Netavis Observer 5.0.1

SNAP XML R1.0

Simple Netavis Access Protocol

Netavis SNAP XML R1.0 for Observer 5.0.1

Valid from Observer 5.0.1
Published in June 2019

Copyright

Netavis Software GmbH
Handelskai 388, Top 221
A-1020 Vienna
Austria

Tel. +43 1 503 1722 - 0
Fax. +43 1 503 1722 - 400
info@netavis.net
www.netavis.net

Contents

1. Introduction

This document describes the Simple Netavis Access Protocol (SNAP) that allows bidirectional communication between a Netavis Observer server and a client application. SNAP is an XML-formatted interface using HTTP as transport layer. The first part of the document describes the most important concepts of the interface, which is followed by a description of all SNAP requests and responses.

Please note: For the rest of this document we use the term Netavis to refer to Netavis Observer.

To create a program that uses SNAP with either Java or .NET you need to use the available SNAP APIs (please refer to *2.4 SNAP APIs for Java and .NET*).

The following basic functions are available with this release of SNAP:

- Create a SNAP session with user authentication
- Access the camera topology
- Access live streams and archive recordings of cameras
- Access information of cameras, camera groups, customers, host, users, user groups, and other basic data
- Receive notification on Observer events
- Query Observer events in the EMS database
- Propagate external events to the Observer EMS
- Start Observer actions
- Annotate video streams with text

2. Important SNAP concepts

The SNAP interface provides bidirectional multi-channel communication between a Netavis Observer server (Server) and a client application using the SNAP interface (Client). After creating a SNAP session (Session), the Client can send SNAP Requests to the Server, which are answered by the Server with SNAP Responses. The Server can also notify the Client on Netavis events, which the Client is registered for (Event).

2.1 SNAP Sessions and security

Before you can access any Netavis data or functionality via SNAP, the Client has to create a new SNAP Session. A Session has to be created with the appropriate authentication – valid user name and corresponding password (see *OpenSession* below). The Netavis user's privileges and access rights are then granted to the Session. The Session is alive until it is closed by the Client (see request *CloseSession* below). The Session will be invalidated by the Server automatically if the Client does not send any request within a period of time. Each Session is identified by a SessionID, which is an integer number sent by the Server to the Client as response to the *OpenSession* request. Due to security reasons, the Server accepts requests (identified by a SessionID) only from the IP address that issued the *OpenSession* request.

2.2 SNAP Channels

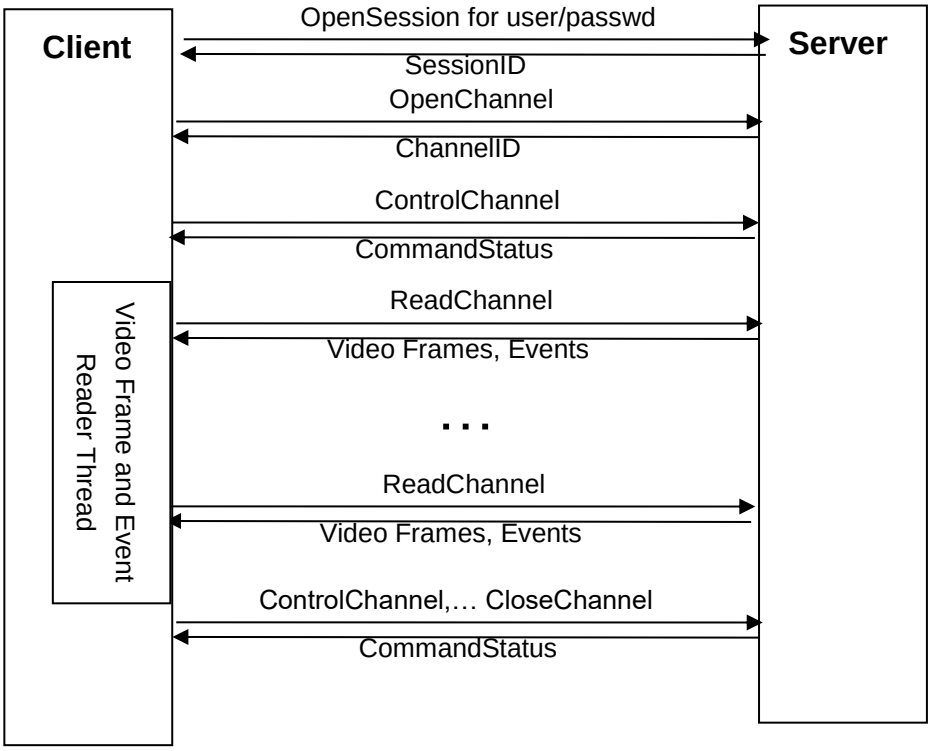
There are two ways how the Server sends information to the Client. The first way is to send response(s) as reply to the Client's HTTP requests. This is the normal request/response communication as we know it for HTTP communication.

The second way is to use a SNAP channel (Channel). The Channel is the way of pushing large amounts of data (video streams, audio data and events) from the Server to the Client.
A Channel is nothing else but an HTTP request initiated by the Client and responded by the Server only if there are data to be transmitted. This means that the HTTP request for reading the channel (see *ReadChannel* below) will be blocked by the Server until there is some data to be sent to the Client (e.g. live or archived video frames, or events). Thus, the SNAP request *ReadChannel* will not return a response immediately in every case. The *ReadChannel* request may be blocked for several seconds (see Timeout parameter of the *OpenChannel* request). Therefore, it is recommended to read the Channel in a separate thread of the Client application, which does not block the application's main event loop. On the other hand, the Server will finalize the *ReadChannel* request in any case after the specified timeout. This means that the Server terminates the *ReadChannel* request independent of data availability on the Server. The Client has to initiate the next *ReadChannel* request, right after terminating the previous *ReadChannel* request. If the Client does not issue another request in time then the Server invalidates the Session. Every further request referring to the invalid SessionID will be rejected by the Server.

The Client can open any number of Channels. One Channel can be used to transfer video stream of one single camera, but it can also be used for transferring multiple live or archived streams of different cameras, and to transfer Events as well. Thus, in many cases one single Channel is enough to transfer video streams of many cameras, and Events from the Server to the Client.

2.3 Using SNAP Sessions and Channels

The Client application has to start the communication by creating a new Session, sending an <*OpenSession*> request. The server will answer immediately by sending a new *SessionID* back. This *SessionID* has to be used in all further requests. Then the Client must open at least one Channel by sending the Request <*OpenChannel*>. The server will respond with a *ChannelID*, which is unique to this session. This *ChannelID* has to be used in all further operations on this Channel. Now, the client may issue one or more <*ControlChannel*> requests, which instructs the server to start filling data into the Channel. Then, the Client may start a thread (Channel Reader Thread) which sends <*ReadChannel*> requests to the Server and processes the response. As response to each <*ReadChannel*> request, the Client receives a multipart http response. The content of the multipart stream depends on previous <*ControlChannel*> request of the client. If the server terminates the multipart response (End of file received), the Channel Reader Thread has to send a new <*ReadChannel*> request, as long as the Channel is still open. In parallel to the Channel Reader Thread, the Client may send any <*ControlChannel*> requests to the server. It may add or remove live or archived streams of cameras, start/stop the live stream of all cameras with a single <*ControlChannel*> request, or register the client for Event notification, or even close the Channel by sending <*CloseChannel*>. After closing the last Channel, the Session has to be closed as well. Otherwise, the server will invalidate the Session after a timeout. It is the Client-implementor's decision how many parallel Channels are used in the Client application. The following figure shows the process of Session and Channel management between Client and Server:



2.4 SNAP APIs for Java and .NET

The SNAP API is available for both Java and .NET and consists of the following parts:

- SNAP API documentation online
- sample source code and executables
- SNAP API libraries to be incorporated in your application

You can download the APIs for Java and .NET from your Netavis Observer server by selecting **Start Customizer** from the main web page. Login as admin user and click on **Download configuration files**.

Installing the SNAP API for .NET

Download **snap.Net20API.zip** (for .NET 2.0 or greater). When you open the ZIP file, you find an executable setup program. Start the setup program to install the SNAP API files on your computer.

The installation contains offline documentation for SNAP .NET and two sample applications, one with a Windows GUI and the other one as console application. Both samples are available as binaries and also as C# source code for Visual Studio.

The library **snap.dll** is available for linking to your programs.

Installing the SNAP API for Java

Download **snapJavaAPI.zip**. When you open the ZIP file, you find documentation, libraries and Java source code. Please open the README.txt for further information.

