get the list of servers provided by the service. This is done through a call to `EVWEB2.Service.getServers`, which returns a list of `EVWEB2.Server` instances. Each of these server instances are used to interact with an exacqVision server and its associated cameras.

Note: Each `EVWEB2.Server` instance retains a reference to the `EVWEB2.Service` instance that created it. It is generally not necessary to retain a service instance after retrieving the list of servers, as the service can be accessed through a given server instance.

Example

```
// Define the callback
var onGetServersResponse = function(response, serverList) {
    if(response.success) {
        // Create a HTML list of server names
        var ul = $('<ul></ul>');
        var i;
        for(i=0; i<serverList.length; i++) {
            ul.append('<li>' + serverList[i].name + '</li>');
        }
        // Append the list to the body element
        $('body').append(ul);
    }
}
// Request the server list from the service.
myService.getServers(onGetServerResponse);
```

Logging In

Many servers are configured to be "passthrough" servers—servers that don't require a login to begin working. However, those that do will have the `loginRequired` property set to `true` and must be logged in with a valid username and password to function.

Note: In the cross-site situations mentioned in Connecting to a Service above, even servers that have `loginRequired` set to `false` will require a login.

Illustration 3: exacqVision Web Service SDK (evWebAPI) Documentation Snippet

3 exacqVision Client SDK Usage Overview

ExacqVision SDKThis section provides a high level overview on how to use the exacqVision Client SDK (evAPI), along with some pseudo code implementations. For detailed implementation examples, please refer to the sample code provided alongside the evAPI.

3.1 Handles

When connecting to an exacqVision Server, a handle (evHandle) is created which the application needs to use for all subsequent API calls. There is no hard limit on how many evHandles can be created within an application, which allows for simultaneous connections to multiple different exacqVision Servers or multiple connections to the same server.

3.1.1 Thread Safety

The evAPI implementaion is not safe, however multi-threaded integrations are possible. Care must be taken to not share evHandles between threads to avoid conflicting requests being made. As an example, if the implementation processes both events (such as motion, video loss, etc.) and video streaming, a separate evHandle should be created for each function group.

3.2 Callbacks

Most of the information provided to the application by the evAPI is provided through a user defined callback function. This applies to both status messages such as Motion, Video Loss, etc. as well as responses to requests such as search results. It is possible to assign each evHandle its own callback function.

Parameters made available in the callback include the type of event that invoked the callback, a payload containing additional information (if applicable), as well as a pointer to user defined data.

3.3 Server / Device / Camera Settings

Detailed information about the exacqVision Server to which the application is connected, along with settings of the various devices/cameras attached to that server, are made available through the evAPI.

3.4 Video

Both live and recorded video can be retrieved from the exacqVision server. The video frames received from the server will always be compressed according to the format used by the camera at time of capture (MJPEG, H264, etc); The evAPI provides a set of functions to decode the frames received by the server to multiple different standard image formats.

Requests for video are made from within the main application, and the actual data is provided through the callback function.

3.4.1 Live Video

If a live camera stream is requested, the server will continuously send compressed video frames until either an end streaming request is made, or a different camera stream is requested for that evHandle. Each frame received by the server will invoke a callback, in which frame information is made available and the compressed frames can be copied into memory; the evAPI does not provide any internal buffering of live frames received and only makes the last frame received available. Retrieval of the last frame received can be performed outside of the callback, e.g. whenever the application is ready to display the frame.

```
evCallback(type, payload)
{
     if (type == live_frame)
     {
          frame = GetCurrentFrame(evHandle);
          timestamp = payload.time;
          liveFrames.push(frame, timestamp);
     }
}

main()
{
     evHandle = evConnectToServer();
     cameraId = evGetCameraId(evHandle,
     0); evStreamCamera(evHandle,
     cameraId); playLive();
}
```

