VIVOTEK
www.vivotek.com
IP Surveillance

Control/Event Channel Specification

Version 1.1f

2010/01/14

www.vivotek.com

Storage Manager

Revision History

Version	Issue date	Editor	Comment
1.0a	2007/05/21	Bruce/ShengFu	Make initial draft
1.0b	2007/06/08	ShengFu	Revised by meeting minutes
1.0c	2007/06/08	Joe Wu	1. Revised the title in banner 2. Add more descriptions for HTTP tunnel
1.0d	2007/08/17	David Liu	Revised the duration of subscribe request
1.0e	2007/09/26	David Liu	Revised the format of control messages
1.0f	2007/10/24	David Liu	Revised the format of event reply message
1.0g	2007/11/27	David Liu	Add visignal usage explanation.
1.0h	2008/11/18	David Liu	Add tampering detection event type.
1.1a	2009/05/14	Elia	1. Update the document template. 2. Remove tampering detection event type. 3. Add event definition messages. 4. Modify event subscribe message to fit event manager. 5. Add current event status in reply message for event subscription. 6. Add timezone and dst in event notify reply message. 7. Revised the duration of subscribe request. Remove in-correct duration node description.
1.1b	2009/07/03	David Liu	1. Update the part of tunnel establishment. 2. The maximum length of session cookie is 32.
1.1c	2009/07/13	Bowz Liu	1. Modify the format of event definition message. 2. The maximum length of event_type_name is 63.

			3. The maximum length of status is 63. 4. Only deal with the first “max_event_count” events in request message.
1.1d	2009/07/16	Bowz Liu	1. Before receiving a reply message, the client is not allowed to subscribe another event.
1.1e	2009/09/17	Bowz Liu	1. Adding the definition of events which are supported by VIVOTEK’s products.
1.1f	2010/01/14	Bowz Liu	1. All the message content written in XML format, the special characters should be replaced by the transformed characters.

Table of Contents

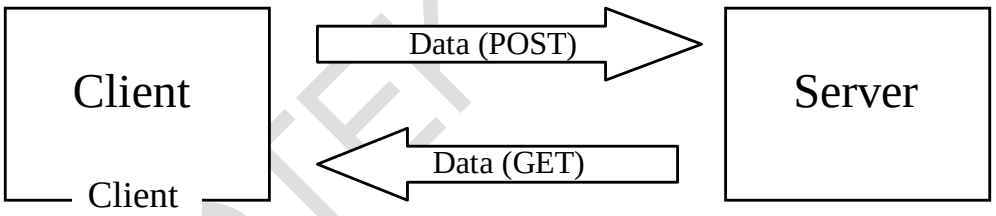
Introduction **5**
HTTP tunneled connection **6**
Message format **10**
1. Event definition messages 11
2. Request messages 13
   2.1. Event subscribe message 13
   2.2. Event unsubscribe message 15
3. Reply messages 16
4. Event notify message 17
5. Control Message 19
6. Option message 20

Introduction

Originally, client can control video server/network camera via cgi command. In addition, user can receive event notification by email, FTP site or HTTP server. These methods are not very real time. Furthermore, the event notification messages can not be managed by a remote centralized surveillance system. The control/event channel provides mechanism which is used to transmit control and event subscription/notification messages more efficiently. All the messages from one client will be transmitted via one pair of HTTP tunnel connections. In addition, subscriptions from multiple clients are possible. HTTP tunneled connection can be consistent so that TCP connection won't be established each time a message would be delivered. Also, HTTP connection can be the most traversable to firewall and compliant to HTTP authentication if security issue is concerned. Thus, the video server/network camera can be managed more efficiently and systematically

HTTP tunneled connection

The HTTP tunneled connections use the capability of HTTP's GET and POST methods to carry an indefinite amount of data in their reply and message body respectively. Generally, the client makes an HTTP GET request to the server to open the one way connection of server-to-client. Client can use this connection to receive data from server. Then the client makes a HTTP POST request to the server to open the one way connection of client-to-server. Client can use this connection to send data to server. Server will bind these 2 paired connections according to the x-sessioncookie filed in HTTP header to form a virtual full-duplex connection makes it possible to send and receive data from one client.



