Accellence Technologies GmbH

Universal Video Management System

vimacc SDK

TCP Control Interface

VIMACC_CONTROL interface specification
to control a **vimacc** system by a
higher level management system

Protocol version: **1.10.5.3**

Legal Notice

Publisher

Company:	Accellence Technologies GmbH
Commercial register:	HRB 110799 Hannover
Managing director:	Dipl.-Inf. (FH) Frank Christ, Dr.-Ing. Heinz Stephanblome
Editor:	Torsten Heinrich, Mike Plötz
Phone:	+49 (0)511 277 2400
Fax:	+49 (0)511 277 2499
email:	info@accellence.de
Web:	www.accellence.de / www.vimacc.de
Address:	Accellence Technologies GmbH Garbsener Landstrasse 10, 30419 Hannover, Germany

 Date: 29 December 2020

Table of Contents

List of Abbreviations

ASCII	American Standard Code for Information Interchange
AAC	Advanced Audio Coding
AES	Advanced Encryption Standard
CA	Certificate Authority or Certification Authority
CCTV	Closed Circuit Television
DB	Database
DCOM	Distributed Component Object Model
DVR	Digital Video Recorder
IP	Internet Protocol
iSCSI	Internet Small Computer System Interface
GUI	Graphical User Interface
GOP	Group of Pictures
HID	Human Interface Device
LDAP	Lightweight Directory Access Protocol
MMI	Man Machine Interface
NAS	Network Attached Storage (Fibre Channel, iSCSI, …)
NFR	Non-functional Requirement
NTP	Network Time Protocol
NVR	Network Video Recorder
OPC	OLE for Process Control
RFC	Request for Comments
RTSP	RealTime Streaming Protocol
PTZ / SNZ	Pan Tilt Zoom
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
SAN	Storage Area Network (CIFS, NFS, SMB etc.)
SAS	Serial Attached SCSI
SDP	Session Description Protocol (see RFC 4566)
SHA	Secure Hash Algorithm
SQL	Structured Query Language
SRTP	Secure Real-Time Transport Protocol
SSD	Solid State Drive
SSL	Secure Sockets Layer
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 Introduction

1.1 Document Purpose

This document is part of the system documentation of the **vimacc®** video management system from Accellence Technologies GmbH.

It describes the control and streaming interfaces of a **vimacc®** system to higher level management systems.

The three digits on the document cover refer to the version number of the **vimacc®** control protocol. The last digit represents the version of the document itself, (example: 1.10.5.2 -> Protocol version 1.10.5, revision 2).

1.2 Documentation Structure

The **vimacc®** system documentation comprises several documents dealing with different sub-aspects.

The following documents are available by default:

- **vimacc®** System Documentation: Introduction
  Overview of the general features and respective fields of application
- **vimacc®** System Documentation: Features and Characteristics
  Detailed description of technical performance parameters and features/characteristics
- **vimacc®** System Documentation: System Requirements
  Information on minimum requirements for hardware and operating system
- **vimacc®** System Documentation: System Design
  Conditions to be met when planning a video system and assistance with dimensioning the complete system
- **vimacc®** System Documentation: List of Video Sources
  List of available video sources (cameras, encoders), drivers and other connectable peripherals
- **vimacc®** System Documentation: Architecture
  Detailed overview of the architecture (internal document)

1.3 vimacc Editions

The video management software **vimacc®** is provided in different versions which differ in function scope and operation purpose.

vimacc® Professional
- video management system for installation on a single server or PC
- supports up to 64 cameras in full resolution and with full framerate (requires appropriate hardware equipment)
- up to 2 additional, standalone workstations
- simultaneous display of live, playback and alarm videos as well as site layouts
- unlimited number of monitors per workstation
- stores videos in original format
- export of videos to CD/DVD or USB incl. metadata
- integrated video wall functions
- time-synchronized playback at up to 1000 times the normal speed
- 100-fold digital zoom in live and playback videos
- interactive and stackable site layouts
- private zone masking with 2 security levels
- video system control via HTTPS or TCP SDK
- audio recording/playback

vimacc® Enterprise
- supports all **vimacc®** Professional functions
- number of cameras and workstations not limited by software
- decentralized installation of vimacc processes possible
- server/service redundancy and load balancing
- can be integrated into domain infrastructures like Active Directory
- can be connected to external systems (e.g. guide systems, Interkom)

vimacc® OA video subsystem for SCADA systems like SIMATIC WinCC OA
- supports all **vimacc®** Enterprise functions
- No user interface for operation. Integrates completely and transparently into a higher level SCADA system.

vimacc® Parking is a special edition for car park operators
- supports all **vimacc®** Enterprise functions
- deep integration with Commend/Schneider intercom systems incl. speech channel and door/gate control

vimacc® Safe Office for office environments with increased risk potential
- supports all **vimacc®** Enterprise functions
- supports alarm sensors to activate silent alarms
- automatic display of alarm images in neighbouring offices and/or at security services

vimacc® Visitor Management for active guidance of persons within buildings
- supports all **vimacc®** Professional functions
- additionally: direct two-way communication with respective cameras

2 Control Interfaces

2.1 Overview:

vimacc® supports different data interfaces to establish connections between streaming sources and display processes or to route live/playback streaming data to third party processes.

The functions provided by **vimacc®** are assigned to several communication channels (CONTROL, LIVE, PLAYBACK) according to Figure 2.1:

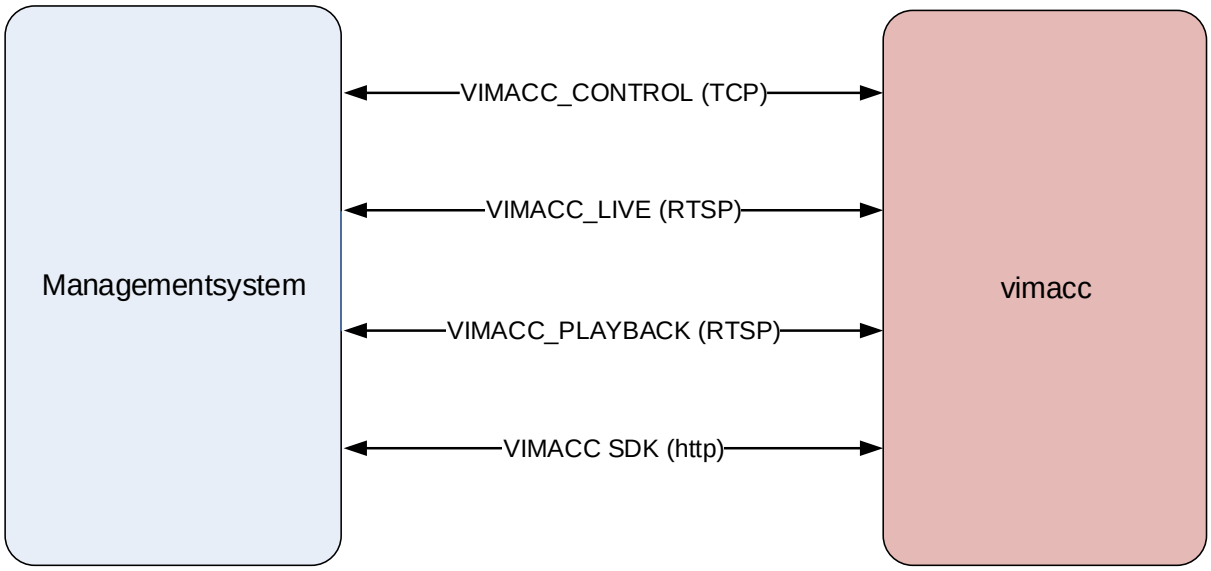

Figure 2.1: vimacc SDK – interfaces

VIMACC_CONTROL (TCP): Standardized control of connections for live presentation and archive access, control of devices like PTZ cameras, I/O contacts, signalling of events, loading and running of scripts and reading device ID lists and device states.

VIMACC_LIVE (RTSP): Access to live streams of digital video devices like network cameras or video recorders.

VIMACC_PLAYBACK (RTSP): Access to recorded streaming data for subsequent external processing, e.g. post analyses processes, playback, etc.

VIMACC SDK (http): The control and streaming interfaces via the HTTP protocol are described in a separate document.
See ***vimacc_Systemdokumentation_HTTP_Interface.pdf***

The term Management System (MMS) comprises all disciplines that implement a pairing with **vimacc®** like control centre systems, etc.

3 VIMACC_CONTROL

3.1 General

The VIMACC_CONTROL data interface enables a **vimacc®** system to exchange data with an external system. Connections between video sources and display processes can be established. In case of certain sources (e.g. PTZ cameras) additional control commands can be sent.

The VIMACC_CONTROL data interface is implemented as a simple, stateless interface. This means that **vimacc®** does not store any states of the established connections at the interface and that the controlling instance is always responsible for disconnecting.

Following the topology of the target system where the **vimacc®** system is operated as structured subsystem, **vimacc®** control interfaces can be installed and executed in each partial segment.

On **vimacc®** level, all control interfaces are equally important, i.e. each single interface is able to control the processes of the entire **vimacc®** system. The coordination of these processes (access rights, priority rules, etc.) has to be performed by the higher level management system respectively higher level control instance.

The VIMACC_CONTROL interface for controlling a **vimacc®** system is available in different versions that support different function sets.

The availability of the respective variant is defined in the licence file and has to be configured accordingly in the AccVimaccAdministrationCenter, see **vimacc®** Administrator's Guide.

The following protocol variants are defined:

- **VIMACC_CONTROL_BASIC**
  This protocol represents the simplest form of the VIMACC_CONTROL control interface.
  It contains basic control commands to establish live video connections and query device states.

- **VIMACC_CONTROL_DEVICES_ALARMS_SCENARIOS**
  This protocol adds commands for connecting scenarios, passing on alarms to the **vimacc®** system and accepting/completing generated alarms to the VIMACC_CONTROL_BASIC protocol.

- **VIMACC_CONTROL_ALL**
  This protocol adds commands for writing so called "data points" of the **vimacc®** systems to the VIMACC_CONTROL_DEVICES_ ALARMS_SCENARIOS protocol.

- **VIMACC_CONTROL_FALLBACK**
  Supports only a limited set of commands to query some information about the status of a **vimacc®** system. This protocol is automatically enabled if a **vimacc®** system can no longer be properly operated e.g. due to an expired licence.

The VIMACC_CONTROL interface cannot be used to directly access the streaming data.

3.2 Connection Setup

An MMS and/or higher level control instance may establish any number of TCP connections to the servers of the **vimacc®** system on which a **vimacc®** control interface is running, if required.
vimacc® does not limit the number of simultaneous connections.

An MMS and/or higher level instance should monitor the established TCP connections via test telegrams. If a connection error is detected, the MMS should close the connection and try to restart it after a few seconds.

3.3 Network and Transport Layer

- Network layer: IP
- Transport layer: TCP

3.4 Presentation and Session Layer

- Session layer: TCP socket connections
- Presentation layer: Plain text, UTF-8 encoded

3.5 Application Layer

The VIMACC_CONTROL_BASIC data interface serves as text-based protocol for the application layer.

vimacc® runs on all servers operated as communication entities for the MMS a process that keeps a TCP server in list mode on a configurable port. The MMS can establish a TCP connection to the process addresses consisting of IP address and port number, any time.

After a connection has been successfully established, the **vimacc®** TCP server displays the name of the released protocol (see above), the version number of the protocol and the version number of the vimacc system.

vimacc→MMS:
`<Protokoll-Name>:Version <Versionsnummer>;vimacc:Version <Versionsnummer>`

Then, the MMS initiates a protocol session by authenticating to the **vimacc®** system (see chapter 3.7.2).