PseudoCode 1: Live Video

3.4.2 Recorded Video

Recorded video is retrieved from a server through a search request. As with the live requests, each frame found in the search request will generate a callback with frame information. Unlike live requests, the search results are stored by the evAPI in a temporary file and can be retrieved by index. Only a single search request can be active for an evHandle at a given time, although a single request can search multiple cameras.

```
evCallback(type, payload)
{
      if (type == search_frame)
      {
            framesFound++;
      }
      else if (type == search_complete)
      {
            search_done = true;
      }
}

main()
{
      evHandle = evConnectToServer();
      cameraId = evGetCameraId(evHandle, 0);
      evSearchCamera(evHandle, cameraId, start, end);
      waitForSearchDone();
      for (all framesFound)
      {
            displayFrame(i);
      }
}
```

PseudoCode 2: Search Video

3.5 Displaying Video

As has been mentioned in the sections above, video is received from the exacqVision server on a frame-by-frame basis, which must be displayed by the integration. The evAPI provides a lightweight Windows player that implements displaying video and has playback speed controls for recorded video.

3.6 PTZ

Camera movement can be controlled directly through the evAPI. The PTZ feature is split into two categories. Analog PTZ refers to cameras that physically move to change the image plane being recorded. Digital PTZ is performed on fixed cameras, with the PTZ functions being emulated in digital form.

3.6.1 Analog PTZ

Cameras that have Pan/Tilt/Zoom functionality can be controlled directly through the evAPI. In addition to basic movement, the evAPI has the ability to go to predefined presets or create new presets on the exacqVision server. Please note that cameras do not report their position, i.e., there is no guarantee on when a PTZ preset is reached.

```
cameraId = evGetCameraId(evHandle,
index); evPTZZoom(evHandle, cameraId);
```

PseudoCode 3: Analog PTZ

3.6.2 Digital PTZ

Before Digital PTZ functions can be used on a camera, a virtual PTZ instance of that camera must be created. This virtual instance is to be used for any digital PTZ function calls.

```
cameraId = evGetCameraId(evHandle, index);
digitalPTZcamera = evMakeDigitalPTZ(evHandle,
cameraID); evPTZZoom(evHandle, digitalPTZcamera);
```

PseudoCode 4: Digital PTZ

3.7 Audio

Audio data can be both sent to and received from an exacqVision server in multiple standardized formats. Streams being received from the server have the same format that they were recorded in, i.e., audio data is not transcoded by the evAPI. The option is given to decode to raw (wave) format if desired.

Requests for audio data are made from within the main application, and the actual data is provided through the callback function.

3.8 Serial Data

Serial data transmitted to the exacqVision server by a configured device can be accessed through the evAPI, along with all profile settings stored on the server to interpret the data (such as line masks, etc.).

Requests for serial data are made from within the main application, and the actual data is provided through the callback function.

3.9 Exporting

Saving of video clips locally is made possible through the evAPI's exporting feature. Exports can be performed to either a generic container such as AVI or MOV, or to a PS file for playback with the exacqVision ePlayer (provided for free). Generic containers (MP4, AVI, MOV) are limited to one video and one audio source per file, where a PS file can support up to a total of 32 channels (16 video / 16 audio). Exporting can be performed on both live and recorded streams.

```
evCallback(type, payload)
{
     if (type == search_complete)
     {
          search_done = true;
     }
}

main()
{
     evHandle = evConnectToServer();
     cameraId = evGetCameraId(evHandle, 0);
     evExport(evHandle, “Export.avi”, WANT_AVI);
     evSearchCamera(evHandle, cameraId, start, end);
     WaitForSearchDone();
     evEndExport(evHandle);
}
```

PseudoCode 5: Exporting Video

4 exacqVision Web Service SDK Usage Overview

This section provides a high level overview on how to use the exacqVision Web Service SDK (evWebAPI). For detailed implementation examples, please refer to the sample code provided alongside the evWebAPI.

4.1 Service Limitations

Unlike evAPI, the evWebAPI relies on the exacqVision Web Service to communicate directly with exacqVision Servers. While the web service is able to connect to any number of servers, it must be configured to expose those servers, before any interaction can be performed. Please consult the exacqVision Web Service documentation for configuring the available servers.