If you use Java then make sure to use java compiler version 1.8 or greater and include the **snap.jar** in your classpath. The package also contains javadoc of the library and a tester application to be familiar with the API.

The two platform implementations can be different and some functions may be implemented only in java, or in .NET. From version to version, we are working hard on the synchronization of these libraries, but differences can occur. That's why we suggest to make sure before you use the API if you have the latest version of SNAP API to your platform.

3. Transport of SNAP Requests and Responses

Each SNAP Request is an HTTP POST request containing one XML-formatted SNAP Request. Specific SNAP requests and responses are described in the next chapter.

3.1 HTTP transport format of a SNAP Request

The Netavis Observer server is listening on standard HTTP Listen Port (default: 80). The Client has to send POST request to that port as follows (the ***<bold-italic>*** parts of examples below are placeholders for actual SNAP data):

```
POST /arms/servlet/BrowserServlet HTTP/1.0\r\n
Connection: Keep-Alive\r\n
Cache-Control: No-Cache\r\n
Content-Type: text/xml\r\n
Content-Length: <Length-of-SNAP-Request-in-bytes>\r\n
\r\n
<SNAP-Request>
```

3.2 HTTP transport format of a SNAP Response

The Server responds to each SNAP Request with an XML content type document, except the Request ReadChannel, where the content type is multipart. In case of a multipart response, the following different content types are valid in this release of SNAP:

1. text/xml – The content is an XML format SNAP Response
2. image/jpeg - The content is a JPEG frame
3. video/mpeg – The content is an MPEG image frame
4. video/mxpeg – The content is an MxPEG video frame
5. video/h264 – The content is a H.264 video frame
6. video/mpeg4_gop – The content is an Observer GOP (group of pictures)
7. audio/PCMU – The content is a PCM µlaw audio frame
8. audio/PCMA – The content is a PCM alaw audio frame
9. audio/G726-16 – The content is a G726 16Kbps audio frame
10. audio/G726-24 – The content is a G726 24Kbps audio frame
11. audio/G726-32 – The content is a G726 32Kbps audio frame
12. audio/L16 – The content is an L16 audio frame
13. audio/mpeg4-generic – The content is an AAC audio frame

Each frame and the GOP part are preceded by a text/xml part with content SNAP-Response FrameHeader. Please read description of the SNAP-request *ReadChannel* for a detailed description of the response *FrameHeader*.

XML content type response

This is the response to requests **other than ReadChannel**

```
HTTP/1.1 200 OK\r\n
Content-Type: text/xml\r\n
Content-Length: <Length-of-SNAP-Response-in-bytes>\r\n
\r\n
<SNAP-Response>
```

Multipart content type response

This is the structure of the response to the request **ReadChannel.**

```
HTTP/1.1 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=myboundary\r\n
--myboundary\r\n
Content-Type:  text/xml\r\n
Content-Length: <Length of SNAP Response>\r\n
\r\n
<SNAP-Response>
--myboundary\r\n
Content-Type:  <either text/xml (SNAP Response) or image/jpeg (JPEG Frame )>\r\n
Content-Length: <Length of SNAP Response or JPEG Frame in-bytes>\r\n
\r\n
<SNAP Response or JPEG Frame>
--myboundary\r\n
Content-Type:  <either text/xml (SNAP Response) or image/jpeg (JPEG Frame )>\r\n
Content-Length: <Length of SNAP Response or JPEG Frame in-bytes>\r\n
\r\n
<SNAP Response or JPEG Frame>
    .
    .
    .
```

4. SNAP Requests and Responses

This part of the document describes the format of all SNAP requests and responses. Each request is a HTTP POST request containing one XML formatted SNAP Command. The syntax and semantic of the XML formatted requests and responses is described by XML Schema definition (refer to http://www.w3.org/2001/XMLSchema).

The following requests are implemented in this SNAP release:

- **OpenSession** – Open a SNAP Session
- **CloseSession** – Close a SNAP Session
- **LiveSignal** – Send a live signal to the server
- **OpenChannel** – Open a SNAP Channel
- **ControlChannel** – Specify data which has to be transmitted in the Channel
- **ReadChannel** – Read data from Channel
- **CloseChannel** – Close a SNAP Channel
- **PropagateEvent** – Send an event to Netavis (in order to be processed by the EMS)
- **PerformAction** – Perform an action in Netavis
- **SetArchiveProtection** – Protect or unprotect archive records
- **GetVideoArchiveMap** – Get video archive map of a camera for a specified time period
- **VideoStreamAnnotation** – Annotate video streams with text
- **ShowCameraInViewport** – This request brings up a view panel in the client and places a camera in raw/column position.
- **ShowViewOfWindow** – This request brings up a view panel in the client.
- **StartSmartGuard** – This request starts the shuffling of a group of view panels in the client.
- **StopSmartGuard** – This request stops the shuffling of a group of view panels in the client.
- **GetEntityTree** – Get camera and group topology
- **GetIODeviceList** – Get the list of IO devices
- **GetPTZPositionList –** Get the list of defined PTZ positions for a camera
- **GetPTZRouteList –** Get the list of defined PTZ routes for a camera
- **GetPTZRouteDefinition** – Get the definition of the selected camera's PTZ route
- **LockPTZResource** – Lock the PTZ resource of a camera
- **RefreshPTZResource** – Keep locking on a PTZ resource
- **ReleasePTZResource** – Release the lock of a PTZ resource
- **StartPTZRoute** – Start a predefined PTZ route
- **StopPTZRoute** – Stop the predefined PTZ route
- **SetPTZPosition** –Direct the PTZ head to a preset position
- **PTZCenterClick** – Center the PTZ head on the clicked position
- **ContinuousPTZ** – Continuously move the PTZ head into a given direction with a given speed

- **MovePTZRelative** – Move the PTZ head in the given direction with given amount, relative to its current position

The rest of this chapter describes all SNAP requests and the corresponding responses. Schema descriptions below do not contain the most external XML document element <SNAP>. Some elements in the XML Schema definitions below do appear just as a reference. These references are not resolved in this chapter. The complete schema definition file **SNAP.xsd** can be accessed on any Netavis Observer server. Please follow the link **Start Customizer** on the main Netavis Observer web page, enter administrator name (usually “admin”) and password and click on **Download configuration files**. Here click on the link **SNAP.xsd**.

Some text below refers to diverse schema elements in form of <*ElementName*> or *AttributeName*. The definition of these elements can be found in the schema definition file SNAP.xsd. Some important terms are highlighted as *term* and explained later in the text.

4.1 **OpenSession**

Description: Open a new SNAP Session. This must be the first request in every SNAP communication. If the Netavis server is licensed to support SNAP and if there are any free login accounts available for SNAP then this request will open a new SNAP session by returning a new *SessionID*. Otherwise the session request will be rejected and the error condition will be returned.
Note: Only the user "guest" is allowed when you use the demo license.

Parameters: *User* - Netavis user

Passwd – MD5 encoded password

SNAPVersion – Version number of SNAP used in this session. ("V1" is default)

Response: <*NewSession*> or <*ExecutionStatus*> with attribute ReturnCode != 0 on error. See the schema definition **SNAP.xsd** for a schema of <*ExecutionStatus*> and *Appendix B – List of error codes*.

XML Example for the request OpenSession

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <OpenSession User=”peter” Passwd=”084e0343a0486ff05530df6c705c8bb4” SNAPVersion=”V1”/>
</SNAP>
```

XML Example for the response NewSession

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <NewSession SessionID=”8325”/>
</SNAP>
```

4.2 **CloseSession**

Description: Close a SNAP Session. Closing a session will close all open channels associated with this session and free all resources allocated for this session by the server.

Parameters: SessionID – Returned by OpenSession

Response: <*ExecutionStatus*>. See *Appendix B – List of error codes* for error codes and texts.

XML Example for the request CloseSession

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <CloseSession SessionID=”8325”/>
</SNAP>
```

XML Example for the response ExecutionStatus

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ExecutionStatus ReturnCode=”0”/>
</SNAP>
```

4.3 **LiveSignal**

Description: This request simply shows the server that the client, which opened the session is still alive. This request has to be sent every 15 seconds to the server. Otherwise, the server will close the session and consider the client connection as dead.