After the session has been initiated, the MMS may send control data in the form of text-based protocol commands via the TCP connection. Each protocol command is acknowledged by the respective **vimacc®** process with a text-based message. Acknowledgement may be asynchronously followed by the results of the command execution.

The protocol sequences are transmitted UTF-8 encoded via the TCP connection.
All protocol commands and messages are terminated by the `\r\n` escape sequence.

All parameters of the protocol commands and messages are interpreted as text strings.

3.6 Network Resources

The TCP server for the VIMACC_CONTROL_BASIC data interface can be accessed via a TCP port that can be configured during the project specific **vimacc®** installation. The default port is 4227.

3.7 Telegram Description

3.7.1 Structure of the Control Commands

The control data are transferred as ASCII strings and consist of key-value pairs linked by an equals sign ('=').

A control command always consists of a key-value pair followed by optional parameters separated from each other by a semicolon (';').
A control command always consists of the `cmd` keyword with the name of the command as value followed by a parameter list of any length.
Parameters are to be passed as key-value pair in the following form as well: `<Parameter-Name>=Wert`.

The order of the key-value pairs is not defined.

If spaces, special characters like ("), (\), (\r), (\n), and (\t)), punctuation marks like (;) or signs like (=) should be transmitted in the value of a parameter, they must be preceded by an escape character (\) (i.e. they have to be "escaped").

If key-value pairs should be transmitted in the value of a parameter as well, they have to be "escaped" several times (see command `writeDp` in chapter 3.7.5.2).

A full protocol command is always completed with the escape sequence `\r\n`.

Each control command is acknowledged by **vimacc®** with a response text. Responses are transmitted as ASCII strings as well.
Each response consists of several key-value pairs separated from each other by a semicolon (';').
They are always preceded by the `msgsize` parameter followed by the number of characters contained in the message.
This string is then followed by the keyword `resp`, the name of the control command as value and the list of parameters of the control command separated from each other by a semicolon (';'). Then, the answer is added.
The value of the `msgsize` parameter always contains the number of characters of the response text starting with the keyword `resp`.

Example:
MMS→vimacc:
`cmd=keepalive;userdata=1234`

vimacc→MMS:
`msgsize=38;resp=keepalive;userdata=1234;answer=ok`

Note:
The control interface is being continuously enhanced. Compatibility with previous protocol versions is taken into account. However, in case of existing commands or responses, additional parameters can be added. Thus, a protocol parser should be able to handle a control command response containing additional parameters.

In the following, the possible control commands and arguments are listed. In addition, the responses always returned by the remote station are specified.
For reasons of clarity, entry `msgsize=<len>;` is omitted.

3.7.2 Authentication

After a TCP connection has been successfully established, the authentication is performed. For authentication, the MMS as well as the **vimacc®** system use the same user name and password which are encrypted when transmitted via the network.
Authentication is performed on the base of the digest access authentication process as follows:

First, the MMS has to signalize to the **vimacc®** TCP server that is wants to log in.
Then, the **vimacc®** TCP server generates a random text (the so called 'server challenge') and returns it to the client. The client then has to build a text string consisting of this random text, the user name and the password and generate an MD5 hash value from it.
The ASCII format of the hexadecimal presentation of this hash value has to be sent to the TCP server for a check. The server will then allow or deny the login.

Sequence of commands and associated responses:

1. MMS→vimacc:
   `cmd=login;userdata=<text>`

   vimacc→MMS:
   `resp=login;userdata=<text>;answer=failed,access denied;`
   `serverchallenge=<random string>`

2. MMS→vimacc:
   `cmd=login;userdata=<text>;clientresponse=<Ascii(Md5(<usernam`
   `e>:<password>:<random string from serverchallenge>))>`

   vimacc→MMS:

   a) Client answer correct:
   `resp=login;userdata=<text>;clientresponse=<Ascii(Md5(<userna`

```
me>:<password>:<random string from serverchallenge>))>;
answer=ok,access granted
```

b) Client answer not correct:

```
resp=login;userdata=<text>;clientresponse=Ascii(Md5(<usernam
e>:<password>:<random string from serverchallenge>));
answer=failed,access denied;serverchallenge=<random string>
```

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
The following login data is assumed:

User name: "UserName"

Password: "Password"

- MMS→vimacc:

```
cmd=login;userdata=1234
```

vimacc→MMS:

```
resp=login;userdata=1234;answer=failed,access denied;
serverchallenge=75798a683873f75071b7da939173f09a
```

- MMS→vimacc:

On client side, the following hash value has to be calculated:
Ascii(Md5("UserName:Password:75798a683873f75071b7da939173f09a")) = 792604ca7fb36e0177f24899e004590b

```
cmd=login;userdata=1234;clientresponse=792604ca7fb36e0177
f24899e004590b
```

vimacc→MMS:

```
resp=login;userdata=1234;clientresponse=792604ca7fb36e017
7f24899e004590b;answer=ok,access granted
```

3.7.3 VIMACC_CONTROL_BASIC

3.7.3.1 General

This protocol represents the simplest form of the VIMACC_CONTROL control interface.
It contains basic control commands to establish live video connections and to query device lists and states.

3.7.3.2 Querying Available Commands

Command:
`cmd=help;userdata=<text>`

Response:
`resp=help;userdata=<text>;answer=ok,parameterlist{\r\nbefehl#1 \r\n...befehl#n\r\n]}`

This command is used to query the control commands available in the installed protocol variant.
Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  `cmd=help;userdata=1234`

  vimacc→MMS:
  `resp=help;userdata=1234;answer=ok,parameterlist{\r\nlogin \r\nhelp\r\nkeepalive\r\nshow\r\n… subscribeevents\r\n}`

3.7.3.3 Monitoring the Control Connection

Command:
`cmd=keepalive;userdata=<text>`

Response:
`resp=keepalive;userdata=<text>;answer=ok`

The MMS should send a `keepalive` command every 5 seconds. If this command is not sent, **vimacc®** closes the control connection. The MMS then has to establish a new control connection and authenticate itself again.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  `cmd=keepalive;userdata=1234`
  vimacc→MMS:
  `resp=keepalive;userdata=1234;answer=ok`

3.7.3.4 Connecting Live or Playback Streams

Command:
`cmd=show;contextid=<text>;deviceid=<CameraID>;dest=<DisplayID>;videodlg=<number>;userdata=<text>`

Response:
`resp=show;contextid=<text>;deviceid=<CameraID>;dest=<DisplayID>;videodlg=<number>;userdata=<text>;answer=ok|failed`

This command can be used to connect streaming sources `(CameraID)` to display processes `(DisplayID)`. Display processes may be **vimacc®** Display instances as well as **vimacc®** Workstation instances.
Live as well as playback streams can be used as streaming sources.
Playback streams can be controlled by the `streamcontrol` command after connection (see chapter 3.7.3.9).

The `videodlg` parameter is used to address the number of a video quadrant of the workstation. Only integer values greater than or equal to 1 are supported.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Parameter `contextid` is not evaluated but used for logging and returned within the responses to ensure a better differentiation of the messages received.

The maximum number of video dialogs available and the arrangement in the display area depend on the **vimacc®** Display resp. workstation applications used.
Video dialogs are counted from top left to bottom right.

The following figure shows an example of a numbering scheme:

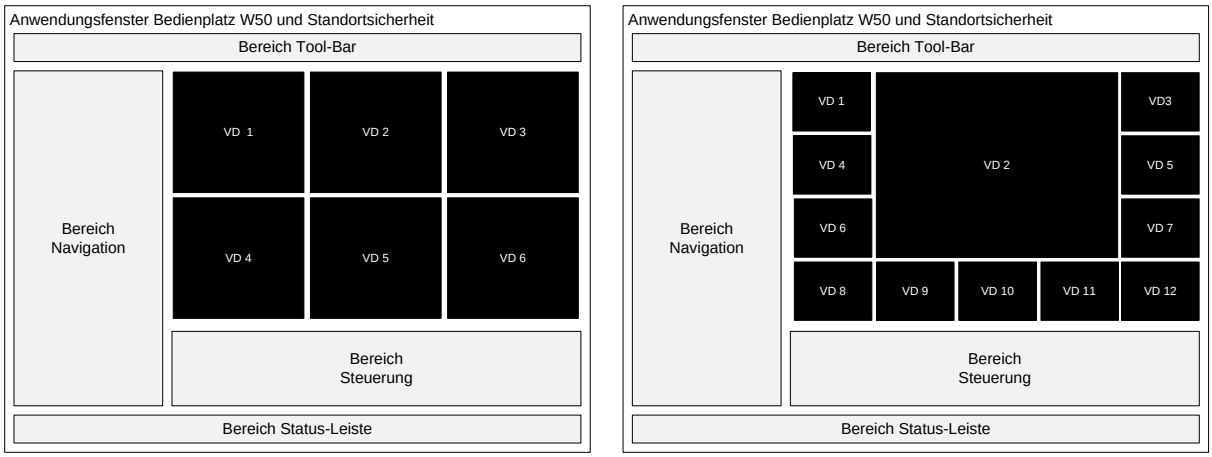

Figure 3.1: Numbering scheme of the video dialogs of a vimacc Workstation

Example:

- MMS→vimacc:
  `cmd=show;contextid=778;deviceid=Camera_0001;dest=AP_1;videodlg=5`

  vimacc→MMS:
  `resp=show;contextid=778;deviceid=Camera_0001;dest=AP_1;videodlg=5;answer=ok`

3.7.3.5 Disconnecting Live or Playback Streams

Command:
`cmd=clear;contextid=<text>;dest=<`*`DisplayID`*`>;videodlg=<`*`number`*`>;userdata=<text>`

Response:
`resp=clear;contextid=<text>;dest=<`*`DisplayID`*`>;videodlg=<`*`number`*`>;userdata=<text>;answer=ok|failed`

This command can be used to terminate the display of live and playback streams. This operation closes the streaming connection.

Parameter `dest` defines the ID of the display process, i.e. the ID of a **vimacc®** Display instance or **vimacc®** Workstation instance.

The `videodlg` parameter is used to address the number of a video quadrant of the workstation (→ command `cmd=show`). If value `0` is passed here, all video quadrants are deleted.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred

as assignment characteristic when the command is logged in the **vimacc** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  `cmd=clear;contextid=778;dest=AP_1;videodlg=5`

  vimacc→MMS:
  `resp=clear;contextid=778;dest=AP_1;videodlg=5;answer=ok`

3.7.3.6 Defining the Geometry of a vimacc Workstation Instance

Command:

`cmd=setworkstationgeometry;dest=<`*`DisplayID`*`>;xpos=<Wert>;ypos=<Wert>;width=<Wert>;height=<Wert>;showframe=<0|1>;topmost=<0|1>;hidden=<0|1>;shownavigation=<0|1>;showcontrols=<0|1>;title=<text>;contextid=<text>;userdata=<text>`

Response:

`resp=setworkstationgeometry;dest=<`*`DisplayID`*`>;xpos=<Wert>;ypos=<Wert>;width=<Wert>;height=<Wert>;showframe=<0|1>;topmost=<0|1>;hidden=<0|1>;shownavigation=<0|1>;showcontrols=<0|1>;title=<text>;contextid=<text>;userdata=<text>;answer=ok|failed`

This command can be used to define the position, size and look of a **vimacc** Workstation instance.