To work with HTTP tunneled connection, client must

(1) Establish one TCP socket to server (download socket) and send GET HTTP message.
Please note that the sessioncookie in the HTTP header should be unique and the length of cookie should be less than or equal to 32 bytes.
(2) Receive 200 OK from server
(3) Establish the other TCP socket to server (upload socket) and send POST HTTP message with the same x-sessioncookie.
(4) Receive tunnel status string in download socket ("HTTP tunnel accept=1") to confirm one pair of tunneled sockets are ready
(5) Client is ready to send data in upload socket and receive reply in download socket

For example:

From client to server (in download socket)

```
GET /cgi-bin/admin/ctrlevent.cgi HTTP /1.0
User-Agent: TunnelClient
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Accept: application/x-vvtk-tunnelled
Pragma: no-cache
Cache-Control: no-cache
Connection: Keep-Alive
```

To make HTTP tunnel works optimally:

- Be made using HTTP version 1.0
- Include in the header an x-sessioncookie directive whose value is a globally unique identifier (GUID). The GUID makes it possible for the server to unambiguously bind the two connections.
- In POST requests, the application/x-vvtk-tunneled MIME type for both the Content-Type and Accept directives must be specified; this MIME type reflects the data type that is expected and delivered by the client and server.

From server to client (in download socket)

```
HTTP/1.1 200 OK
Content-Type: application/x-vvtk-tunnelled
Date: Sun, 9 Jan 1972 00:00:00 GMT
Cache-Control: no-store
Pragma: no-cache
x-server-ip-address: 168.95.2.32
Server: Boa/0.94.14rc21
Accept-Ranges: bytes
Connection: close
```

When the server receives an HTTP GET request from a client, it must respond with a reply whose header specifies the application/x-vvtk-tunneled MIME type for both the Content-Type and Accept directives.

Server reply headers may optionally include the Cache-Control: no-store and Pragma: no-cache directives to prevent HTTP proxy servers from caching the transaction. It is recommended that implementations honor these headers if they are present.

The Date directive specifies an arbitrary time in the past. This keeps proxy servers from caching the transaction.

Server clusters are often allocated connections by a round-robin DNS or other load-balancing algorithm. To insure that client requests are directed to the same server among potentially several servers in a server farm, the server may optionally include the x-server-ip-address directive followed by an IP address in dotted decimal format in the header of its reply to a client's initial GET request. When this directive is present, the client must direct its POST request to the specified IP address regardless of the IP address returned by a DNS lookup.

From client to server (in upload socket, the second method)

```
POST /cgi-bin/admin/ctrlevent.cgi HTTP/1.1
Expires: Sun, 9 Jan 1972 00:00:00 GMT
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 32767
User-Agent: TunnelClient
Connection: Keep-Alive
```

From server to client (in download socket)

```
HTTP tunnel accept=1
```

The sample client POST request includes three optional header directives that are present to control the behavior of HTTP proxy servers:

- The Pragma: no-cache directive tells many HTTP 1.0 proxy servers not to cache the transaction.
- The Cache-Control: no-cache directive tells many HTTP 1.1 proxy servers not to cache the transaction.
- The Expires directive specifies an arbitrary time in the past. This directive is intended to prevent proxy servers from caching the transaction.

Server will check the correct URI and matching x-sessioncookie in the GET and POST message to bind tunneled sockets. Therefore there must be a header of x-sessioncookie with the same string in both GET and POST message for server to check.

There are two methods to generate proper POST message for tunnel establishment.