Note: You need this request only if there is no opened channel in your session! Use of this command is not necessary if the client maintains at least one open channel (sending periodic ReadChannel requests)

Parameters: *SessionID* – Returned by OpenSession

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<LiveSignal SessionID =”3456”/ >
</SNAP>
```

4.4 **OpenChannel**

Description: Open a new SNAP Channel. See *Important SNAP concepts* for more information on SNAP channels.

Parameters: *SessionID* – Returned by OpenSession

TimeLimit – Timeout after which the server terminates a ReadChannel request in any case (seconds).

DataLimit – Size of transferred data in the channel after which the server terminates a ReadChannel request in any case (KBytes). The value 1 cause termination of ReadChannel request frame by frame.

FrameBased – Optional parameter relevant only for archive recording queries (default is false). If true, the MPEG4 and H.264 streams will be sliced by the server into standard MPEG4 or H.264 frames instead of the Observer-specific GOP (group of pictures) format. If false, then the Observer-specific GOP format is used.

Response: <*NewChannel*> on success or <*ExecutionStatus*> on error

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    < OpenChannel SessionID =”3456” TimeLimit =”60”/>
</SNAP>
```

XML Example for the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <NewChannel ChannelID=”1”/>
</ SNAP>
```

4.5 **ControlChannel**

Description: Defines what the channel does and what is transferred in the channel. You can control several live streams, archive streams, event streams with one single ControlChannel request. The following elements are:

- <*StartLiveVideoStream*> Start live video stream of a camera.
- <*StartArchiveVideoStream*> Start archive video stream of a camera.
- <*StartLiveEventStream*> Start sending live events.
- <*StartArchiveEventStream*> Start sending archived events.
- <*StopStream*> Stop (cancel) any stream.
- <*PauseAllLiveVideoStream*> Pause all live video streams.
- <*ResumeAllLiveVideoStream*> Resume all live video streams after a pause.
- <*RegisterForDataChangeNotification*> Resister channel for data change notification.
- <*StartLiveViewStream*> Start sending “stitched” views of multiple camera images in one single stream.
- <StartTimedArchiveVideoStream> Start an archive video stream of a camera with server-side inter-frame timing. After issuing the command the server prepares the stream and puts it into pause mode, so the first call of the client application should be a resume. Later the stream can be paused, resumed or closed.

Note: The XML schema description of the request below does not contain all details. Please look at the full schema SNAP.xsd for all details.

IMPORTANT NOTE: All stream requests contain a *StreamID* parameter. This parameter is a unique ID of the stream, which is assigned to a stream by the client application. *StreamID* identifies the stream for which a data item (Frame or Event) is pushed into the channel (attribute *StreamID* of <*FrameHeader*> or <*Event*>). *StreamID* is also used when a stream should be stopped.

IMPORTANT NOTE: In the <*StartArchiveEventStream*> command please do not request for more than 50000 records! For a listing of valid event names for filtering please refer to Appendix A.

Parameters: *SessionID* – Returned by OpenSession
ChannelID – Returned by OpenChannel

Response: ExecutionStatus

XML Example for the request (start 4 live streams from 2 different cameras)

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ControlChannel SessionID=”8325” ChannelID=”1”>
        <StartLiveVideoStream StreamID=”1” EntityID=”2” Size=”Large”/>
        <StartLiveVideoStream StreamID=”2” EntityID=”4”/>
        <StartLiveVideoStream StreamID=”3” EntityID=”2” Size=”Small” Quality=”High” Fps=”10”/>
        <StartLiveVideoStream StreamID=”4” EntityID=”4” Size=”Small” Quality=” High” Fps=”10”/>
    </ControlChannel>
</SNAP>
```

XML Example for the request (stop 2 streams started in previous request)

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ControlChannel SessionID=”8325” ChannelID=”1”>
        <StropStream StreamID=”3”/>
        <StropStream StreamID=”4”/>
    </ControlChannel>
</SNAP>
```

XML Example for the request (start live event stream and archive video stream)

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ControlChannel SessionID=”8325” ChannelID=”1”>
        <StartLiveEventStream StreamID=”5” EventType=”MotionDetection”/>
        <StartArchiveVideoStream StreamID=”2” EntityID=”4” MaxFrames=100/>
    </ControlChannel>
</SNAP>
```

XML Example for the response (see Schema of <*ExecutionStatus*>)

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    < ExecutionStatus ReturnCode=”0”/>
</SNAP>
```

XML Example for the request. Start a view stream for a mobile device having 817 by 1624 pixels of screen size. We create a 3 by 4 grid in it and start a live stream from camera 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ControlChannel ChannelID="1" SessionID="1234">
        <StartLiveViewStream Fps="25.0" StreamID="1">
            <Canvas BgColor="#444444" Height="817" Width="1624"/>
            <Grid Columns="4" Gap="1" GapColor="#aaaaaa" Rows="3"/>
            <ViewPort ColSpan="1" EntityID="2" FillMode="LETTERBOX" Gridx="0" Gridy="0"
RowSpan="1">
                <ViewPortTitle BgColor="#eeeeee" FgColor="#000088" Height="15" Position="BOTTOM"
Text="Cam"/>
            </ViewPort>
        </StartLiveViewStream>
    </ControlChannel>
</SNAP>
```

XML example for a timed archive video request.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SNAP>
<ControlChannel ChannelID="1" SessionID="1801580153">
<StartTimedArchiveVideoStream EntityID="3" Mode="4" PlaybackSpeedRatio="25.0" StreamID="1"
TargetFPS="5.0">
<DateTimeFilter End="2017-09-05T09:11:00" Start="2017-09-05T09:10:00"/>
</StartTimedArchiveVideoStream>
</ControlChannel>
</SNAP>
```

XML example for controlling the stream.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SNAP>
<ControlChannel ChannelID="1" SessionID="1801580153">
<StreamControl Data="0" StreamID="1" Type="2" TypeName="RESUME_STREAM"/>
</ControlChannel>
</SNAP>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SNAP>
<ControlChannel ChannelID="1" SessionID="1148005993">
<StreamControl Data="0" StreamID="1" Type="1" TypeName="PAUSE_STREAM"/>
</ControlChannel>
</SNAP>
```

4.6 **ReadChannel**

Description: Read data from the channel. This request is blocked by the server until there is any data to be pushed to the client. Which data is pushed is defined by ControlChannel calls. ControlChannel can be called to an open channel any time, independent from ReadChannel. The response to this request is terminated by the server after a timeout or transferred data limit specified in OpenChannel request. The client then has to send a new ReadChannel request within the Session timeout (see *SNAP Sessions and security* for details about Session timeout). The server will cancel the session if no ReadChannel request is received in time.

Parameters: *SessionID* – returned by OpenSession

ChannelID – returned by OpenChannel

Response: Multipart HTTP stream. Each part may contain one of the SNAP-response items described below, or a JPEG frame. ExecutionStatus as text/xml part is the response in case of error. Possible response items:

- <*StartOfStream*> Marks the beginning of a video or event stream. This element precedes the first element of the stream. The attribute *DataItemCount* is added in case of archive streams.
- <*EndOfStream*> Marks the termination of a stream. This element follows the last element of the given stream.
- <*FrameHeader*> Header of a frame in live or archive stream. This part is always followed by a part containing the frame itself.
- <*Event*> A live or archive event.
- <*EntityTreeChanged*> Notification on any change of <*EntityTree*>. See more details at description of the request <*GetEntityTree*>.

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    < ReadChannel SessionID =”3456” ChannelID =”1”/>
</SNAP>
```

XML Schema of response EntityThreeChanged (part of MULTIPART)

The part containing this response is sent by the server whenever the camera topology has changed on the server. The client may than reload the appropriate part of the tree.

```
<xs:element name="EntityTreeChanged">
    <xs:annotation>
        <xs:documentation>Notification on EntityTree change. (response to
            ReadChannel)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="RootEntityID" type="xs:integer" use="required">
            <xs:annotation>
                <xs:documentation>Topmost entity ID of changed part of the tree. </xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```

XML Schema of response FrameHeader (part of MULTIPART)

The part containing this response precedes JPEG frames of live or archive video stream. FrameHeader may contain optional <*InPictureEvent*> descriptor element. The element <*InPictureEvent*> is attached to archived frame if any event has been generated by analyzing this frame (MotionDetection event for example).

```
<xs:element name="FrameHeader">
    <xs:annotation>
        <xs:documentation>Header of one frame of live or archive video.  (response to
            ReadChannel, next part is the frame) </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="InPictureEvent"/>
        </xs:sequence>
        <xs:attribute name="StreamID" type="xs:integer" use="required"/>
        <xs:attribute name="EntityID" type="xs:integer" use="required">
            <xs:annotation>
                <xs:documentation>ID of the source camera. </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="Type" use="required">
            <xs:annotation>
                <xs:documentation>Frame type. Until SNAP R1.1: JPEG, since R1.2: JPEG | MPEG | MXPEG |
H264 | GOP </xs:documentation>
            </xs:annotation>
            <xs:simpleType>
                <xs:restriction base="xs:Name">
                    <xs:enumeration value="JPEG"/>
                    <xs:enumeration value="MPEG"/>
                    <xs:enumeration value="MXPEG"/>
                    <xs:enumeration value="H264"/>
                    <xs:enumeration value="GOP"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="MillisStamp" type="xs:unsignedLong" use="required">
```

```
            <xs:annotation>
                <xs:documentation>Frame creation stamp in millisec since
                    beginnig of 1970 (unix).</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="DateTimeStamp" type="xs:dateTime" use="required">
            <xs:annotation>
                <xs:documentation>Frame creation stamp converted in
                    XML Schema DateTime format.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```

XML Schema of response Event (part of MULTIPART).

The part containing this response is sent by the server if the client has been registered for receiving live or archived event stream. See *Appendix A – List of events* for list of possible EventTypes and their properties.