The parameters `xpos, ypos, width` and `height` define the geometrical properties of the application window of the **vimacc®** Workstation instance.

Parameter `showframe` defines whether the application is shown with or without a frame.

Parameter `shownavigation` defines whether the left navigation pane of the application should be shown.

Parameter `showcontrol` defines whether the lower area or the control elements of the application should be shown.

The `topmost` parameter defines whether the application window should overlay other windows.

Parameter `hidden` defines whether the application should be visible or not.

Parameter `title` defines the text that should be displayed in the header of the addressed **vimacc®** Workstation instance.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred as assignment characteristic when the command is logged in the **vimacc** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Note:
For this command, the parameters `xpos`, `ypos`, `width`, `height`, `showframe`, `showcontrols`, `shownavigation`, `topmost`, `hidden` and `title` are optional, i.e., not all parameters have to be transferred with the command. Command
`cmd=setworkstationgeometry;dest=4001;contextid=1234;title=Test`
and
`cmd=setworkstationgeometry;dest=4001;contextid=1234;hidden=1`
are supported.

Example:
MMS→vimacc:
```
cmd=setworkstationgeometry;dest=4001;xpos=0;ypos=0;width=1024;
height=768;showframe=1;topmost=1;hidden=0;shownavigation=1;sho
wcontrols=<0|1>;titel=Videomodul;contextid=1234
```

vimacc→MMS:
```
resp=setworkstationgeometry;dest=4001;xpos=0;ypos=0;width=1024
;height=768;showframe=1;topmost=1;shownavigation=1;showcontrol
s=1;hidden=0;titel=Videomodul;contextid=1234;answer=ok
```

The following figure shows the standard display mode of a **vimacc** workstation. This mode is also used when the following parameters are received: `showframe=1;shownavigation=1;showcontrols=1;`

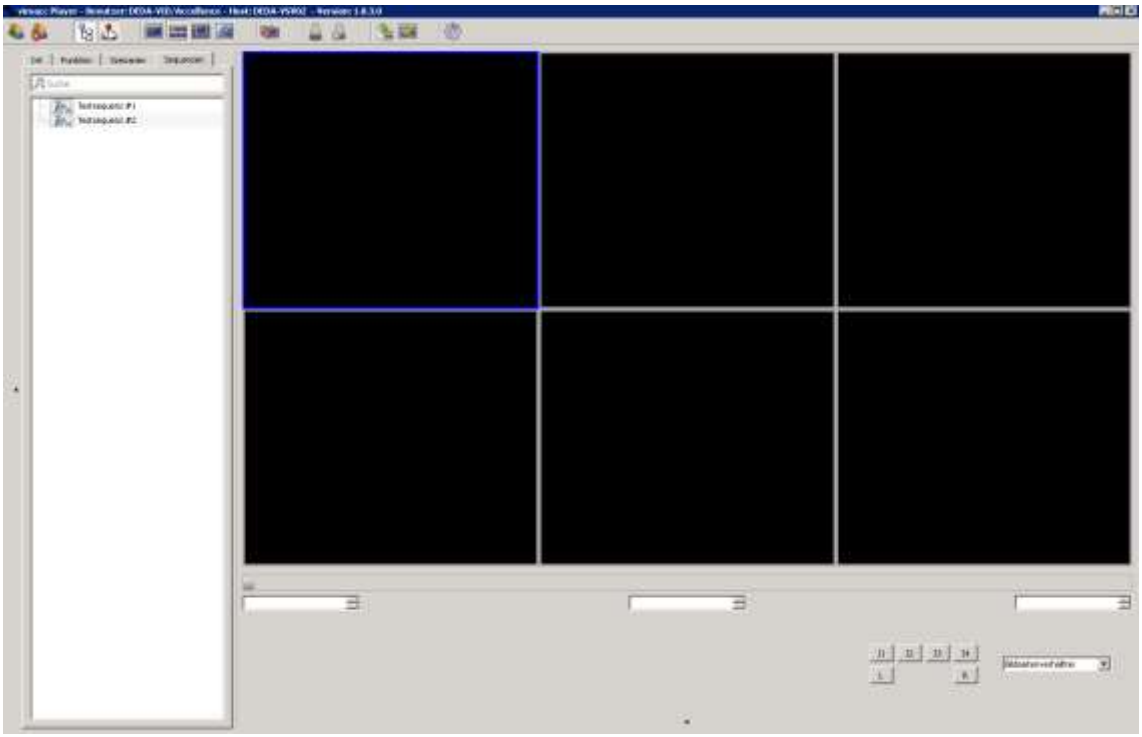

Figure 3.2: vimacc Workstation with frame, navigation pane and control elements

The following figure shows a **vimacc®** workstation instance after sending the parameters `showframe=1;shownavigation=0;showcontrols=1;`

Figure 3.3: vimacc Workstation with frame and control elements but without navigation pane

The following figure shows a **vimacc®** workstation instance after sending the parameters `showframe=1;shownavigation=0;showcontrols=0;`

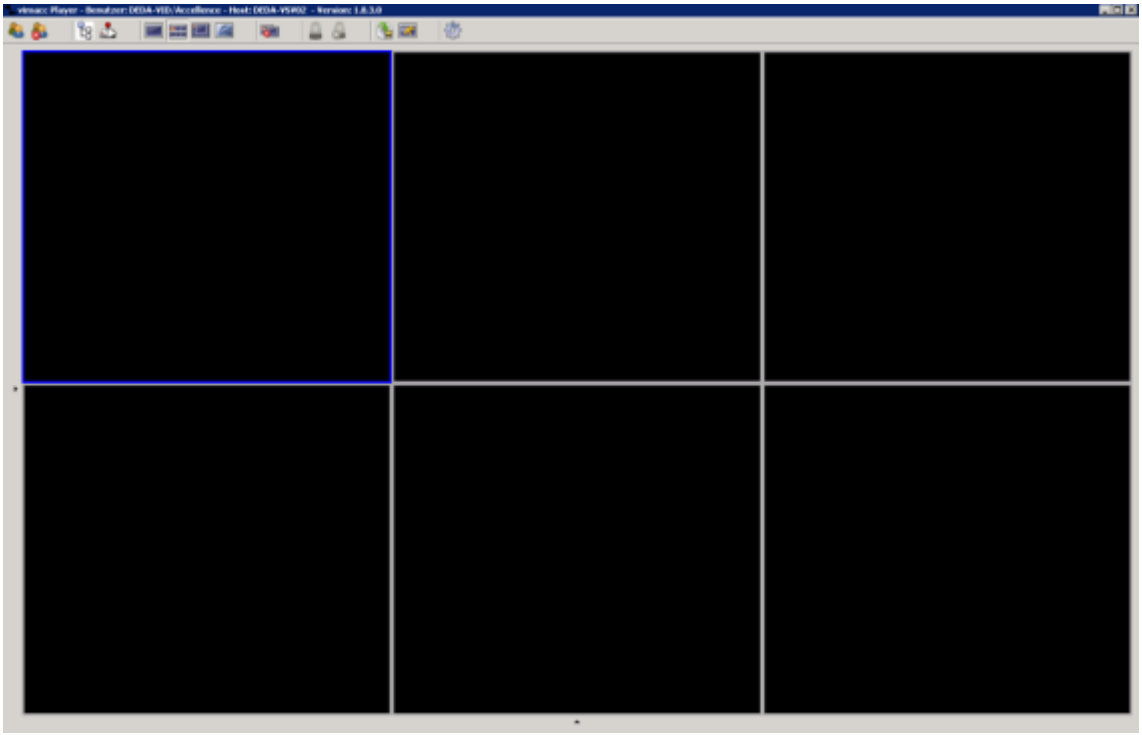

Figure 3.4: vimacc Workstation with frame but without navigation pane and control elements

The following figure shows a **vimacc®** workstation instance after sending the parameters `showframe=0;shownavigation=0;showcontrols=0;`

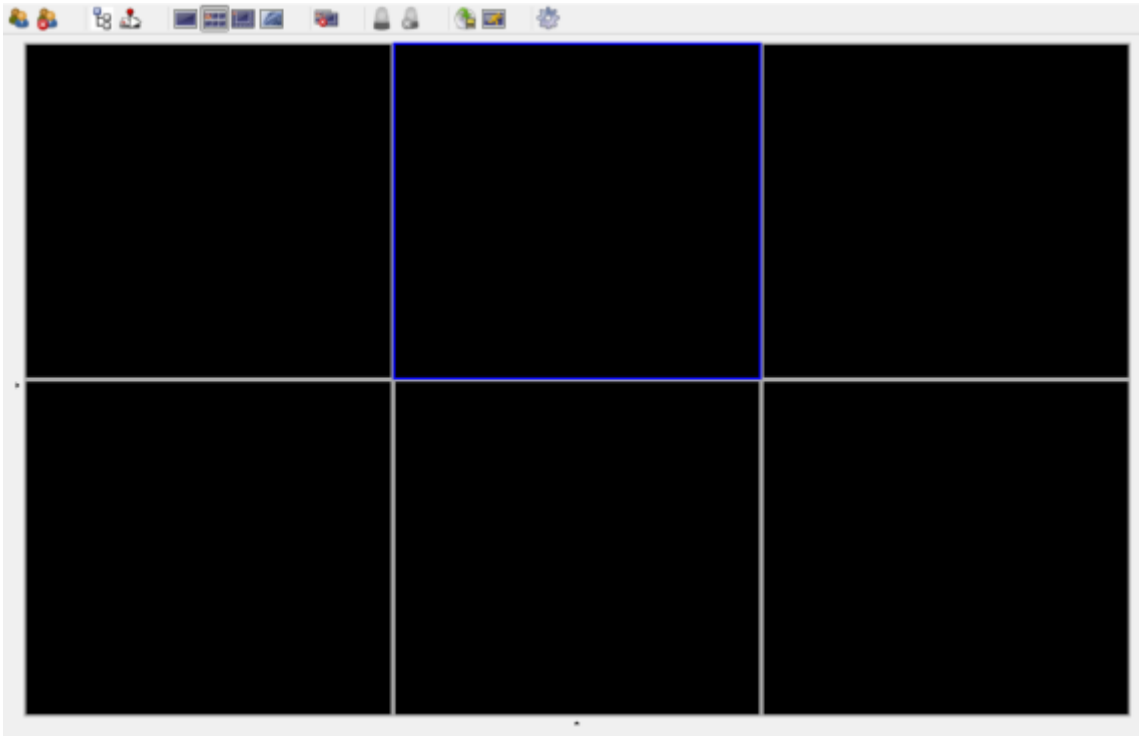

Figure 3.5: vimacc Workstation without frame, navigation pane and control elements

3.7.3.7 Defining the Arrangement of the Video Dialogs of a vimacc Workstation Instance

Command:

```
cmd=setworkstationgrid;dest=<DisplayID>;dialogcount<Wert>|grid
layout=<Layoutname>);clearunused=<0|1:D=0>;contextid=<text>;
userdata=<text>
```

Response:

```
resp=setworkstationgrid;dest=<DisplayID>;dialogcount<Wert>|gri
dlayout=<Layoutname>);clearunused=<0|1:D=0>;contextid=<text>;u
serdata=<text>; answer=ok|failed
```

This command can be used to define the arrangement of the **vimacc** Workstation instance video dialogs to be displayed.

Parameter `dialogcount` defines the number of video dialogs shown simultaneously. Depending on the **vimacc®** Workstation instance variant installed the associated layout is selected automatically. Figure 3.1 shows the default layout for the values `dialogcount=6` and `dialogcount=12` as an example.