The first method is:

```
POST /cgi-bin/admin/ctrlevent.cgi HTTP/1.1
Expires: Sun, 9 Jan 1972 00:00:00 GMT
```

```
x-sessioncookie: 5AasdGTHfgjwsqDdSF33
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 7
User-Agent: TunnelClient
Connection: Keep-Alive
VerifyPassProxy: yes
```

(1) Add "VerifyPassProxy: yes" in the HTTP header.
(2) Specify the Content-Length in the HTTP header to 7 and actually fill in the content of the POST body with 8 bytes. By this way the tunnel server can detect whether the HTTP proxy do check the content length and forward 7 bytes only and decide this session pair is suitable for consistent tunnel or not.

The second method is, in the POST message, header of Content-Length is used to keep the POST connection alive by marking large amount of data to upload (32767 bytes).

Some HTTP proxies might cache POST message. Since client send a POST request with Content-length 32767, sometimes HTTP proxy will not forward any data until certain amount of data is coming. To detect this behavior of HTTP proxy, the tunnel status string is used to confirm the success of tunnel socket binding. If tunnel status string is "HTTP tunnel accept=0", this means server fail to binding the GET message due to time out of POST message. In this case, client should teardown the paired sockets and re-establishes the paired sockets by "normal POST"

In HTTP tunnel mode, client won't disconnect the tunneled sockets actively because the advantage of consistent connection with server will make information exchange more efficiently. However, in the above scenario, client will need to establish a new TCP socket each time to send a POST message with upload data as message body and disconnect TCP connection after finishing transmission. In the mean time, client still keeps GET (download socket) alive. This way can avoid HTTP proxy server to cache the upload message from client. Of course each POST message comes with the same x-sessioncookie header. This is different from tunneled socket which got 2 consistent connections for upload and download data. In this scenario, only download connection is consistent, the other one will be re-established each time a new data is ready to send (upload) from client.

After HTTP tunneled sockets are established successfully, client is ready to send control, event subscribe/unsubscribe messages and receive event notify via the tunneled sockets. All the following control or event messages in the upload/POST tunneled sockets should be base64 encoded to traverse HTTP proxy.

Message format

The message format is as follows.

Message Type	Length Tag	Length Information	Message Content
1 byte	1 bit	7 bits ~128 bytes	Length bytes

- Field descriptions:

Field name	Length	Description
Message Begin Tag	1 byte	0xFF This is used to tell a begin of a message once message length is messed up
Length Tag	1 bit	0 means following 7 bits is the length of message content directly 1 means following 7 bits is the length of content length.
Length Information	7 bits ~128 bytes	If *length tag* is 0, read the value of 7 bits as the length of message content If *length tag* is 1, read the value of 7 bits as the length of content length. It its value is *n*, then read the value of following *n* bytes as the content length. The MSB is in the first byte.
Message Content	m bytes	Information corresponding to message type. `The content of message is written in XML format.`

For example: if the message is transmitted and the length of content is 29 (0x1D) bytes which is not larger than 127 bytes. Its format would be like as following.

Field name	Message Begin Tag	Length Tag	Length Information	Message Content
Length	1 byte	1 bit	7 bits	9 bytes
Value	0xFF	0	0x1D	<option><keepalive/></option>

If the event message is transmitted and the length of content is 250-bytes which is larger than 127 bytes, its format would be like as following.

Field name	Message Begin Tag	Length Tag	Length Information	Message Content
Length	1 byte	1 bit	15 bits	250 bytes
Value	0xFF	1	0x1FA	<message>… …</message>

The message content is in XML format. There are 6 types of message content in the current version. They are definition, request, reply, notify, control and option messages. The root element of definition message is “definition”. The root element of request message is “request”. The root element of reply message is “reply”. The root element of event notification message is “message”. The root element of control message is “control”. The root element of option message is “option”.

Because the message content is written in XML format, all the special characters should be replaced by the transformed characters. The following is the transforming table.

Original character	Transformed character
'	&apos;
"	&quot;
<	&lt;
>	&gt;
&	&amp;

1. Event definition messages

When the tunnel connection is properly set up, sever will send its supported event definitions and information to client. The root element is “definition”. The child elements of event_type define all the event types supported by the server, and the child elements of event_info show the information used in the server.