```
<xs:element name="Event">
    <xs:annotation>
        <xs:documentation>Live event notification or result of an
            event query. (response to ReadChannel)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence minOccurs="0">
            <xs:element ref="EventProperty" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="StreamID" use="required"/>
        <xs:attribute name="EventType" type="xs:Name" use="required">
            <xs:annotation>
                <xs:documentation>Event type name.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="EventID" type="xs:integer" use="required">
            <xs:annotation>
                <xs:documentation>Unique ID of this event. </xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="MillisStamp" type="xs:unsignedLong" use="required">
            <xs:annotation>
                <xs:documentation>Event creation stamp in millisec since
                    beginnig of 1970 (unix).</xs:documentation>
            </xs:annotation>
        </xs:attribute>
        <xs:attribute name="DateTimeStamp" type="xs:dateTime" use="required">
            <xs:annotation>
                <xs:documentation>Event creation stamp converted in XML
                    Schema DateTime format.</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
</xs:element>
```

XML Schema of response StartOfStream (parts of MULTIPART).

```
<xs:element name="StartOfStream">
    <xs:annotation>
        <xs:documentation>Marks start of video or event stream.
            (response to ReadChannel)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="StreamID" type="xs:integer" use="required"/>
        <xs:attribute name="DataItemCount" type="xs:integer">
            <xs:annotation>
                <xs:documentation>Number of data items in stream
                    (only for archive video or event stream).</xs:documentation>
            </xs:annotation>
        </xs:attribute>
    </xs:complexType>
```

```
</xs:element>
```

XML Schema of response EndOfStream (parts of MULTIPART).

```
<xs:element name="EndOfStream">
    <xs:annotation>
        <xs:documentation>Marks end of the stream. (response to ReadChannel)</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="StreamID" type="xs:integer" use="required"/>
    </xs:complexType>
</xs:element>
```

Example for a multipart response to ReadChannel. Two FrameHeaders with attached JPEG frames, and one Event is shown in this example. HTTP protocol tags are included in this example.

```
HTTP/1.1 200 OK\r\n
Content-Type: multipart/x-mixed-replace;boundary=myboundary\r\n
--myboundary\r\n
Content-Type:  text/xml\r\n
Content-Length: 163\r\n
\r\n
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<FrameHeader StreamID=”1” EntityID=”4” Type=”JPEG”
    StampMillis=563272881812345
    StampDateTime=”2005-08-02T12:28:36.45-01:00”/>
</SNAP>
--myboundary\r\n
Content-Type:  image/jpeg\r\n
Content-Length: 9518\r\n
\r\n
<JPEG Frame (9518 bytes length)>
--myboundary\r\n
Content-Type:  text/xml\r\n
Content-Length: 163\r\n
\r\n
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<FrameHeader StreamID=”1” EntityID=”4” Type=”JPEG”
    StampMillis=563272881812385
    StampDateTime=”2005-08-02T12:28:36.85-01:00”/>
</SNAP>
--myboundary\r\n
Content-Type:  image/jpeg\r\n
Content-Length: 9416\r\n
\r\n
<JPEG Frame (9416 bytes length)>
--myboundary\r\n
Content-Type:  text/xml\r\n
Content-Length: 289\r\n
\r\n
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<Event StreamID=”1” EventType=”PlateNumberRecognision” EventID=”34652”
        MillisStamp=”234241234125453” DateTimeStamp=”2005-01-02T00:28:30.23”>
    <EventProperty>
        <Name>EntityID</Name>
        <Value>3</Value>
        <Type>Integer</Type>
```

```
    </EventProperty>
    <EventProperty>
        <Name>PlateNumber</Name>
        <Value>IUR 116</Value>
        <Type>String</Type>
    </EventProperty>
</Event>
</SNAP>
```

4.7 **CloseChannel**

Description: Close a channel. Close channel will automatically stop all streams started within the channel.

Parameters: *SessionID* – returned by OpenSession

ChannelID – returned by OpenChannel

Response: *<ExecutionStatus>*

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    < CloseChannel SessionID =”3456” ChannelID =”1”/>
</SNAP>
```

XML Example for the response (see schema of <ExecutionStatus>)

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    < ExecutionStatus ReturnCode=”0”/>
</SNAP>
```

4.8 **PropagateEvent**

Description: Send an external event to Netavis EMS. Each event which wants to be propagated to the Netavis server has to contain two attributes (*SessionID* and *EventType*) and a list of <*EventPropery*> elements. Each element <*EventPropery*> has to contain the following elements*: <Name>, <Value> and <Type>.* Legal type elements are: String, Integer, Long, Float, Double and Boolean. The content of the element *Value* has to correspond to its type. The list of supported *EventTypes* and its properties is described in *Appendix A – List of events* . Please note, that custom events can also be propagated to the Netavis server (see **CusomEvent** in *Appendix A – List of events*).

Parameters: *SessionID* – Returned by OpenSession

EventType – Type of the Event

EventStamp – UNIX time stamp of the event.

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
< PropagateEvent SessionID =”3456” EventType=”MotionDetection”>
    <EventProperty>
        <Name>CameraID</Name>
        <Value>4</Value>
        <Type>Integer</Type>
    </EventProperty>
    <EventProperty>
        <Name>CameraName</Name>
        <Value>MyCamera</Value>
        <Type>String</Type>
    </EventProperty>
    <EventProperty>
        <Name>EventStamp</Name>
        <Value>1223355554557</Value>
        <Type>Long</Type>
    </EventProperty>
    <EventProperty>
        <Name>EventName</Name>
        <Value>Car on the street</Value>
        <Type>String</Type>
    </EventProperty>
</ PropagateEvent>
</SNAP>
```

4.9 **PerformAction**

Description: Execute a standard Netavis action. Each action must contain two attributes (*SessionID* and *ActionType*) and a list of <*ActionPropery*> elements. Each element <*ActionPropery*> has to contain the following elements*: <Name>, <Value> and <Type>.* Legal type elements are: String, Integer, Long, Float, Double and Boolean. The content of the element *Value* has to correspond to its type. The list of supported *ActionTypes* and its properties is described in *Appendix C – List of available Actions*.

Parameters: *SessionID* – Returned by OpenSession

ActionType – Type of the Action (see *Appendix C – List of available Actions*)

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
< PerformAction SessionID =”3456” ActionType=”SetCameraRecordingState”>
    <ActionProperty>
        <Name>CameraID</Name>
        <Value>4</Value>
        <Type>Integer</Type>
    </ ActionProperty >
    < ActionProperty >
        <Name>Function</Name>
        <Value>Start</Value>
        <Type>String</Type>
    </ ActionProperty >
    < ActionProperty >
        <Name>VideoFormat</Name>
        <Value>JPEG</Value>
        <Type>String</Type>
    </ ActionProperty >
    < ActionProperty >
        <Name>FrameSize</Name>
        <Value>High</Value>
        <Type>String</Type>
    </ ActionProperty >
    < ActionProperty >
        <Name>FrameQuality</Name>
        <Value>Medium</Value>
        <Type>String</Type>
    </ ActionProperty >
    < ActionProperty >
        <Name>FrameRate</Name>
        <Value>5.0</Value>
        <Type>Float</Type>
    </ ActionProperty >
</ PerformAction >
</SNAP>
```

4.10 SetArchiveProtection

Description: Protect or unprotect archive records.
Parameters: *SessionID* – Returned by OpenSession
EntityID – Unique ID of the camera
From – UNIX timestamp when the protection should be started
To – UNIX timestamp when the protection should be finished
Set – True for protect, false for unprotect the records.
Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <SetArchiveProtection SessionID =”3456” EntityID=”38” From=”1328780727” To=” 1328784327”
Set=”true”/ >
</SNAP>
```