Instead of the `dialogcount` parameter the `gridlayout` parameter can be used to specify the desired layout of the video dialogs. Depending on the **vimacc®** Workstation instance variant installed different layouts can be selected. By using the `gridlayout` parameter, arrangements can be selected that cannot be uniquely defined by simply specifying the number of dialogs. For example, `gridlayout=2x3` selects a two or three line layout which could not be clearly defined by simply specifying `dialogcount=6`. Figure 3.6 displays further possibilities for arranging video dialogs.

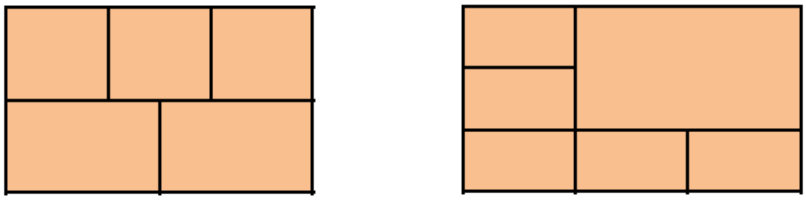

Figure 3.6: vimacc Workstation layouts: "3+2" and "5+1"

The maximum number of video dialogs that can be shown also depends on the installed variant of the **vimacc®** Workstation instance. Usually, **vimacc®** Workstation instances can display up to 24 video dialogs simultaneously.

Parameter `clearunused` defines whether existing connections that are no longer visible after switching to the desired layout should be terminated.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as

assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
MMS→vimacc:
```
cmd=setworkstationgrid;dest=1071;dialogcount=6;clearunused=1;c
ontextid=1234;userdata=test
```

vimacc→MMS:
```
resp=setworkstationgrid;dest=1071;dialogcount=6;clearunused=1;
contextid=1234;userdata=test;answer=ok
```

MMS→vimacc:
```
cmd=setworkstationgrid;dest=1071;gridlayout=2x3;clearunused=1;
contextid=1234;userdata=test
```

vimacc→MMS:
```
resp=setworkstationgrid;dest=1071;gridlayout=2x3;clearunused=1
;contextid=1234;userdata=test;answer=ok
```

3.7.3.8 Defining the Presentation within the Video Dialogs of a Workstation Instance

Command:
```
cmd=setworkstationscalemode;dest=<DisplayID>;cmdparam=<ZOOM|FI
T|KEEP>;contextid=<text>;userdata=<text>
```

Response:
```
resp=setworkstationscalemode;dest=<DisplayID>;cmdparam=<ZOOM|F
IT|KEEP>;contextid=<text>;userdata=<text>; answer=ok|failed
```

This command can be used to define the scaling behaviour of the video dialogs of a **vimacc®** Workstation instance.

The size of the individual video dialogs of a **vimacc®** Workstation instance depends on the application's window size which can be changed by the user but is limited by the maximum resolution of the output screen. Thus, the resulting side ratio of the video dialogs may change in such a way that it does no longer correspond to the side ratio of the transmitted video stream.
Command `setworkstationscalemode` can be used to define how the video image of the connected stream should be displayed in the video dialog.

The following values can be passed for the `cmdparam` parameter:

- `KEEP` Keeps the image side ratio of the video stream.

  The video stream is displayed in the video dialog without changing its image side ratio. Thus, depending on size and side ratio of the dialogs, the stream might have black borders.

- `ZOOM` Zooms to the size of the video dialog.

  The video image is scaled in such a way that it fits into the video dialog without showing any black borders. However, image areas might have been cut.

- `ZOOM` Adjusts to the size of the video dialog.

  The video image is horizontally or vertically scaled so that it fills the whole video dialog. Depending on the side ratio of the video dialog the video image might be more or less distorted.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred as assignment characteristic when the command is logged in the **vimacc** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
MMS→vimacc:
```
cmd=setworkstationscalemode;dest=1071;cmdparam=KEEP;contextid=1234;userdata=test
```

vimacc→MMS:
```
resp=setworkstationscalemode;dest=1071;cmdparam=KEEP;contextid=1234;userdata=test;answer=ok
```

3.7.3.9 Controlling a Playback Stream

Command:
```
cmd=streamcontrol;mode=playback;dest=<DisplayID>;streamcmd=<start | pause | speed | posu | posa | posr | stfw | strw | pint>; cmdparam=<parameter>;contextid=<text>;userdata=<text>
```

Response:

```
resp=streamcontrol;mode=playback;dest=<WorkstationId>;streamcm
d=<start | pause | speed | posu | posa | posr | stfw | strw |
pint>;cmdparam=<parameter>;contextid=<text>;userdata=<text>;an
swer=ok|failed
```

The playback stream is always controlled via the control of the **vimacc®** Display respectively Workstation instance to which the stream is connected by using the `show` command (see chapter).

As the **vimacc®** Display resp. Workstation instances always synchronize the playback streams currently connected, control command `streamcontrol` always effects all playback streams of the controlling instance. (Therefore, no video dialog number is specified for this command.)

Parameter `dest` defines the **vimacc®** ID of the display respectively workstation instance to be controlled.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred as assignment characteristic when the command is logged in the **vimacc** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

The respective control command is defined by using the `streamcmd` parameter. Valid keywords for this parameter are:

- `start` Starts the playback

  Optionally, the `cmdparam` parameter can be used to specify the speed in a range from -500 to +500. The default speed value +100.
  A positive value means that the stream is played in normal direction, a negative values means that the stream is played in reverse direction,
  e.g. `streamcmd=start;cmdparam=200` play forward with double speed

  `streamcmd=start;cmdparam=-200` play backward with double speed

- `pause` Pauses the playback.

  A still image is displayed if there is an image at the current position. Playback should be paused when searching with the position slider.

- `speed` Sets the playback speed.

  For parameter `cmdparam` a speed value between -500 and +500 is expected. The default speed value is +100.
  A positive value means that the stream is played in normal direction, a

negative values means that the stream is played in reverse direction,

- `posu` Subscribes to the position update every x milliseconds.

  For parameter `cmdparam` a specified interval `<intervall ms>` is expected.
  In the `events` data point of the **vimacc®** Display resp. Workstation instance (see chapter 3.7.3.20) the current playback position is reported every x milliseconds

  Notes:
  A value that is too low results in a high number of messages!

- `posa` Move to a specific time position.

  For parameter `cmdparam` a specified UTC time stamp is expected in the `yyyy-MM-dd'T'hh:mm:ss.zzz` notation.

- `posr` Relative positioning

  For parameter `cmdparam` an offset in the form of `Seconds.Milliseconds` is expected (`ss.zzz`).

- `stfw` Single image forward

  This command only makes sense when the state is `Pause`.

- `stfw` Single image backward

  This command only makes sense when the state is `Pause`.

- `pint <0|1>` Activates/deactivates pause interpretation

  For parameter `cmdparam` a value of 0 or 1 is expected.
  1: Show pauses at correct time
  0: Skip pauses

Example:

- MMS→vimacc:
  `cmd=streamcontrol;mode=playback;dest=1071;contextid=1234;userdata=test;streamcmd=start;cmdparam=500`

  vimacc→MMS:
  `resp=streamcontrol;mode=playback;dest=1071;contextid=1234;userdata=test;streamcmd=start;cmdparam=500;answer=ok`

- MMS→vimacc:
  `cmd=streamcontrol;mode=playback;dest=1071;contextid=1234;userdata=test;streamcmd=posa;cmdparam=2014-04-`

```
02T11:17:37.000
```

vimacc→MMS:

```
resp=streamcontrol;mode=playback;dest=1071;contextid=1234
;userdata=test;streamcmd=posa;cmdparam=2014-04-
02T11:17:37.000;answer=ok
```

3.7.3.10 Controlling a PTZ camera

Command:

```
cmd=<move|iris|focus>;<keyword>=<% speed>[;>;<keyword>=<%
speed>…];contextid=<text>;userdata=<text>;source=<CameraID>|ir
is|focus
```

Response:

```
resp=<move|iris|focus>;<keyword>=<% speed>[;>;<keyword>=<%
speed>…];contextid=<text>;userdata=<text>;source=<CameraID>;an
swer=ok|failed
```

In case of control commands for moving the PTZ mechanism of a PTZ camera, the ID of the respective camera has to be passed in the `source` parameter.

It is possible to specify movements for different axes at the same time which have to be separated by a semicolon.

In addition, commands not implemented in the respective camera can be executed. E.g. it might be possible that a camera does not support an automatic aperture control (see below). In this case, the command executed by **vimacc®** would not have any effect. However, using the interface described here the executed command would be positively acknowledged nevertheless.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Valid keywords for the <`keyword`>parameter of command `move` are:

- `up` pan up
- `down` pan down
- `left` pan left
- `right` pan right
- `zoomin` zoom in
- `zoomout` zoom out

The command is always followed by the speed ranging from 0 % to 100 %,

e.g. `up=80`

- `preset` Move to preset position

  The command is followed by the number of the preset position.

- `stop` Stop camera movement

  The `cmd=move;stop=1` command sends all keywords of the command with the O value, except for `preset`. Thus, this command string corresponds to the following command string:

  `cmd=move;up=0;left=0;down=0;right=0;zoomIn=0;zoomout=0`

Valid keywords for the <`keyword`> parameter of the command `iris` are:

- `start` manual aperture setting

  The command is followed by the speed ranging from -100 % to +100 % (`start` > 0 open aperture, `start` < 0 close aperture), e.g. `start=-80`

- `auto=1` Changes to Auto mode

- `stop=1` Stops aperture movement

Valid keywords for the <`keyword`>parameter of the command `focus` are:

- `step` Focus step by step

  The command is followed by the number of steps to be executed.

- `start` Changes to Auto mode

  The command is followed by the speed, ranging from 0 % to 100 %, e.g. `start=-70`

- `stop=1` Stops focussing

Example:

- MMS→vimacc:
  `cmd=move;up=75;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=move;up=75;contextid=1234;source=Camera_0001;answer=ok`

- MMS→vimacc:
  `cmd=move;up=20;left=45;down=20;contextid=1234;source=Camera_0001`

  vimacc→MMS:

```
resp=move;up=20;left=45;down=20;contextid=1234;source=Cam
era_0001;answer=ok
```

- MMS→vimacc:
  `cmd=move;stop=1;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=move;stop=1;contextid=1234;source=Camera_0001;answer=ok`

- MMS→vimacc:
  `cmd=iris;start=50;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=iris;start=50;contextid=1234;source=Camera_0001;answer=ok`
- MMS→vimacc:
  `cmd=iris;stop=1;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=iris;stop=1;contextid=1234;source=Camera_0001;answer=ok`

- MMS→vimacc:
  `cmd=focus;start=50;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=focus;start=50;contextid=1234;source=Camera_0001;answer=ok`

- MMS→vimacc:
  `cmd=focus;stop=1;contextid=1234;source=Camera_0001`

  vimacc→MMS:
  `resp=focus;stop=1;contextid=1234;source=Camera_0001;answer=ok`

3.7.3.11 Querying Configured Cameras

Command:
`cmd=getcameralist;metainfo=<0|1>;userdata=<text>`

Response:
`resp=getcameralist;metainfo=<0|1>;userdata=<text>;answer=ok,parameterlist{\r\nname=<name#1>\;id=<id#1>\;metainfo=<metainfo#1>>\r\n...name=<name#n>\;id=<id#n>\;metainfo=<metainfo#n>>\r\n}`

This command can be used to query the list of cameras configured within the **vimacc®** system.