Format:

```
<definition>
  <event_type>
    <event_type_name_1>
    </event_type_name_1>
    …
    <event_type_name_n>
    </event_type_name_n>
  </event_type>
  <event_info>
    <event_information_1>
    </event_ information _1>
    …
    <event_ information _n>
    </event_ information _n>
  </event_info>
</definition>
```

event_type_name is the name of supported event types. Its child elements are described in the table

below. Please note that the maximum length of event_type_name and status are both 63 Bytes.

Element name	Type	Description
number	Integer	This element indicates the maximum number for the event source. Ex. motion window #1, motion window #2.
method	String "enum" or "compare"	The method to represent the possible condition in the event. It supports two methods, enum and compare.
status	String	This field is required when the method is enum. It lists all possible situations of the event. Adding '~' before the status name indicates the pulse trigger. No '~' implies level/edge trigger.

event_info shows the information of event/control channel. Its child elements are described in the table below.

Element name	Type	Description
max_event_count	Integer	This element indicates the maximum number for events which can be subscribed by each client.

Here is a definition message example.

```
<definition>
  <event_type>
    <di>
        <number>1</number>
        <method>enum</method>
        <status>normal</status>
        <status>trigger</status>
    </di>
    <motion>
        <number>3</number>
        <method>enum</method>
        <status>normal</status>
        <status>trigger</status>
    </motion>
    <tampering>
        <number>1</number>
        <method>enum</method>
        <status>~trigger</status>
    </tampering>
```

```
  </event_type>
  <event_info>
    <max_event_count>20</max_event_count>
  </event_info>
</definition>
```

2. Request messages

There are two kinds of request messages. They are event subscription message and event unsubscription message respectively.

Event subscribe message is initiated by client which is used to subscribe the type of event client wants to watch. The event type name, source, and status could be achieved from event definition messages. Client can subscribe multiple events in one message. Since the event_type and the event_info had been defined in the definition message, event/control channel only deals with the first *max_event_count* events in the request message. Those events listed after that will be ignored.

Also, one event notification message can contain several event types.

2.1. Event subscribe message

Here is an example of event subscribe format.

```
<request>
    <subscribe>
        <motion>
            <source>1</source>
            <status>trigger<status>
            <delay>10</delay>
            <duration>
                <weekday>1-5</weekday>
                <time>08:30:00-17:30:00</time>
            </duration>
        </motion>
        <di>
            <source>0</source>
            <status>normal~trigger<status>
            <delay>5</delay>
        </di>
        …
    </subscribe>
</request>
```

The events client subscribes list here

- "source" element means the source of the event type. For example, there might be several di, do or motion detection widows to subscribe. It starts from 0, increases by 1.
- "status" element means the status client wants to subscribe.

  If the event is level-triggered, the content should be as follows,

  <status> one_possible_status </status>

  If the event is edge-triggered, the content should be as follows,

  <status> one_possible_status~another_possible_status </status>

  If the event is pulse-triggered, the content should be as follows,

  <status> one_possible_status </status>

  Note: Such status must include "~" as prefix. For example, "~trigger", the status of network.

  Multiple status value can be listed in this element, which must be separated by a comma.

  <status> one_possible_status, another_possible_status </status>
- "delay" (unit in second) means the event won't be triggered again before a certain interval. For example, if "trigger" status is subscribed and "delay" of 10 seconds is assigned, the trigger event notification will be sent out every 10 seconds before "trigger" status is changed.

  Setting as -1 means only detect event once.

  Setting as -2 means run once when entering the valid interval every time.
- "duration" contains the detail representation of specific schedule. It is the parent element of date, month, monthday, weekday, time, hour, minute, second, and expiretime. If not specified, it means always.

  Note: The duration format of expire-time and day-time cannot be used at the same time.

  Please refer to the table below for detail format of duration format.