4.11 GetVideoArchiveMap

Description: Get *recording-status-map* of the video archive of a camera. The *recording-status-map* contains information about recording status at the timeline specified by the request. The timeline is split into several time periods. Each period is represented by an element <MapItem> in the response to this request. Thus, the response is a list of <*MapItem*>s. Each <MapItem> covers a part of the requested timeline. The start of time period covered by <*VideoArchiveMap*> is described by attribute *StartDateTime.* The <*VideoArchiveMap*> contains a *Unit* attribute, which describes the duration of one *time-unit* within the <MapItem>. The *time-unit* can be either "Day" or "Minute". If the *time-unit* is "Day" then the timeline is split into <MapItems> having duration of 1 month (28-31 *time-units*). If the *time-unit* is "Minute" then the timeline is split into <MapItems> having duration of 1 hour (60 *time-units*). Each <*MapItem*> contains attribute *FrameCount* which contains the total number of frames recorded in the time period covered by the item. Each <*MapItem*> contains a sequence of hexadecimal coded character (0-F). Each character represents one *time-unit*. The information coded in one time-unit is the following:

- Bit0 is set: there are recorded frames within time unit
- Bit1 is set: there are event-triggered recordings within time unit
- Bit2 is set: frames within time unit are protected against removal
- Bit3: reserved for future use

Values should be extracted using bit operations.

Parameters: *SessionID* – returned by OpenSession

EntityID – EntityID of the camera

Unit – Specifies time unit on the timeline. It can be "Day" or "Minute". When omitting this parameter, the server defines unit automatically as follows: If timeline longer than 1 day then unit is "Day" else unit is "Minute".

<*DateTimeFilter*> or <*MillisFilter*> elements specify the time interval requested. Start of interval is rounded down; end of interval is rounded up to the next day.

Response: <*VideoArchiveMap*> on success or <*ExecutionStatus*> on error.

XML Example for the request GetVideoArchiveMap

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetVideoArchiveMap SessionID="8325" EntityID="7" Unit="Minute">
        <DateTimeFilter Start="2005-02-01T00:00:00" Duration="P2D"/>
    </GetVideoArchiveMap>
</SNAP>
```

XML Examples for response VideoArchiveMap

Response for a Map request for 2 days, staring at 2005-02-01, unit is “Minutes”.

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <VideoArchiveMap EntityID=”7” StartDateTime=”2005-02-01T00:00:00”  Unit=”Minute”>
        <MapItem FrameCount=”3421”>
            00000000011111111113111100011101111000011311111111111 11100011111</MapItem>
        <MapItem FrameCount=”642”>
            00000000000000000000000000000000033301131111111111 11100011000</MapItem>
        <MapItem FrameCount=”1235”>
            11111111111111111000000000000000000000003111111111111 11100011111</MapItem>
                .
                . 44 items
                .
        <MapItem FrameCount=”4567”>
            11000011011111111113111100011101111000011001111111111 11100011001</MapItem>
    </VideoArchiveMap >
</SNAP>
```

Response for a Map request for 1 year, staring at 2005-01-01, unit is “Day”.

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <VideoArchiveMap EntityID=”7” StartDateTime=”2005-01-01T00:00:00”  Unit=”Day”>
        <MapItem FrameCount=”1663421”>
            00000000011111111113111100011110</MapItem>
        <MapItem FrameCount=”445512”>
            00000000000000000000000000000000</MapItem>
        <MapItem FrameCount=”34523”>
            11111111111111111000000000000000</MapItem>
                .
                . 8 items
                .
        <MapItem FrameCount=”112567”>
            11000011011111111113111100011110</MapItem>
    </VideoArchiveMap >
</ SNAP>
```

4.12 VideoStreamAnnotation

Description: This request is used for annotating live or archive video streams with text.
For backward compatibility reasons we also kept the “AddAnnotationToVideoStream” request. The parameters from ForegroundColor to BlinkDuration are not yet supported on server-side.

Parameters: *SessionID* – Returned by OpenSession

AnnotationID – User declared id for future identification

CameraID – ID of the camera for the annotation

Text – Annotation text

Action – “Start” or “Stop”, default: “Start”

Destination – “Live”, “Archive” or LiveAndArchive”, default: “Live”

ForegroundColor – Color of the text, default: “White”. Possible formats are “RGB0,0,0”, html style “#FFFFFF” or named color one of “White“, “Black“, “Green“, “Blue“, “Red“, “Orange“, “Gray“, “Darkgray“, “Lightgray“, “Magenta“, “Pink“, or “Cyan“

BackgroundColor– Color of the text background, default: not defined (transparent). Same format as foreground color.

FontType – Text font, default: “System”

FontSize – Text size, default: “12”

Alignment – Where to stick the box in the video image, default: “LowerCenter”. Possible values are: “UpperLeft”, “UpperRight”, “UpperCenter”, “LowerLeft”, “LowerRight”, “LowerCenter”

Margin – Size of the margin around the text in pixels, default: “5”

Wrapping – Boolean for wrapping the text or not, default: “false”

StartTimestamp – Timestamp when to start the annotation, default: now. You can define a timestamp in the ISO8601-compliant format ‘yyyy-MM-dd’T’HH:mm:ss.SSS’ or also in the standard Unix timestamp format

StopTimestamp – Timestamp when to stop the annotation.

StopAfterMillisec – Stop the annotation automatically after this value.

StopOnClick – Stop the annotation on a mouse click in the client GUI.
Caution: The annotation is unlimited if no StopTimestamp, StopAfterMillisec, or StopOnClick isdefined.

BlinkDuration – Blinks the annotation with this duration in millisec.

EventStorage – How to save the corresponding event in the event database.
Valid values are: “NoSave”, “SaveStart”, “SaveStop”, “SaveAll”.
Default: “SaveStart"

EventVisibility – How to show the corresponding event in the client GUI Event bars.
Valid values are: “NoShow”, “ShowStart”, “ShowStop”, “ShowAll”.
Default: “NoShow"

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <VideoStreamAnnotation SessionID =”1” CameraID=”1” Text=”Alert!” EndOnClick=”true” />
</SNAP>
```

4.13 ShowCameraInViewport

Description: This request brings up a view panel in the client and places a camera in raw/column position.

Parameters: *SessionID* – Returned by OpenSession

TargetIP –Address of the client on which the operation will be executed (can be an empty string)

TargetUser – Target machine is selected based on the name of the logged user (can be an empty string)

NOTE: TargetIP and TargetUser can not be empty at the same time

EntityID – The unique identifier of the selected camera

WindowID – Defines the ID of the Online Monitor window in which the operation will take place (should be -1 if not defined).

PanelName – Name of the panel in the client which will be used for showing the camera

RawIndex – Defines the raw in which the camera image will be shown (should be -1 if not defined)

ColumnIndex – Defines the column in which the camera image will be shown (should be -1 if not defined)

Response: *ExecutionStatus*

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <ShowCameraInViewport SessionID =”3456” TargetIP=”192.168.7.2” TargetUser=”” EntityID=”38”
        WindowID=”1” PanelName=”First Floor” RawIndex=”0” ColumnIndex=”0” / >
</SNAP>
```

4.14 ShowViewOfWindow

Description: This request brings up a view panel in the client.

Parameters: *SessionID* – Returned by OpenSession

TargetIP – Mandatory (but can be an empty string), address of the client on which the operation will be executed

TargetUser – Mandatory (but can be an empty string), target machine is selected based on the name of the logged user

NOTE: TargetIP and TargetUser can not be empty at the same time

WindowID – Mandatory (should be -1 if not defined), defines the ID of the Online Monitor window in which the operation will take place

PanelName – Mandatory, name of the panel in the client which will be used for showing the camera

Response: *ExecutionStatus*

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <ShowViewOfWindow SessionID =”3456” TargetIP=”192.168.7.2” TargetUser=””
        WindowID=”1” PanelName=”First Floor” / >
</SNAP>
```

4.15 StartSmartGuard

Description: This request starts the shuffling of a group of view panels in the client.