Parameter `metainfo` defines whether the response should contain the metadata stored for the device.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each configured camera a line in the form of
`name=Kamera-Name;id=Kamera-Id`
or
`name=Kamera-Name;id=Kamera-Id;metainfo=Kamera-Metainfos`
is returned with the response text.

Example:
MMS→vimacc:
`cmd=getcameralist;metainfo=0;userdata=1234`

vimacc→MMS:
```
resp=getcameralist;userdata=1234;answer=ok,parameterlist{\r\nn
ame=cameraName#1\;id=cameraId#1\r\n
name=cameraName#2\;id=cameraId#2\r\n}
```

3.7.3.12 Querying Available Playback Streams

Command:
`cmd=getplaybacklist;mediatype=<video|audio>;userdata=<text>`

Response:
```
resp=getplaybackList;mediatype=<video|audio>;userdata=<text>;a
nswer=ok|failed,parameterlist{\r\nid=<id#1>;name=<name#1>;medi
atype=<type>;definition_parameter=<text#1>;function=<function#
1>;merge=<merge#1>;privacy=<privacy#1>;associations=<associati
ons#1>\r\n … id=<id#n>;name=<name#n>;...\r\n}
```

This command can be used to query the list of playback streams available within the **vimacc®** system.

The `mediatype` parameter defines the type of streams to be queried. The `video` value only queries video streams, the `audio` value only queries audio streams. If several media types should be queried, the respective `mediatype` values are separated by a comma (',').

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each available playback stream a line in the form of

```
id=<id>;name=<name>;mediatype=<type>;definition_parameter=<tex
t>;function=<function>;merge=<mergeID>;privacy=<privacy>;
associations=<associations>
```

is returned whereby not all of the parameters have to be used.

The following information is returned:

- `id`:
  ID for a unique reference to the playback stream.

- `name`:
  The name of the playback stream. This usually is the name of the camera (or device) that recorded the stream.

- `mediatype`:
  The media type of the playback stream, i.e. `video` or `audio`.

- `definition_parameter`:
  This parameter returns system specific configuration parameters specified during the configuration of the camera (resp. devices) that recorded the stream. The values are automatically transmitted to the playback stream by **vimacc®**.

- `function`:
  This parameter returns the system specific configuration parameter of the `function` data field that has been specified during the configuration of the camera (resp. the devices) that recorded the stream. The values are automatically transmitted to the playback stream by vimacc.

- `merge`:
  This parameter returns the system specific configuration parameter of the `merge` data field that has been specified during the configuration of the camera (resp. the devices) that recorded the stream. The values are automatically transmitted to the playback stream by **vimacc®**.

  Data field `merge` is used to mark associated playback tracks so that the *AccVimaccServer* can (`'merge'`) respective data when the stream is retrieved.

  This is e.g. useful when a regular archive recording with low refresh rate and an alarm recording with high refresh rate have been enabled for a camera. When the *AccVimaccServer* then retrieves the stream of a camera, it loads and combines both tracks and preferably plays the part with the higher refresh rate.

- `privacy`:
  This parameter returns the system specific configuration parameter of the `privacy` data field that has been specified during the configuration of the camera that recorded the stream. The values are automatically transferred to

the playback stream by **vimacc®** and return the private zones configured for this camera.

- `associations`:
  This parameter returns the system specific configuration parameter of the `associations` data field that has been specified during the configuration of the camera (resp. the devices) that recorded the stream. The values are automatically transferred to the playback stream by **vimacc®** and return the channels associated with this channel. Such an association might be an audio channel assigned to a video channel.

Example:
MMS→vimacc:

```
cmd=getplaybacklist;mediatype=video;userdata=1234
```

vimacc→MMS:

```
resp=getplaybacklist;mediatype=video;userdata=1234;answer=ok,p
arameterlist {
id=Showroom;mediatype=video;name=Showroom;function=PTZ,EMA
id=autobahn;mediatype=video;definition_parameter=playbackonly\
=true
id=cam0002;mediatype=video;definition_parameter=hidefromtopolo
gy\=1
id=cam0116_archiv;mediatype=video;name=Aussenhaut/Tor/Nord;fun
ction=FIX,Tor,Aussenhaut;merge=cam0116_prealarm
}
```

3.7.3.13 Querying Playback Sessions of a Playback Stream

Command:

```
cmd=getplaybacksessionsforplaybackid;playbackid=<id>;userdata=
<text>
```

Response:

```
resp=getplaybacksessionsforplaybackid;playbackid=<id>;userdata
=<text>;answer=ok|failed,parameterlist{\r\n
sessionid=<id#1>\;sessionname=<sessionname#1>\;controller=<con
troller#1>\;port=<port#1>\r\n …
sessionid=<id#1>\;sessionname=<sessionname#1>\;controller=<con
troller#1>\;port=<port#1>\r\n}
```

This command can be used to query the list of the playback sessions of a **vimacc®** playback stream.

vimacc® can record audio and video data on multiple servers. For example, in case of a server failure for a system with redundant servers, the recording is automatically continued on the redundant server. As a result, the content has to be loaded from two different servers when the recorded stream should be played. To assign the individual parts to a certain playback stream, so called sessions are used which are uniquely determined by an ID as well as the name and IP port of the recording server.

The sessions of a specific playback stream can be retrieved by using the `getplaybacksessionsforplaybackid` command whereby the `playbackid` parameter specifies the ID of the playback stream and thus references the clip to be played.

The list of available playback streams can be retrieved by using the `getplaybacklist` command (see chapter 3.7.3.12).

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each available playback session a line in the form of

```
sessionid=<id>\;sessionname=<sessionname>\;controller=<control
ler>\;port=<port>
```

is returned whereby not all of the parameters have to used.

The following information is returned:

- `sessionid`:
  ID for a unique reference to the playback session.

- `sessionname`:
  The name of the playback session.

- `controller`:
  The name of the recording server providing this playback session.

- `port`:
  The IP port used to retrieve the associated stream from the recording server.

Note:
This command is relevant only when a playback stream is retrieved by a playback object that directly communicates with the **vimacc®** recording servers via TCP/IP and retrieves the streams itself. One example for this is an autonomic **vimacc®** video widget (for an implementation sample see the EWO video at WinCC OA).
An autonomic video widget is not subjected to the **vimacc®** Rights Management as it is not connected to the central **vimacc®** configuration.

However, in a **vimacc®** system, the playback streams are only referenced with the ID of the stream and the **vimacc®** objects retrieve the different parts from the recoding servers autonomously.

Example:
MMS→vimacc:
```
cmd=getplaybacksessionsforplaybackid;playbackid=cam0230;userda
ta=1234
```

vimacc→MMS:
```
resp=getplaybacksessionsforplaybackid;playbackid=cam0230;userd
ata=1234;answer=ok,parameterlist {
sessionid=114;sessionname=REC1_PREALARM;controller=laptop-
heinrich;port=9371
}
```

3.7.3.14 Querying the Time Limits of a Playback Stream

Command:
```
cmd=getstreaminfo;playbackid=<playbackid>;userdata=<text>
```

Response:
```
resp=getstreaminfo;playbackid=<playbackid>;userdata=<text>;ans
wer=ok|failed;begintime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>;endtime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>
```

This command can be used to query the limits of a playback stream. Here, the absolute points in time for the beginning and the end of the stream are determined. If the recorded content is stored on several recording servers, the limits for the entire content are determined.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Parameter `playbackid` defines the ID of the playback stream whose limits should be determined.

The limits determined are returned as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

Example:
MMS→vimacc:
```
cmd=getstreaminfo;playbackid=cam0231_archiv;userdata=test
```

vimacc→MMS:
```
resp=getstreaminfo;playbackid=cam0231_archiv;userdata=test;ans
wer=ok;begintime=2014-04-01T11:42:19.246;endtime=2014-04-
01T11:52:26.196
```

3.7.3.15 Querying the Timeline of a Playback Stream

Command:

```
cmd=getstreamtimeline;playbackid=<playbackid>;userdata=<text>
```

Response:

```
resp=getstreamtimeline;playbackid=<playbackid>;userdata=<text>
;answer=ok|failed,parameterlist{\r\n
begintime=<timestamp>;endtime=<timestamp\r\n
…
begintime=<timestamp>;endtime=<timestamp\r\n
begintime_merged=<timestamp>;endtime_merged=<timestamp\r\n
…
begintime_merged=<timestamp>;endtime_merged=<timestamp
\r\n}
```

This command can be used to query the entire timeline of a playback stream. A so called timeline may consist of several related time ranges with different pause times in between.

This commands differentiates from the `getstreaminfo` command (see chapter 3.7.3.14) as follows: All related parts of the playback stream are determined and not only the limits of the entire time range are returned.

If the recorded content is stored on multiple recording servers, the time ranges are determined on all servers and all ranges found are returned.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Parameter `playbackid` defines the ID of the playback stream whose limits should be determined.

The limits determined are returned as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

For each related time range a line in the form of

```
begintime=<timestamp>\;endtime=<timestamp>\r\n
```

and/or

```
begintime_merged=<timestamp>\;endtime_merged=<timestamp>\r\n
```

is returned in the response text.

The ranges defined by `begintime` and `endtime` describe ranges in which recorded material has been found on the recording servers.
The ranges defined by `begintime_merged` and `endtime_merged` describe ranges in which recorded material has been found in further recording tracks. This e.g. applies to video recordings stored in the respective alarm recording track due to an alarm event.

As usually only one timeline is displayed when a stream is played in an application with user interface, the gathered information can be used to differentiate parts of an archive recording from parts of an alarm recording e.g. by colour.

Note:
This command puts a higher load on the recording servers as the `getstreaminfo` command (see chapter 3.7.3.14). Thus, if only the outer limits of a stream should be determined, command `getstreaminfo` has to be used.

Example:
MMS→vimacc:
```
cmd=getstreamtimeline;playbackid=cam0231_archiv;userdata=test
```

vimacc→MMS:
```
resp=getstreamtimeline;playbackid=cam0231_archiv;userdata=test
;answer=ok;parameterlist
{\r\nbegintime=2014-04-01T11:42:19.246\;endtime=2014-04-
01T12:49:38.727\r\n
begintime_merged=2014-04-01T11:50:51.197\;endtime=2014-04-
01T11:55:26.196\r\n}
```

3.7.3.16 Querying the Configured Monitors of the vimacc System

Command:
```
cmd=getmonitorlist;metainfo=<0|1>;userdata=<text>
```

Response:
```
resp=getmonitorlist;metainfo=<0|1>;userdata=<text>;answer=ok,p
arameterlist{\r\nname=<name#1>;id=<id#1><;metainfo=<metainfo#1
>>\r\n...name=<name#n>;id=<id#n><;metainfo=<metainfo#n>>\r\n}
```

This command can be used to query the list of Display instances configured within the **vimacc®** system.

Parameter `metainfo` defines whether the response should contain the metadata stored for the device.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each configured Display instance a line in the form of
`name=Display-Name;id=Display-Id`
or
`name=Display-Name;id=Display-Id;metainfo=Display-Metainfos`
is returned with the response text.