Element name	Type	Period of schedule	Description
date	MM/DD	year	MM indicates the month; DD indicates the day of the month. They are separated by slash.
month	Integer	year	The number of months since January, in the range 1 to 12.
monthday	Integer	month	The day of the month, in the range 1 to 31. (The maximum value depends on the month)
weekday	Integer	week	The number of days since Sunday, in the range 0 to 6.
time	hh:mm:ss	day	hour, minute and second separated by colon. Every value has to display in two digits, for example 08:32:00.
hour	Integer	day	The number of hours past midnight, in the range 0 to 23.
minute	Integer	hour	The number of minutes after the hour, in the range 0 to 59.
second	Integer	minute	The number of seconds after the minute, normally in the range 0 to 59

expiretime	Integer	minute	The expire time of subscription

The following table indicates the events which are supported by VIVOTEK's products. Because the source number of each event may be different in each model, this table only indicates the *event_type_name* and status.

Please note that the server program is backward supportive. In the previous version, the server program accepts the statuses called rising and falling in the di event. In this version, these two statuses are also accepted in the di event, and *Rising* means *normal~trigger*, *falling* means *trigger~normal*.

Event_type_name	Status
di	normal, trigger, *rising*, *falling*
do	normal, trigger
motion	normal, trigger
motion0	normal, trigger
pir	normal, trigger
iva	normal, trigger
linkstatus	down, up
daymode	day, night
recnotify	~trigger
tampering	~trigger
network	~trigger
temperature	~trigger
boot	~trigger
visignal	~loss
virestore	~normal

2.2. Event unsubscribe message

Here is the format of event unsubscribe message.

```
<request>
    <unsubscribe>
        <eventid>111</eventid>
        <eventid>222</eventid>
        …
    </unsubscribe>
</request>
```

The events client unsubscribes list here which take effect immediately

"eventid" element is the id returned by server. (See 3. Reply messages)

Client can compose one message containing subscribe and unsubscribe event information.

For example:

```
<request>
```

```
    <subscribe>
        <di>
            <source>0</source>
            <status>trigger,rising<status>
            <duration>
                <expiretime>3600</expiretime>
            </duration>
        </di>
    </subscribe>
    <unsubscribe>
        <eventid>222</eventid>
        <eventid>333</eventid>
    </unsubscribe>
</request>
```

This way subscription and unsubscription of events can be achieved in one message transmission.

3. Reply messages

Server will reply status of subscription or unsubscription to client to confirm status by reply message. So the client is not allowed to subscribe/unsubscribe event before receiving a reply message.

The root element of reply messages is “reply”.

Here is the format of reply message for event subscription/unsubscription.

```
<reply>
    <subscribe>
        <event_type_name_1>
            <eventid>111</eventid>
            <return-code>value</return-code>
            <status>value</status>
        </event_type_name_1>
        <event_type_name_2>
            <eventid>222</eventid>
            <return-code>value</return-code>
            <status>value</status>
        </event_type_name_2>
    </subscribe>
    <unsubscribe>
        <event_type_name_1>
            <eventid>333</eventid>
```

```
            <return-code>value<return-code>
        </event_type_name_1>
    </unsubscribe>
</reply>
```

- "eventid" element means the id of the subscription of the event. The same event id will be in the event notify message for client to match. The value of "eventid" element will be unique during the life time of Control/Event channel
- "return-code" element means the reply status (see table below)

Value of return-code	description
200	OK
406	Not acceptable (no id, illegal source, illegal status, no delay…)
501	Not implemented (no such event to create)
503	Service Unavailable (event subscription is exceeded)

- "status" element is to inform client of the current status of event.

4. Event notify message

Event notify message is replied from server once any subscribed event is triggered. The root element of event notify messages is "message".

Here is the format of event notify.

```
<message>
    <event_type_name>
        <eventid>111</eventid>
        <status>value</status>
        <time>
            <sec>XXXXXXXX</sec>
            <msec>YYY</msec>
            <timezone>VALUE</timezone>
            <dst>BOOLEAN</dst>
        </time>
    </event_type_name >
</message>
```

It indicates the time when the event occurred.