Parameters: *SessionID* – Returned by OpenSession

TargetIP – Address of the client on which the operation will be executed (can be an empty string)

TargetUser – Target machine is selected based on the name of the logged user (can be an empty string)

NOTE: TargetIP and TargetUser can not be empty at the same time

ViewGroupName – Mandatory, name of the group which will be shuffled

Response: *ExecutionStatus*

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <StartSmartGuard SessionID =”3456” TargetIP=”192.168.7.2” TargetUser=””
        ViewGroupName =”Group1” / >
</SNAP>
```

4.16 StopSmardGuard

Description: This request will stop a previously started shuffling of a group of view panels in the client.

Parameters: *SessionID* – Returned by OpenSession

TargetIP – Address of the client on which the operation will be executed (can be an empty string)

TargetUser – Target machine is selected based on the name of the logged user (can be an empty string)

NOTE: TargetIP and TargetUser can not be empty at the same time

Response: *ExecutionStatus*

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <StopSmartGuard SessionID =”1234” TargetIP=”192.168.7.2” TargetUser=”” / >
</SNAP>
```

4.17 GetEntityTree

Description: Get camera topology from the server. The camera tree is represented by a list of <*EntityTreeNode*> elements. Each <*EntityTreeNode*> element represents either a camera or a group (see *Type* attribute of <*EntityTreeNode* >). Each <*EntityTreeNode*> element has attributes *EntityID* and *ParentEntityID*. An *EntityID* is a unique ID (integer >= 1) of the camera or group on the session's Netavis server. The attribute *ParentEntityID* is either the ID of the group which contains this entity, or 0 in case of the root of the tree currently transmitted (not neccesarily the root of the camera tree managed by the server). The root <*EntityTreeNode*> element and any *mount-point* <*EntityTreeNode*> do contain a <*Host*> element, while other nodes do not contain <*Host*> element. A *mount-point* is an <*EntityTreeNode*> which has been imported from a remote Netavis server (exporter host) to the session's Netavis server. A *mount-point* can be a camera or a group. Note that the *EntityID* of a mounted camera or group on session's Netavis server may differ from that on the exporter server. If the connection to the remote host is alive then the value of element <*EntityTreeNode* ><*Host*><*Connected*> is "true". If the mount-point is a group and the <*Host*> is connected then all child elements under the mount-point are visible in the tree. If the <*Host*> is not connected (TCP/IP connection is broken or the remote host is not alive) then no child elements are incorporated under the mount-point and the mount-point's attributes *Name* and *Type* are not set (empty strings). Any change in the EntityTree (add/remove/modify nodes and change of connected status of mount points) is propagated to the SNAP client via <*EntityTreeChanged*>. The element <*EntityTreeChanged*> is pushed by the server in a Channel (response to <*ReadChannel*>, please see description further in the chapter) which is registered for this change propagation via <*ControlChannel*><*RegisterForDataChangeNotification*>. The SNAP client may reload the part of the tree which has been marked as changed by the attribute *RootEntityID* of <*EntityTreeChanged*>.

Parameters: SessionID – Returned by OpenSession

RootEntityID – EntityID of the root node of this request (default = 1)

RequestedDetails – Optional parameter for defining the additional information the user wants about camera entities. Possible values are as follows:

- IP_ADDRESS – to include the IP address of the camera,
- STATUS_CODES – to include the current status of the camera,
- PTZ_DETAILS – to include the PTZ related capabilities of the camera.

Sort – Optional parameter, if set to true sort the tree by entity name (default = false)

Response: <*EntityTree*> on success or <*ExecutionStatus*> on error.

XML Example for the request GetEntityTree

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetEntityTree SessionID=”8325” RequestedDetails=”PTZ_DETAILS" />
</SNAP>
```

XML Example for the response EntityTree

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<EntityTree>
    <EntityTreeNode EntityID=”1” ParentEntityID=”0”  Type=”Group”  Name=”My Root Group”>
        <Host>
            <Name>MyNetavisServerName</Name>
            <Address>192.168.0.12</Address>
            <HostID>23234235145346267</HostID>
            <Connected>true</Connected>
        </Host>
    </EntityTreeNode>
    <EntityTreeNode EntityID=”2” ParentEntityID=”1”  Type=”Camera”  Name=”My Camera”/>
        <PTZCapabilities>
            <SupportsPan>false</SupportsPan>
            <SupportsTilt>false</SupportsTilt>
            <SupportsZoom>false</SupportsZoom>
            <SupportsCenterPointMove>false</SupportsCenterPointMove>
            <SupportsContinuousMove>false</SupportsContinuousMove>
            <SupportsRelativeMove>false</SupportsRelativeMove>
            <SupportsAbsoluteMove>false</SupportsAbsoluteMove>
            <SupportsQueryPosition>false</SupportsQueryPosition>
            <SupportsPresetPositions>false</SupportsPresetPositions>
        </PTZCapabilities>
    <EntityTreeNode EntityID=”3” ParentEntityID=”1”  Type=”Group”  Name=”My Group”/>
    <EntityTreeNode EntityID=”4” ParentEntityID=”3”  Type=”Camera”  Name=”Partners Camera1”>
        <Host>
            <Name>PartnersRemoteNetavisServerName</Name>
            <Address>partner.dyndns.org:6080</Address>
            <HostID>456782898982221123</HostID>
            <Connected>true</Connected>
        </Host>
        <PTZCapabilities>
            <SupportsPan>false</SupportsPan>
            <SupportsTilt>false</SupportsTilt>
            <SupportsZoom>false</SupportsZoom>
            <SupportsCenterPointMove>false</SupportsCenterPointMove>
            <SupportsContinuousMove>false</SupportsContinuousMove>
            <SupportsRelativeMove>false</SupportsRelativeMove>
            <SupportsAbsoluteMove>false</SupportsAbsoluteMove>
            <SupportsQueryPosition>false</SupportsQueryPosition>
            <SupportsPresetPositions>false</SupportsPresetPositions>
        </PTZCapabilities>
    </EntityTreeNode>
    <EntityTreeNode EntityID=”5” ParentEntityID=”3”  Type=”Group”  Name=”Partners Group”>
        <Host>
            <Name>PartnersRemoteNetavisServerName</Name>
            <Address>partner.dyndns.org:6080</Address>
            <HostID>456782898982221123</HostID>
            <Connected>true</Connected>
        </Host>
    </EntityTreeNode>
    <EntityTreeNode EntityID=”6” ParentEntityID=”5”  Type=”Camera”  Name=”Partners Camera2”>
        <PTZCapabilities>
            <SupportsPan>false</SupportsPan>
            <SupportsTilt>false</SupportsTilt>
            <SupportsZoom>false</SupportsZoom>
            <SupportsCenterPointMove>false</SupportsCenterPointMove>
            <SupportsContinuousMove>false</SupportsContinuousMove>
            <SupportsRelativeMove>false</SupportsRelativeMove>
```

```
            <SupportsAbsoluteMove>false</SupportsAbsoluteMove>
            <SupportsQueryPosition>false</SupportsQueryPosition>
            <SupportsPresetPositions>false</SupportsPresetPositions>
        </PTZCapabilities>
    </EntityTreeNode>
    <EntityTreeNode EntityID=”7” ParentEntityID=”5”  Type=”Camera”  Name=”Partners Camera3”>
        <PTZCapabilities>
            <SupportsPan>false</SupportsPan>
            <SupportsTilt>false</SupportsTilt>
            <SupportsZoom>false</SupportsZoom>
            <SupportsCenterPointMove>false</SupportsCenterPointMove>
            <SupportsContinuousMove>false</SupportsContinuousMove>
            <SupportsRelativeMove>false</SupportsRelativeMove>
            <SupportsAbsoluteMove>false</SupportsAbsoluteMove>
            <SupportsQueryPosition>false</SupportsQueryPosition>
            <SupportsPresetPositions>false</SupportsPresetPositions>
        </PTZCapabilities>
    </EntityTreeNode>
</EntityTree>
</ SNAP>
```

4.18 GetIODeviceList

Description: This request returns the list of the available IO devices on the server.
Parameters: *SessionID* – Returned by OpenSession
Response: *<IODeviceList>* on success or *<ExecutionStatus>* on error.

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetIODeviceList SessionID =”3456”/ >
</SNAP>
```

XML Example of the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>???
    <IODevice DeviceID="0" DeviceName="DeviceName0" StatusID="0" StatusText="StatusText0">
      <IOPort PortID="0" PortType="PortType0" PortValue="PortValue0"/>
   </IODevice>
</IODeviceList>
</SNAP>
```

4.19 GetPTZPositionList

Description: This request simply returns the list of the selected camera's defined PTZ positions.
Parameters: *SessionID* – Returned by OpenSession
EntityID – The unique identifier of the selected camera
Response: <*PTZPositionList*> on success or <*ExecutionStatus*> on error.