Example:

MMS→vimacc:
```
cmd=getmonitorlist;metainfo=0;userdata=1234
```

vimacc→MMS:
```
resp=getmonitorlist;metainfo=0;userdata=1234;answer=ok,paramet
erlist{\r\nname=displayName#1\;id=displayId#1\r\nname=displayN
ame#2\;id=displayId#2\r\n}
```

3.7.3.17 Querying the Configured Workstation Instances of the vimacc System

Command:
```
cmd=getworkstationlist;metainfo=0|1;userdata=<text>
```

Response:
```
resp=getworkstationlist;metainfo=<0|1>;userdata=<text>;answer=
ok,parameterlist{\r\nname=<name#1>;id=<id#1>\;metainfo=<metain
fo#1>>\r\n...name=<name#n>\;id=<id#n>\;metainfo=<metainfo#n>>\
r\n}
```

This command can be used to query the list of workstation instances configured within the **vimacc®** system.

Parameter `metainfo` defines whether the response should contain the metadata stored for the device.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each configured workstation instance a line in the form of
`name=Workstation-Name\;id=Workstation-Id`
or
`name=Workstation-Name\;id=Workstation-Id\;metainfo=Workstation-Metainfo`
is returned with the response text.

Example:
MMS→vimacc:
```
cmd=getworkstationlist;metainfo=0;userdata=1234
```

vimacc→MMS:
```
resp=getworkstationlist;metainfo=0;userdata=1234;answer=okpara
meterlist{\r\nname=AP#1\;id=workstationId#1\r\nname=
AP#2\;id=workstationId#2\r\n}
```

3.7.3.18 Querying the Configured Scenarios of the vimacc System

Command:

```
cmd=getscenariolist;userdata=<text>
```

Response:
```
resp=getscenariolist;userdata=<text>;answer=ok,parameterlist{\
r\nname=<name#1>\;id=<id#1>\r\n...name=<name#n>\;<id>=<id#n>\r
\n}
```

This command can be used to query the scenarios configured within the **vimacc®** system.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

For each configured scenario a line in the form of `name=Szenario-Name;id=Szenario-Id` is returned with the response text.

Example:
MMS→vimacc:
```
cmd=getscenariolist;userdata=1234
```

vimacc→MMS:
```
resp=getscenariolist;userdata=1234;answer=ok,parameterlist{\r\
nname=Szenario#1\;id=SzenarioId#1\r\nname=Szenario#2\;id=Szena
rioId#2\r\n}
```

3.7.3.19 Requesting Status Information from vimacc Devices

Command:
```
cmd=subscribedevicestatus;function=<text>;userdata=<text>
```

Response:
```
resp=subscribedevicestatus;function=<text>;userdata=<text>;ans
wer=ok|failed
```

Status message:
```
resp=devicestatus;userdata=<text>;deviceid=<deviceid>;property
=<property>;content=<status text>
```

This command can be used to subscribe to changes to the status information of the **vimacc®** device types.

Within a **vimacc®** system, devices are always parameterized with one or more function identifiers.
Identifiers might be:

- `PTZ` or `FIX` for PTZ resp. Fix cameras
- `HID` for human interface devices
- `SERVER` for **vimacc®** streaming servers
- `WORKSTATION` for **vimacc®** Workstation instances

- etc.

The `function` parameter in this command defines the device type from which the status information is requested.
If multiple device types should be subscribed, the respective `function` values have to be separated from each other by a comma (','). Each call of `subscribedevicestatus` overrides previous calls of this command.

Entry `function=none` terminates the transmission of status information.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

After successful execution of this command the status information of all respective device types are sent to the MSS with status message `resp=devicestatus` once. Further status messages `resp=devicestatus` are only sent to the MMS when the status of a subscribed **vimacc®** device has changed.

Note:
Depending on the number of configured **vimacc®** devices there may be a large number of status changes within the **vimacc®** system. Transferring all status information to the controlling instances could result in a very high network load.
To ensure that such a load is not generated accidentally, entry `function=all` has been omitted and all device types to be subscribed to have to be explicitly specified.

Note:
This command must be used very carefully. **vimacc®** is a dynamic system where each device immediately publishes its status changes. In case of systems with a high number of devices (e.g. several hundred video cameras), a thoughtless subscribing to status messages may lead to a very high number of messages from the control interface. In this case, the number of subscribed messages should be restricted by selecting the right `function` parameter.

Example:

- MMS→vimacc:
  `cmd=subscribedevicestatus;function=PTZ,FIX;userdata=789`

  vimacc→MMS:
  `resp=subscribedevicestatus;function=PTZ,FIX;userdata=789;answer=ok`

  vimacc→MMS (spontaneous status change):
  `resp=devicestatus;userdata=789;deviceid=1001;property=streaming;content=video\=ok`

3.7.3.19.1 Status Information from vimacc Devices

Information about the status of **vimacc®** devices are usually divided into several sections that are returned with the `property=<property>` parameter in each case. The returned `content=<status text>` parameter then contains the actual status information in text form.

Status information from **vimacc®** devices are amongst others presented by using the `control`, `streaming`, `recording`, `function` and `availability` sections, whereby not all of them have to be used for each device type.
In case of a **vimacc®** Workstation instance (`function=WORKSTATION`), e.g. entry `recording` always contains the content `state=off` as recording connections from the workstation instance to the recording servers are never established.
Possible Parameters are:

- `control`:
  Displays the status of device control.
- `streaming`:
  Displays the status of streaming connections.
- `recording`:
  Displays the status of recording connections.
- `function`:
  Displays the functions supported by the device.
- `availability`:
  Displays whether a device is available. Availability in this context defines whether the device was found in the system database, initialized by the respective **vimacc®** software component and registered to the **vimacc®** system. Whether the device can be accessed correctly can be determined from additional status information like `control`, `streaming` or `recording` (see above).

3.7.3.19.2 Status Information Reported by Cameras

In the following, an example of the status information referring to a camera is given:

- `control`:
  If the connection to the controlling instance could be established, the associated status message for a certain camera reads as follows:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =control;content=state\=ok
  ```
-
- `streaming`:
  If the streaming connection could be established, the associated status message for a certain camera reads as follows:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =streaming;content=video\=ok
  ```

  If the streaming connection was terminated due to a network error, the associated status message reads as follows:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =streaming;content=video\=disconnected
  ```

  Then, **vimacc®** periodically tries to re-establish the connection to the camera. Thus, the following messages may be shown in alternation.
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230_prealarm
  ;property=streaming;content=video\=no rtp

  resp=devicestatus;userdata=5678;deviceid=cam0230_prealarm
  ;property=streaming;content=video\=pending
  ```

  As soon as the connection could be re-established, the following message appears:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =streaming;content=video\=ok
  ```

- `recording`:
  The associated status message for a certain camera is:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =recording;content=REC1_PREALARM@localhost\=start prealar
  m 0 3600
  ```
- `function`:
  The associated status message for a certain camera is:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =function;content=PTZ,EMA
  ```
- `availability`:
  The associated status message for an available camera is:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =availability;content=ok
  ```
  For a camera that is no longer available the message is:
  ```
  resp=devicestatus;userdata=5678;deviceid=cam0230;property
  =availability;content=not ok
  ```

3.7.3.20 Requesting Event Information from vimacc Devices

Command:
`cmd=subscribeevents;function=<text>;userdata=<text>`

Response:
`resp=subscribeevents;function=<text>;userdata=<text>;answer=ok|failed`

Event message:
`resp=event;deviceid=<deviceid>;property=<property>;content=<event text>`

This command can be used to subscribe to alarms resp. events of **vimacc®** devices. As soon as an event of a subscribed **vimacc®** device is detected, it is routed as message `resp=event` to the MMS.

The `function` parameter in this command defines the device type from which the event information is requested (→ see command `subscribedevicestatus`).
If multiple device types should be subscribed, the respective `function` values have to be separated from each other by a comma (',').
Each call of `subscribedevicestatus` overrides previous calls of this command.

Entry `function=all` can be used to subscribe to the event messages of all devices within the **vimacc** system.

Entry `function=none` terminates the transmission of event information.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  `cmd=subscribedevicestatus;function=PTZ,FIX;userdata=7`

  vimacc→MMS:
  `cmd=subscribeevents;function=PTZ,FIX;userdata=7;answer=ok`

  vimacc→MMS (spontaneous event):
  `resp=event;userdata=7;deviceid=4001;property=deviceup;content=true`

3.7.3.20.1 Event Information Reported by vimacc Devices

Event information from **vimacc®** devices are not standardised as they vary depending on the device class.

As with the status information, the different parts are returned with the `property=<property>` parameter.
The returned `content=<event text>` parameter then contains the actual event information in text form.

3.7.3.20.2 Event Information Reported by Workstation Instances

In the following, some possible events of **vimacc®** Workstation instances are listed.

`deviceup`:
The workstation instance is started. The associated message is:

```
resp=event;userdata=5678;deviceid=401;property=deviceup;content=true
```

`playback`:
The workstation instance uses this event to report the playback status of one or more streams. As a result, the user can determine the current playback position, whether a playback is started or stopped and the time range covered by the connected playback streams. As playback streams are always synchronised this information is valid for all connected playback streams.
The associated message is:

```
resp=event;userdata=5678;deviceid=401;property=playback;content=streaming 100 3605364223097 3605356915780 3605412483511
```

`dialogs/VD<dialognummer>`
The workstation instance uses this event to report the stream connected to the respective video dialog, its status and whether it is a live or playback stream.
The associated message is:

```
resp=event;userdata=5678;deviceid=401;property=dialogs/VD1;content=live streaming cam0231_prealarm
```

`alarmhandling`:
Displays information about the handling of alarms at a workstation instance. Parameter `status` in `<event text>` specifies the alarm handling type.

If e.g. an alarm is triggered at a **vimacc®** Workstation by a user or by connecting an alarm scenario (→command `showscenario`), the respective message is a follows:

```
resp=event;userdata=5678;deviceid=401;property=alarmhandling;c
ontent=status\\\=created\\\;alarmid\\\=APLaptop-Heinrich 2013-
07-09 10:59:07 Uhr\\\;timestamputc\\\=2013-07-09T08:59:07.665
```

If an alarm is accepted by a user at a **vimacc®** Workstation, the following message appears:

```
resp=event;userdata=5678;deviceid=401;property=alarmhandling;c
ontent=status\\\=accepted\\\;alarmid\\\=1234\\\;timestamputc\\
\=2013-07-09T09:48:58.866
```

In case of an alarm terminated at a **vimacc** Workstation, the following message appears:

```
resp=event;userdata=5678;deviceid=401;property=alarmhandling;c
ontent=status\\\=finished\\\;alarmid\\\=APLaptop-Heinrich
2013-07-09 10:59:07 Uhr\\\;timestamputc\\\=2013-07-
09T09:36:54.406
```

3.7.3.21 Requesting Status Information from Playback Streams

Command:

```
cmd=subscribeplaybackstatus;mediatype=<video|audio|all>;userda
ta=<text>
```

Response:

```
resp=subscribeplaybackstatus;mediatype=<video|audio|all>;userd
ata=<text>;answer=ok|failed
```

Status message:

```
resp=playbackstatus;playbackid=<playbackid>;property=<property
>;content=<status text>
```

This command can be used to subscribe to changes to the status information of **vimacc®** playback streams.

The `mediatype` parameter in this command defines the media type from which the status information should be requested. For the **vimacc®** system the media types `video` and `audio` are available.

If multiple media types should be subscribed, the respective values have to be separated from each other by a comma (',').
Each call of `subscribeplaybackstatus` overrides previous calls of this command.

Entry `mediatype=none` terminates the transmission of status information.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

After successful execution of this command the status information of all respective media types are sent to the MSS with status message `resp=playbackstatus` once.