The *time* element indicates the time when the event occurred. It contains four elements, *sec*, *msec*, *timezone*, and *dst*. The first two give the number of seconds and microseconds since the Epoch

(00:00:00 UTC, January 1, 1970). The current time zone location and is in day light saving time or not could be achieved from *timezone* and *dst*.

For motion detection event, there is one optional element, *p*, in the *motion* element. The value of *p* element indicates the percent of motion detection. It ranges from 0 to 100.

Here are some examples of event notify messages.

```
<message>
    <visignal>
        <eventid>111</eventid>
        <status>loss</status>
        <time>
            <sec>1073741826</sec>
            <msec>199</msec>
            <timezone>-360</timezone>
            <dst>1</dst>
        </time>
    </visignal>
</message>
```

```
<message>
    <di>
        <eventid>111</eventid>
        <status>falling</status>
        <time>
            <sec>1073741823</sec>
            <msec>899</msec>
            <timezone>-360</timezone>
            <dst>1</dst>
        </time>
    </di>
    <motion>
        <eventid>222</eventid>
        <status>trigger</status>
        <p>53</p>
        <time>
            <sec>1073741823</sec>
            <msec>899</msec>
            <timezone>-360</timezone>
            <dst>1</dst>
```

```
        </time>
    </motion>
</message>
```

5. Control Message

To send some specific request to configure the setting of video server or network camera, the control message is used. Currently control message encapsulate cgi command in XML format to achieve this function. The root element of control message is “control”. One control message can contain more than one cgi commands.

Here is the format of control message

```
<control >
    <get>
        <cgi>
            <cgiid>111</cgiid>
            <url>cgi command 1</url>
        </cgi>
        <cgi>
            <cgiid>222</cgiid>
            <url>cgi command 2</url>
        </cgi>
        …
    </get>
    <post >
        <cgi>
            <cgiid>333</cgiid>
            <url>cgi command 3</url>
            <body>post body</body>
        </cgi>
    </post>
</control>
```

If the length of cgi command is shorter than 1024, put the command in the group of “get” element, otherwise, put them in the group of “post” element.

Control/Event channel will extract cgi command from this control message and send to HTTP server. For the group of cgi commands in the “get” element, Control/Event channel will compose into HTTP GET method cgi command. For the group of cgi commands in the “post” element,

Control/Event channel will compose into HTTP POST method cgi command and list parameters in the message body. HTTP server will handle this cgi command just like regular one from client. Once HTTP server confirms cgi application finishes the job, it will reply the result to Control/Event channel. Then Control\Event channel will inform client the result of cgi command by reply message

Here is the format of reply message for control

```
<reply>
    <control>
        <cgi>
            <cgiid>111</cgiid>
            <status>reply status value</status>
            <body>message body from HTTP server</body>
        </cgi>
        <cgi>
            <cgiid>222</cgiid>
            <status>reply status value</status>
            <body>message body from HTTP server</body>
         </cgi>
    </control>
</reply>
```

HTTP status code will be adopted directly as reply status value. Client can request multiple cgi command in one control message. However, the reply message might not return the status of cgi command all together.

element *"cgiid"* indicates the id of the cgi command. This id must be unique during the whole life time of Control/Event channel because client needs to tell which reply message of control it is by cgiid since the reply messages return asynchronously.

6. Option message

Currently option message is used for keep alive notification from client to server.

If client crashes, server won't be able to detect of that unless event is triggered. This will lead to zombie subscribed event and might deny further subscription. To avoid this situation, Client needs to send keep-alive message to server side periodically (every 20 seconds). Once server fails to receive keep-alive message in 1 minute, server will teardown the Control/Event channel.

Here is the format of keep-alive message

The root element of keep-alive message is "option"

```
<option>
    <keepalive/>
</option>
```

Server does NOT need to reply keep-alive message.

<End of document>