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetPTZPositionList SessionID =”3456” EntityID=”38” / >
</SNAP>
```

XML Example for the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<PTZPositionList>
   <PTZPosition PresetPosID="1" EntityID="38" Name="Door" Comment="Blinden gasse" Pan="0" Tilt="0"
Zoom="0"/>
   <PTZPosition PresetPosID="2" EntityID="38" Name="Window" Comment="Blinden gasse" Pan="0"
Tilt="0" Zoom="0"/>
</PTZPositionList>
</SNAP>
```

4.20 GetPTZRouteList

Description: This request simply returns the list of the selected camera's defined PTZ routes.
Parameters: *SessionID* – Returned by OpenSession
EntityID – Mandatory, the unique identifier of the selected camera
Response: <*PTZRouteList*> on success or <*ExecutionStatus*> on error.

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetPTZRouteList SessionID =”3456” EntityID=”38” / >
</SNAP>
```

XML Example for the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<PTZRouteList>
   <PTZRoute EntityID="38" RouteID="1" Name="Outside" Comment=""/>
</PTZRouteList>
</SNAP>
```

4.21 GetPTZRouteDefinition

Description: This request returns the list of the selected camera's PTZ route definition.

Parameters: *SessionID* – Returned by OpenSession

EntityID – The unique identifier of the selected camera

RouteID – The unique identifier of the selected route, given by the GetPTZRoutesList

Response: <*PTZRouteDefinition*> on success or <*ExecutionStatus*> on error XML Schema of the request

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <GetPTZRouteDefinition SessionID =”3456” EntityID=”38” RouteID=”1” / >
</SNAP>
```

XML Example for the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
<PTZRouteDefinition>
   <PTZRouteDefinitionElement EntityID="38" RouteID="1" PresetPosID="1" StayOnTargetTime="10000"
MoveToTargetTime="0" Sequence="0"/>
   < PTZRouteDefinitionElement EntityID="38" RouteID="1" PresetPosID="2" StayOnTargetTime="20000"
MoveToTargetTime="0" Sequence="0"/>
</PTZRouteDefinition>
</SNAP>
```

4.22 LockPTZResource

Description: This request allocates the related PTZ resource for further PTZ operations.
Parameters: *SessionID* – Returned by OpenSession
EntityID – The unique identifier of the selected camera
Response: ExecutionStatus

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <LockPTZResource SessionID =”3456” EntityID=”38” / >
</SNAP>
```

4.23 RefreshPTZResource

Description: This request keeps locking on the related PTZ resource.
Parameters: *SessionID* – Returned by OpenSession
EntityID – The unique identifier of the selected camera
Response: ExecutionStatus

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <RefreshPTZResource SessionID =”3456” EntityID=”38” / >
</SNAP>
```

4.24 ReleasePTZResource

Description: This request releases the exclusive access on the related PTZ resource.
Parameters: *SessionID* – Returned by OpenSession
EntityID – The unique identifier of the selected camera
Response: ExecutionStatus

XML Example for the request

```
<?xl version="1.0" encoding="UTF-8"?>
<SNAP>
    <ReleasePTZResource SessionID =”3456” EntityID=”38” / >
</SNAP>
```

4.25 StartPTZRoute

Description: This request starts the predefined PTZ routing on a camera.
Parameters: *SessionID* – Returned by OpenSession
EntityID – The unique identifier of the selected camera
RouteID – The unique identifier of the selected route
Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <StartPTZRoute SessionID =”3456” EntityID=”38” RouteID=”1” / >
</SNAP>
```

4.26 StopPTZRoute

Description: This request stops the predefined and related PTZ routing on a camera.
Parameters: *SessionID* – Returned by OpenSession
EntityID – Mandatory, the unique identifier of the selected camera
RouteID – Mandatory, the unique identifier of the selected route
Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <StopPTZRoute SessionID =”3456” EntityID=”38” RouteID=”1” / >
</SNAP>
```

4.27 SetPTZPosition

Description: This request directs the PTZ head of the camera to a preset position.

Parameters: *SessionID* – Returned by OpenSession

EntityID – The unique identifier of the selected camera

PositionID – Identifier of the position. Existing IDs can be obtained by the GetPTZPositionList command.-

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <SetPTZPosition SessionID =”3456” EntityID=”38” PositionID=”1” / >
</SNAP>
```

4.28 PTZCenterClick

Description: Move the defined (X,Y) position to the center of the picture.

Parameters: *SessionID* – Returned by OpenSession
EntityID – Mandatory, the unique identifier of the selected camera
X – X coordinate of the click relative to the top-right corner.
Y – Y coordinate of the click relative to the top-right corner.
ImageWidth – Width of the image on which the click happened.
ImageHeight – Height of the image on which the click happened.

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <PTZCenterClick SessionID =”3456” EntityID=”38” X=”100” Y=”50” ImageWidth=”320”
ImageHeight=”240” / >
</SNAP>
```

4.29 ContinuousPTZ

Description: Move the PTZ head in the given direction with given speed.

Parameters: *SessionID* – Returned by OpenSession
EntityID – Mandatory, the unique identifier of the selected camera
PanSpeed – Speed of pan. Valid values are from -100 to 100.
TiltSpeed – Speed of tilt. Valid values are from -100 to 100.
ZoomSpeed – Speed of zoom. Valid values are from -100 to 100.

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <ContinuousPTZ SessionID =”3456” EntityID=”38” PanSpeed=”10” TiltSpeed=”10” ZoomSpeed=”0”
ImageHeight=”240” / >
</SNAP>
```

4.30 MovePTZRelative

Description: Move the PTZ head in the given direction with given amount, relative to its current position.

Parameters: *SessionID* – Returned by OpenSession

EntityID – The unique identifier of the selected camera
PanDir – Horizontal movement direction: left, right.
PanSpeed – Speed of pan. Valid values are from 1, 2 or 3.
TiltDir – Vertical movement direction: up, down.
TiltSpeed – Speed of tilt. Valid values are from 1, 2 or 3.
ZoomDir – Zoom direction: in, out.
ZoomSpeed – Speed of zoom. Valid values are from 1, 2 or 3.

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
    <MovePTZRelative SessionID =”3456” EntityID=”38” PanDir=”left” PanSpeed=”1” TiltDir=”up” TiltSpeed=”1” ZoomDir=”in” ZoomSpeed=”1” / >
</SNAP>
```

4.31 GetLicensePlateLists

Description: Retrieve the saved license plate lists from the server.

Parameters: *SessionID* – Returned by OpenSession.

Response: *<LicensePlateLists>* on success or *<ExecutionStatus>* on error with the specific error code.

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
   <GetLicensePlateLists SessionID="1234"/>
</SNAP>
```

XML Example for the response

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
   <LicensePlateLists>
      <LicensePlateList ListID="1">
         <Name>my_list1</Name>
         <IgnoreSeparators>true</IgnoreSeparators>
         <Tolerance>1</Tolerance>
         <LicensePlates>
            <LicensePlate>ACI-142</LicensePlate>
            <LicensePlate>BBC-132</LicensePlate>
            <LicensePlate>BOBO-00</LicensePlate>
         </LicensePlates>
      </LicensePlateList>
      <LicensePlateList ListID="2">
         <Name>my_list2</Name>
         <IgnoreSeparators>true</IgnoreSeparators>
         <Tolerance>1</Tolerance>
         <LicensePlates>
            <LicensePlate>GHI-142</LicensePlate>
            <LicensePlate>BII-123</LicensePlate>
            <LicensePlate>BOBO-01</LicensePlate>
         </LicensePlates>
      </LicensePlateList>
   </LicensePlateLists>
</SNAP>
```

4.32 SetLicensePlateLists

Description: Set license plate lists to the desired values. Any element that is not present in the request will be ignored by the server (e.g.: omitting <Name> for a list will not change the name of the list). Attribute paramters are mandatory and thus cannot be omitted from the request.

Parameters: *SessionID* – Returned by OpenSession

<*LicensePlateLists*> – The license plate lists to set.