Further `resp=playbackstatus` status messages are only sent to the MMS when the status of a subscribed **vimacc®** playback stream has changed.

Note:
When switching to redundancy mode, e.g. the entry for the associated playback sessions changes and this change is indicated by the `playbacksessions` status property and content `changed`. In this case, the list of the respective playback sessions has to be retrieved again by using the `getplaybacksessionsforplaybackid` (see chapter 3.7.3.13) command.

Example:

- MMS→vimacc:
  `cmd=subscribeplaybackstatus;mediatype=video;userdata=789`

  vimacc→MMS:
  `resp=subscribeplaybackstatus;mediatype=video;userdata=789;answer=ok`

  vimacc→MMS (spontaneous status change):
  `resp=playbackstatus;userdata=789;playbackid=tankstelle;property=streaming;content=video\=ok`

3.7.3.21.1 Status Information from Playback Streams

Information about the status of **vimacc®** playback streams are usually divided into several sections that are returned with the `property=<property>` parameter in each case.
The returned `content=<status text>` parameter then contains the actual status information in text form.

Status information from **vimacc®** devices are amongst others represented using the `control`, `streaming`, `recording`, `function` and `availability` sections, whereby not all sections have to be used for each device type.
Possible Parameters are:

- `control`:
  Displays the status of the device control.
- `streaming`:
  Displays the status of streaming connections.
- `recording`:
  Displays the status of recording connections.
- `function`:
  Displays the functions supported by the device.

- `availability`:
  Displays whether a playback stream is available.

- `playbacksessions`:
  If this parameter contains the status text `changed`, the list of the playback sessions of a stream was modified by the recording servers.

3.7.3.22 Requesting Status Information from the vimacc System

Command:
`cmd=subscribesystemstatus;userdata=<text>;activate=<0|1>`

Response:
`resp=subscribesystemstatus;userdata=<text>;activate=<0|1>;answer=ok|failed`

Status message:
`resp=systemstatus;userdata=<text>;property=<property>;content=<status text>`

This command can be used to subscribe to changes to the status information of the **vimacc** system.
Examples for system information:

- Information about **vimacc**® computers that cannot be reached
- Information about **vimacc**® services that cannot be reached
- Information about cameras not working
- Information about incorrect or insufficient licences
- etc.

Parameter `activate` is used to subscribe to/unsubscribe from status information.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

After successful execution of this command the status information of the **vimacc**®system are sent to the MMS with the status message `resp=systemstatus`. Further `resp=systemstatus` status messages are only sent to the MMS when the system status has changed.

Example:

- MMS→vimacc:
  `cmd=subscribesystemstatus;userdata=789;activate=1`

  vimacc→MMS:
  `resp=subscribesystemstatus;userdata=789;activate=1;answer=ok`

vimacc→MMS (spontaneous event):
`resp=systemstatus;userdata=789;property=devices;content=state\=error\;source\=cam2001,cam2002"`

3.7.3.22.1 Reported Status Information of the vimacc System

Information about the status of the **vimacc®** system are also divided into several parts that are returned with the `property=<property>` parameter in each case.
The returned `content=<status text>` parameter then contains the actual status information in text form.

The following information is returned:

- `availability:`
  Displays whether the overall status of the **vimacc®** system could be determined. If the relevant information is not available, the following message is sent:
  `resp=systemstatus;userdata=1234;property=availability;content=not ok`

  As soon as the relevant information is available, the following message is sent:

  `resp=systemstatus;userdata=1234;property=availability;content=ok`

- `system`:
  Displays the overall status of the **vimacc** system. If there is no error, the respective message reads as follows:
  `resp=systemstatus;userdata=1234;property=system;content=state\=ok`

  If an error was detected, the affected areas are listed in the `source` parameter. The list of devices that reported an error is then also transferred via a status message with the respective `property` parameter.

  The following example shows a message that is displayed when errors are detected in the `services` and `devices` areas:

  `resp=systemstatus;userdata=1234;property=system;content=state\=error\;source\=services,devices`

- `devices`:
  Displays the overall status of **vimacc®** devices like cameras or HID controller (e.g. for joysticks). Workstation and Display instances are not included here, instead the data is transmitted in separated messages.
  Faulty devices are output in a comma separated list within the `source` parameter.

  The respective message is as follows (example):
  `resp=systemstatus;userdata=1234;property=devices;content=state\=error\;source\=4000,cam0231b_archiv,cam0231b_preal`

```
arm,cam0232_archiv,cam0232_prealarm
```

- `displays`:
  Displays the overall status of the configured **vimacc®** Display instances. Faulty Display instances are output in a comma separated list within the `source` parameter.

  The respective message is as follows (example):
  ```
  resp=systemstatus;userdata=1234;property=displays;content
  =state\=ok
  ```

- `workstations`:
  Displays the overall status of the configured **vimacc®** Workstation instances. Faulty Workstation instances are output in a comma separated list within the `source` parameter.

  The respective message is as follows (example):
  ```
  resp=systemstatus;userdata=1234;property=workstations;con
  tent=state\=ok
  ```

  If none of the configured **vimacc®** Workstation instances has been started or if the last configured **vimacc®** Workstation instance is terminated, the following error message is generated:

  ```
  resp=systemstatus;userdata=1234;property=workstations;con
  tent=state\=error\;errcause\=no workstation available\;so
  urce\=401
  ```

- `hosts`:
  Displays the overall status of the configured **vimacc®** controller instances. Faulty controller instances are output in a comma separated list within the `source` parameter.

  The respective message is as follows (example):
  ```
  resp=systemstatus;userdata=s;property=hosts;content=state
  \=ok
  ```

- `services`:
  Displays the overall status of all configured **vimacc®** services. **vimacc®** services are the provided functions of certain **vimacc®** modules like *AccVimaccEventManager* or *AccVimaccControlInterface*. Faulty services are output in a comma separated list within the `source` parameter.

  The respective message is as follows (example):
  ```
  resp=systemstatus;userdata=s;property=services;content=st
  ate\=ok
  ```

- `licence`:
  Displays whether the licence check detected any errors with respect to the **vimacc®** module or the input/output channels used.
  The respective message is as follows (example):
  `resp=systemstatus;userdata=s;property=licence;content=state\=ok`

3.7.3.23 Requesting Status Information about the vimacc Configuration Servers

Command:
`cmd=subscribeconfigserverstatus;userdata=<text>;activate=<0|1>`

Response:
`resp=subscribeconfigserverstatus;userdata=<text>;activate=<0|1>;answer=ok|failed`

Status message:
`resp=configserverstatus;property=<property>;content=<status text>`

vimacc® can be operated as distributed system where the central components can be operated on different computers to ensure a higher reliability. In such a system, the central database service for the entire configuration of a **vimacc** system, its control and all of its system states (the so called **vimacc** Configuration server) is operated on two different computers where one of the computers assumes the leading role and the other (the redundant partner) is operated in hot standby mode.

Using the `subscribeconfigserverstatus` command, status messages about the active configuration server can be subscribed.

Parameter `activate` is used to subscribe to/unsubscribe from status information.

After successful execution of this command the status information of the active configuration server is sent to the MMS with status message `resp=configserverstatus` once.
In this case, further `resp=devicestatus` status messages are only sent to the MMS when the respective values are changed.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.
Example:
MMS→vimacc:
`cmd=subscribeconfigserverstatus;userdata=786;activate=1`

vimacc→MMS:
`resp=subscribeconfigserverstatus;userdata=786;activate=1;answe`

```
r=ok
```

vimacc→MMS (spontaneous event):

```
resp=configserverstatus;userdata=786;property=hostname;content
=Server#1
```

3.7.3.23.1 Reported Status Information about the vimacc Configuration Servers

Information about the status of the **vimacc®** configuration servers are usually divided into several sections that are returned with the `property=<property>` parameter in each case.
The returned `content=<status text>` parameter then contains the actual status information in text form.

The following information is returned:

- `hostname`:
  Returns the name of the active configuration server.
  The associated message is:
  ```
  resp=configserverstatus;userdata=1234;property=hostname;c
  ontent=laptop-heinrich
  ```

- `redundancyState`:
  Returns the current redundancy status.

  The associated message is:
  ```
  resp=configserverstatus;userdata=1234;property=redundancy
  State;content=RC0
  ```

  `RC0` means that both configuration server are active and that the **vimacc®** configuration database `RC0` is loaded. The current configuration master is returned with the `hostname` parameter (see above).

  `RC1` means that a configuration server is down and that the **vimacc®** configuration database `RC1` is loaded. The current configuration master is returned with the `hostname` parameter (see above).

  `RC2` means that a configuration server is down and that the **vimacc®** configuration database `RC2` is loaded. The current configuration master is returned with the `hostname` parameter (see above).

3.7.4 VIMACC_CONTROL_DEVICES_ALARMS_ SCENARIOS

3.7.4.1 General

This protocol adds commands for connecting scenarios, transferring alarms to the **vimacc®** system and accepting/completing generated alarms to the VIMACC_CONTROL_BASIC protocol.

3.7.4.2 Connecting a Scenario

Command:
`cmd=showscenario;scenario=<name>;dest=<DisplayID>;createAlarm=<1|0>;contextid=<text>;userdata=<text>`

Response:
`resp=showscenario;scenario=<name>;dest=<DisplayID>;createAlarm=<1|0>;contextid=<text>;userdata=<text>;answer=ok|failed`

This command can be used to connect a scenario to a **vimacc®** Workstation or **vimacc®** Display instance (*DisplayID*).

Scenarios are referenced by their name. They have to be configured in the **vimacc®** system first. If the scenario is not known at the time of the connection, the response to the command contains the text `answer=failed,unknown scenario.`

Parameter `createAlarm=<1|0>` defines whether an alarm should also be triggered by the respective **vimacc®** Workstation instance when the scenario is connected. Triggering an alarm in connection with a scenario has the effect that the alarm recording for the video cameras involved is activated (if a recording connection is configured for the respective cameras). Usually, activating an alarm recording of a camera initiates the backup of a possibly configured pre-alarm recording and the increase of the refresh rate of the recording connection.
From this time on, the triggered alarm is identified by the value passed with the `contextid` parameter. Here, the caller has to be ensure a unique `contextid`.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

To terminate the connection established in context with the scenario the `cmd=clear` command (see above) can be used.

Note:
An alarm triggered due to the connection of a scenario and the respective alarm recording can be terminated by acknowledging the alarm at one of the **vimacc®** Workstation instances or by using the respective command on the control interface (→ command `finishAlarm`).

Example:

- MMS→vimacc:
  `cmd=showscenario;scenario=Scenario#1;contextid=1234;dest=AP_1`

  vimacc→MMS: (known scenario)
  `resp=showscenario;scenario=Scenario#1;contextid=1234;dest=AP_1;answer=ok`

  vimacc→MMS: (unknown scenario)
  `resp=showscenario;scenario=Scenario#1;contextid=1234;dest=AP_1;answer=unknown scenario`

3.7.4.3 Reporting an Alarm Event to the vimacc System

vimacc® can react to spontaneous events and execute actions automatically that were assigned to the event during configuration, if required.
Examples for a spontaneous event are a motion detected in a camera image, a closed or opened contact at an I/O module or an alarm transferred from the MMS.

A **vimacc® Workstation** can also display events respectively alarms signalized in the **vimacc®** system in an alarm queue to provide the user logged in with an overview of the signalized events and to enable processing in a certain order.