4.2 JavaScript API

In addition to the HTTP API detailed here, evWebAPI also provides a full JavaScript API that fully encapsulates the HTTP API. This JavaScript API can be a preferred choice when developing browser-only integrations.

4.3 Session IDs

Similar to the evAPI handles, each connection to a server is managed by a Session ID. This ID is generated upon initial login to the server and all subsequent requests related to that server require that session ID. In general, session IDs can be considered an identifier of a single connection to a server, which is persisted throughout its usage.

4.3.1 Passthrough

The web service allows for pre-configuring server connections as “passthrough accounts”. These accounts act as public accounts that do not require a login and, instead, can just be directly accessed. When using these server connections, the server ID is equivalent to the session ID.

4.4 Server / Device / Camera Settings

Detailed information about the exacqVision Server to which the application is connected, along with settings of the various devices/cameras attached to that server, are made available through the evWebAPI.

4.5 Video

Both live and recorded video can be retrieved from the exacqVision server. The video frames received from the server will always be compressed according to the format used by the camera at time of capture (MJPEG, H264, etc); evWebAPI provides the ability to retrieve video in its source format or transcoded into JPEGs.

For best support in a web browser environment, transcoded JPEGs are the recommended rendering format.

4.5.1 Live Video

When requesting live video frames via transcoded JPEG (`GET video.web?s=SESSION_ID&camera=CAMERA_ID&format=6`), the most recent frame will be returned from the request, requiring a polling for subsequent frames. Since the polling interval can be faster than the generation of new frames (e.g. polling at 300ms with a camera recording at 1fps), the web service will block subsequent frame requests until a new frame is available. This functionality is facilitated through the streamId parameter, which is just an arbitrary string to uniquely identify a stream of polling requests.

4.5.2 Recorded Video

Recorded video is retrieved from a server through the same mechanism (`GET video.web`) as with live, passing an additional searchID parameter. With recorded data, the playhead is manually controlled, such that when paused, subsequent requests for a frame will return the same frame until playback is restored. Changing the play state of recorded video is done through requests to set the play speed (`POST playspeed.web`) or by manually setting the playback frame (`POST step.web`)

4.6 PTZ

Camera movement can be controlled directly through the evWebAPI. The PTZ feature is split into two categories. Analog PTZ refers to cameras that physically move to change the image plane being recorded. Digital PTZ is performed on fixed cameras, with the PTZ functions being emulated in digital form.

4.6.1 Analog PTZ

Cameras that have Pan/Tilt/Zoom functionality can be controlled directly through the evWebAPI. In addition to basic movement, the evAPI has the ability to go to predefined presets or create new presets on the exacqVision server. Please note that cameras do not report their position, i.e., there is no guarantee on when a PTZ preset is reached. All of these options are accessed through a single endpoint (`POST ptz.web`)

4.6.2 Digital PTZ

Before Digital PTZ functions can be used on a camera, a virtual PTZ instance of that camera must be created (`POST digitalptz.web`). Once created, this virtual instance can be used in all analog PTZ requests which will perform the action on the generated video.

4.7 Audio

Audio data can be both sent to and received from an exacqVision server in multiple standardized formats. Streams being received from the server have the same format that they were recorded in, as well as, raw decoded data (wave format).

Requests for audio data are made via `GET audio.web` with a resulting stream of audio data.

4.8 Serial Data

Serial data transmitted to the exacqVision server by a configured device can be accessed through the evWebAPI, along with all profile settings stored on the server to interpret the data (such as line masks, etc.).

Requests for serial data are through a number of endpoints hosted on `GET serial.web`, providing fine-grained access to serial data, as well as the configuration.

4.9 Exporting

Saving of video clips locally is made possible through the evWebAPI's export feature. Exports can be performed to either a generic container such as AVI or MOV, a PS file for playback with the exacqVision ePlayer (provided for free), or a bundled Windows executable containing the exacqVision ePlayer. Generic containers (MP4, AVI, MOV) are limited to one video and one audio source per file, where a PS file can support up to a total of 32 channels (16 video / 16 audio). Exporting can only be performed with recorded streams.