Response: ExecutionStatus

XML Example for the request

```
<?xml version="1.0" encoding="UTF-8"?>
<SNAP>
  <SetLicensePlateLists SessionID="1234">
    <LicensePlateLists>
      <LicensePlateList ListID="1">
        <Name>my_list1</Name>
        <IgnoreSeparators>true</IgnoreSeparators>
        <Tolerance>1</Tolerance>
        <LicensePlates>
          <LicensePlate>ACI-142</LicensePlate>
          <LicensePlate>BBC-132</LicensePlate>
          <LicensePlate>BOBO-00</LicensePlate>
        </LicensePlates>
      </LicensePlateList>
      <LicensePlateList ListID="2">
        <Name>my_list2</Name>
        <IgnoreSeparators>true</IgnoreSeparators>
        <Tolerance>1</Tolerance>
        <LicensePlates>
          <LicensePlate>GHI-142</LicensePlate>
          <LicensePlate>BII-123</LicensePlate>
          <LicensePlate>BOBO-01</LicensePlate>
        </LicensePlates>
      </LicensePlateList>
    </LicensePlateLists>
  </SetLicensePlateLists>
</SNAP>
```

Appendix A – List of events

This appendix describes the list of Events supported by the current SNAP implementation. Please refer to requests <StartLiveEventStream>, <StartArchiveEventStream> and <PropagateEvent> for more information about how to receive or send events from/to a Netavis server.

Note: CameraID used below can be obtained by requesting the camera tree by the SNAP request <GetEntityTree>, but it also appears in the Netavis standard client application in the Camera Administration Tool next to the camera name in parentheses.

The list of valid events and their actual parameters in Netavis may change from version to version. This information can be found and downloaded from each Netavis server via the web interface. Start an internet browser and enter the IP address of your Netavis host. Click on the **Customizer login** and then to the **Download configuration files** link. On the next page lookup and download the EMSAPIdoc.zip. After un-compressing the archive open the index.html file with your browser. All filterable events can be looked at in details after selecting the **server.event_manager.event** package.

CustomEvent is a special case where details for seding is different. Therefore, a detailed description is given here separately.

EventType: **CustomEvent** (used only for propagating external custom events to Observer)

Description: A custom event is an arbitrary event specified and sent by the SNAP client to the Netavis server. Besides standard events described above, any custom event can be propagated by the SNAP Client via <*PropagateEvent*>. Before sending a custom event, it needs to be registered. Both registration and sending has to be done via request <*PropagateEvent*>. The server does *register* a custom event having type *EventType (value of the **property** "EventType", see below in the table)* at the very first time, when the event is propagated. Registration of a custom event means that the server assigns *EventText* and parameter mappings to an *EventType*. Thus, the property *EventText* has to be included at the first time when the event is propagated, but it must be omitted (do not include the property named *EventText*) at any further event propagation. A custom event of type "*EventType"* will be **unregistered**, if an event with *empty EventText* is propagated. *Empty EventText* means, that the property with *Name EventText* is sent, but the *Value* element is empty or omitted).

*Note, that unregistering an event causes **removing all events** of this type from the event database. Propagating an event with a non-empty EventText, which is different from the actual registration, will replace the event text in the serve'rs event registration database (*Netavis *1.7 or higher).*

Any custom event may contain a maximum of 5 custom properties. At registration time, properties named "Param1" to "Param5" can be mapped to any *custom event parameter* name. The custom parameter name has to be appended to the name ParamN separated by a colon. (e.g.: "Param1:MyParamName") Custom parameter name must not contain any whitespaces. The custom parameter mapping is allowed only at first time, when the event is propagated (registration time). The server registers the parameter mapping, so any further custom events can be sent either by using the property name "ParamN" or using the custom name of it. Mapping of custom parameter name cannot be changed. Propagating an event with different parameter property mapping then the actual registration will cause an error (see error code 401 in Appendix C). Each custom parameter must have one of the following types: String, Integer, Float, Double or Boolean. The Value element of an <*EventProperty*> must match the type of the property.

***Important**: It is necessary to send the first event just for registration purpose. If the property named "EventText" is included, then the event will not be saved or handled, but it will be used for registration (if value element is not empty) or deregistration (if value element is empty).*

Properties:

Name	TYPE	DESCRIPTION
EventType	String	Name of the custom event type (e.g. “MyPlateNumberRecognition”)
EventText	String	Log text of the event (appears in the Observer event list). It may contain placeholders for event parameters which will be replaced by the actual value of the parameter. The format of the placeholder is %ParamN%, where N is a number between 1 and 5, or use the custom parameter name like %MyParamNName% if specified (see below). *Note: if the value of this property is empty then the event of type EventType will be unregistered, thus all events will be removed from the database!*
EventStamp	Long	Time stamp of the event. (millis since epoch, 1.jan.1970). Use the attribute *MillisStamp* of a video frame if the event is generated by analyzing a frame.
CameraID	Integer	EntityID of a camera, if the event is camera specific. The property can be omitted in case of non-camera specific events. If this property is specified then Netavis will try to retrieve the first archive frame having timestamp >= *EventStamp,* whenever the user clicks on the event in the event list or in the event bar.
Param1:MyParam1Name	legalParamType	Generic event parameter with optional custom mapping
Param2:MyParam2Name	legalParamType	Generic event parameter with optional custom mapping
Param3:MyParam3Name	legalParamType	Generic event parameter with optional custom mapping
Param4:MyParam4Name	legalParamType	Generic event parameter with optional custom mapping
Param5:MyParam5Name	legalParamType	Generic event parameter with optional custom mapping

LegalParamTypes are: String, Integer, Long, Float, Double or Boolean.

Appendix B – List of error codes

The codes and error texts below are returned by the server in response ExecutionStatus.

Code	Error Text
0	No error. Execution OK.
10	XML format error: %
101	Cannot open SNAP session (no SNAP license available).
102	Cannot open SNAP session (no more user licenses available).
103	Illegal user/passwd. User=%
104	Illegal request received from IP %, session has been opened form IP %.
200	Session not found. SessionID=%
201	Channel not found. ChannelID=%
202	Entity not found. EntityID=%
203	Entity is not a camera. EntityID=%
204	Stream not found. ChannelID=% StreamID=%
205	No camera access right. EntityID=% User=%
206	No event access right. User=%
207	Resource locked by another user. Resource=% User=%
208	Server is busy (or can not take any more URL connections)
209	Server has reached the max number of configured transcoders
210	Session user has no right to access license plate lists
211	No license plate list found for the specified *ListID*
212	Command not available on an NNS slave
300	TimeLimit out of range. TimeLimit=%
301	DataLimit out of range. DataLimit=%
400	Illegal EventType. EventType=%
401	Illegal event/action property name. EventActionType=% PropertyName=%
402	Illegal event/action property value. PropertyName=% PropertyValue=%
403	Illegal event/action property type. PropertyName=% PropertyType=%
404	Missing property EventText for Costom Event registration. EventType=%
405	Illegal ActionType. ActionType=%
406	Missing event/action property. EventActionType=% PropertyName=%
500	Illegal stream property. PropertyName=% PropertyValue=%
600	File operation ERROR: %
601	Resource lock operation ERROR: %

Appendix C – List of available Actions

This appendix describes the list of Actions supported by the current SNAP specification. Please refer to request <PerformAction> for more information about how to perform an action via SNAP.

Note: CameraID used below can be obtained by requesting the camera tree via the SNAP request <GetEntityTree> and it also appears in the standard Netavis Observer client application in Camera administration next to the camera name in parentheses.

ActionType: **SetCameraRecordingState** *(implemented in* Netavis *Observer R1.7 or later)*

Description: Control recording state of a camera. The functions “Start” and “Stop” do starting and stopping recording immediately. If the camera is just recording with different parameters (scheduled or MD triggered) then recording parameters will be changed to those specified in this request. Recording parameters must be specified as described in the table below. The property CameraID is mandatory. The properties VideoFormat, FrameSize, FrameQuality and FrameRate are required if function is “Start”.

Properties:

Name	TYPE	Description
CameraID	Integer	Unique camera ID (EntityID)
Function	String	“Start” or “Stop”
VideoFormat	String	Valid values: • “JPEG” – record MJPEG stream • “MPEG4” – record MPEG4 stream (Netavis.1.8 or later) • “MXPEG” – record MxPEG stream (Netavis 4.3 or later) • “H264” – record H.264 stream (Netavis 4.3 or later)
FrameSize	String	The following values are valid: • “Large” -- 4CIF (704x576) or full-size VGA (640x480) • “Medium” -- CIF (352x288) or VGA (320x240) • “Small” -- QCIF (176x144) or VGA (160x120)
FrameQuality	String	The following values are valid: • “High” -- Best quality (lowest compression) • “Medium” – Medium quality (medium compression) • “Low” -- Lowest quality (highest compression)
FrameRate	Float	Frames Per Second value. A float number between 0.001 and 30.0.
Duration	Integer	Duration of recording in seconds. Recording will be

		stopped automatically after given seconds. This property is optional.