Command `createalarmforalarmqueue` can be used to create alarms within the **vimacc®** system that should be displayed in the alarm queue and "processed" by a user.
There are also commands that can be used to terminate alarms or to accept and confirm a certain alarm from the alarm queue at a **vimacc®** Workstation (see chapter 3.7.4.4 and 3.7.4.5).

Command:
`cmd=createalarmforalarmqueue;contextid=<text>;timetolive=<wert>;scenario=<name>;alarmtype=<text>;alarmprio=<wert>;destinationids=<Komma-separierte ID Liste>;userdata=<text>`

Response:
`resp=createalarmforalarmqueue;contextid=<text>;scenario=<name;timetolive=<wert>;alarmtype=<text>;alarmprio=<wert>;destinationids=<ids>;userdata=<text>;answer=ok|failed`

This command can be used to create an alarm within the **vimacc®** system that is automatically added to the system's alarm queue.

Parameter `contextid` is then used to identify and reference the generated alarm. Here, the caller has to ensure a unique `contextid`.
If an alarm with this `contextid` has already been activated within the **vimacc®** system, response `answer=failed,duplicate contextid` is returned.

Later, the `contextid` passed here is required to accept or terminate an alarm (→ see commands `acceptalarm` and `finishalarm`).

An alarm generated this way is automatically added to the so called alarm queue and it is expected that the alarm is accepted and confirmed by a user (or MMS).

Parameter `timetolive` defines the alarm lifecycle in seconds. Value `0` means `indefinite` so that the alarm persists until it is accepted and confirmed, i.e. it is not terminated automatically by the **vimacc®** system.
A value of `x mit x>0` means that the alarm should automatically be terminated by the **vimacc®** system after x seconds.
Note: If a value greater than 0 is passed for the `timetolive` parameter, the alarm is automatically terminated by the **vimacc®** system after the specified period of time regardless whether it has been accepted and confirmed by a user or not.

Parameter `scenario` specifies the scenario to be connected when the alarm is accepted at a **vimacc®** workstation. The alarm can be accepted by the user itself or programmatically by the MMS (→ see command `acceptalarm`).
Scenarios are referenced by their name. They have to be configured in the **vimacc®** system first.

Parameter `alarmtype` is optional and can be used to define a type description for the alarm triggered. This type can be used for a better differentiation between the different alarms.

Parameter `alarmprio` is optional and can be used to determine an alarm priority. Within the **vimacc®** system, the highest alarm priority is defined by the `0` value. If this parameter is not specified, an alarm with a priority of `1` is generated.

Parameter `destinationids` is optional and enables the user to specify the **vimacc®** workstations at which the generated alarm should be displayed in the alarm queue. Here, a comma separated list of **vimacc®** IDs from the respective **vimacc®** workstation instances is expected. If the parameter is not specified, the alarm is triggered at all **vimacc®** workstations.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  `cmd=createalarmforalarmqueue;contextid=1234;timetolive=0;`

```
scenario=scenario\ #1
```

vimacc→MMS: (contextid has not yet been assigned)

```
resp=createalarmforalarmqueue;contextid=1234;timetolive=0
;scenario=scenario\ #1;answer=ok
```

vimacc→MMS: (contextid has already been assigned)

```
resp=createalarmforalarmqueue;contextid=1234;timetolive=0
;scenario=scenario\ #1;answer=failed,duplicate contextid
```

3.7.4.4 Accepting an Alarm for a Workstation Instance

Command:

`cmd=acceptalarm;contextid=<text>;dest=<DisplayID>;userdata=<text>`

Response:

`resp=acceptalarm;contextid=<text>;dest=<DisplayID>;userdata=<text>;answer=ok|failed`

This command can be used to assign a queueing alarm within the **vimacc®** system directly to a **vimacc** Workstation instance for processing, i.e. the controlling instance accepts the respective alarm on behalf of the user.

When an alarm is accepted it is removed from the alarm queue of the system and a scenario which might have been passed before (→ see command `createalarm`) is connected to the respective **vimacc®** Workstation instance.

Parameter `contextid` is used to identify the alarm. If there is no alarm with this `contextid` in the **vimacc®** system, the command cannot be executed and response `answer=failed,unknown contextid` is returned.

Parameter `dest` defines the **vimacc®** Workstation instance that should accept the alarm. A scenario that has already been transferred (→ see command `createalarm`) is connected to this very **vimacc®** Workstation instance. If the referenced workstation is not available, the command cannot be executed and response `answer=failed,device not available` is returned.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received. In addition, this value is logged in the **vimacc®** Event database and can be used as search criterion in the course of a post-processing step.

Example:

- MMS→vimacc:
  `cmd=acceptalarm;contextid=1234;dest=AP_1;userdata=data`

vimacc→MMS: (contextid known)
```
resp=acceptalarm;contextid=1234;dest=AP_1;userdata=data;answer=ok
```

vimacc→MMS: (contextid not known)
```
resp=acceptalarm;contextid=1234;dest=AP_1;userdata=data;answer=failed,unknown contextid
```
vimacc→MMS: (Workstation instance not available)
```
resp=acceptalarm;contextid=1234;dest=AP_1;userdata=data;answer=failed,device not available
```

3.7.4.5 Terminating an Alarm

Command:
```
cmd=finishalarm;contextid=<text>;userdata=<text>;tags=<text>
```

Response:
```
resp=finishalarm;contextid=<text>;userdata=<text>;tags=<text>;answer=ok|failed
```

This command can be used to terminate an alarm within the **vimacc®** system.

If an alarm is terminated, associated actions configured are executed automatically and the alarm is completed. Thus, no further actions specific for this alarm can be performed.

If the alarm is still contained in the alarm queue of the system, it is automatically removed.

Parameter `contextid` is used to identify the alarm. If there is no alarm with this `contextid` in the **vimacc®** system, the command cannot be executed and response `answer=failed,unknown contextid` is returned.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received. In addition, this value is logged in the **vimacc®** Event database and can be used as search criterion in the course of a post-processing.

Parameter `tags` is not evaluated. However, it is logged in the **vimacc®** Event database and can be used as search criterion in the course of a post-processing step.

Example:

- MMS→vimacc:
  `cmd=finishalarm;contextid=1234;userdata=data;tags=EMA`

  vimacc→MMS: (contextid known)
  `resp=finishalarm;contextid=1234;userdata=data;tags=EMA;answer=ok`

  vimacc→MMS: (contextid not known)
  `resp=finishalarm;contextid=1234;userdata=data;tags=EMA;answer=unknown contextid`

3.7.4.6 Triggering the Alarm Status of a vimacc Device

Command:

```
cmd=triggerdevicealarm;source=<sourceID>;alarmid=<text>;alarmtime=<utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz,
Default: current daytime>;userdata=<text>;contextid=<text>
```

Response:

```
resp=triggerdevicealarm;source=<sourceID>;alarmid=<text>;alarmtime=<utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz,
Default: current daytime>;userdata=<text>;contextid=<text>;
answer=ok|failed
```

This command can be used to set a certain **vimacc®** device to the "Alarm" status.

The "Alarm" status may trigger different reactions depending on the device.
For example, in a **vimacc®** system, different recording connections can be configured for video cameras from which one connection is usually used for "normal" archiving and another for recording alarms. Then, an alarm recording can be configured in a way that – when an alarm is triggered – the system backs up a possibly configured pre-alarm recording and increases the refresh rate until the alarm is terminated.

Thus, this command can be used to activate an alarm for a specific **vimacc®** device.

In case of a **vimacc®** device, the "Alarm" status remains until the signalling is revoked e.g. by using the `cleardevicealarm` command (see chapter 3.7.4.7).

Parameter `source` defines the ID of the device for which the "Alarm" status should be triggered.

Parameter `alarmid` is used to identify the alarm within the devices.
For a device, several alarms can be triggered whereby each alarm must have a different `alarmid`.

Each alarm triggered has to be cleared by using the `cleardevicealarm` command and the respective `alarmid`. Only when all alarms have been cleared, the "Alarm" status is revoked by the device (see chapter 3.7.4.7).

Parameter `alarmtime` is used to set an alarm time. This parameter is optional. If a time stamp is passed, it is passed as UTC (Coordinated Universal Time) time stamp in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz` .
It no time stamp is passed, the current date and time are used as alarm time.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred as assignment characteristic when the command is logged in the vimacc documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Note:
vimacc® automatically activates an alarm for the devices when a command for triggering an alarm (see chapter 3.7.4.2 and 3.7.4.3) is passed.

Note:
Please note that this command can only trigger the alarm status of a single **vimacc®** device. The reaction to an "Alarm" status occurs inside the device as described above. This command cannot be used to trigger an alarm within the **vimacc®** system. For this purpose, the commands for generating an alarm have to be used (see chapter 3.7.4.2 and 3.7.4.3).

Example:
MMS→vimacc:
```
cmd=triggerdevicealarm;source=Camera_001;alarmid=testalarm;ala
rmtime=2014-03-20'T'18:20:37.125;userdata=test;contextid=1234
```

vimacc→MMS:
```
resp=triggerdevicealarm;source=Camera_001;alarmid=testalarm;al
armtime=2014-03-20'T'18:20:37.125;userdata=test;contextid=1234
;answer=ok
```

3.7.4.7 Clearing the Alarm Status of a vimacc Device

Command:
`cmd=cleardevicealarm;source=<sourceID>;alarmid=<text>;userdata=<text>;contextid=<text>`

Response:
`resp=cleardevicealarm;source=<sourceID>;alarmid=<text>;userdata=<text>;contextid=<text>;answer=ok|failed`

This command can be used to clear a previously triggered alarm for a specific **vimacc®** device.
However, the overall "Alarm" status is not revoked until all alarms triggered within the device are cleared.

Resetting the "Alarm" status may trigger different reactions depending on the device.

For the video cameras described in chapter 3.7.4.6, clearing the "Alarm" status has the effect that the alarm recoding is stopped.

Parameter `source` defines the ID of the device for which the "Alarm" status should be cleared.

Parameter `alarmid` is used to identify the alarm within the devices.

Parameter `alarmtime` is used to set an alarm time. This parameter is optional. If a time stamp is passed, it is passed as UTC (Coordinated Universal Time) time stamp in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.
It no time stamp is passed, the current date and time are used as alarm time.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Note:
The clearing of an alarm is automatically activated by **vimacc®** for the devices when a command for terminating an alarm is passed (see chapter 3.7.4.5).

Note:
Please note that this command clears the alarm status of a single **vimacc®** device only. This command cannot be used to terminate an alarm within the **vimacc®** system. For this purpose, the command for terminating an alarm has to be used (see chapter 3.7.4.5).

Example:

MMS→vimacc:
```
cmd=cleardevicealarm;source=Camera_001;alarmid=testalarm;userd
ata=test;contextid=1234
```

vimacc→MMS:
```
resp=cleardevicealarm;source=Camera_001;alarmid=testalarm;user
data=test;contextid=1234;answer=ok
```

3.7.4.8 Delete Protection for Time Ranges of a Playback Stream

Command:
```
cmd=addstreamprotection;playbackid=<playbackid;begintime=<utc
timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;endtime=<utc
timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;
userdata=<text>;contextid=<text>
```

Response:
```
resp=addstreamprotection;playbackid=<playbackid>;begintime=<ti
mestamp>;endtime=<timestamp>;userdata=<text>;userdata=<text>;c
ontextid=<text>;answer=ok|failed
```

This command can be used to protect a specified time range of a playback stream from being overwritten (delete protection). This can be useful when the respective stream is usually recorded in a ring and certain areas should not be deleted during the next automated archive adjustment.

If the recorded content is stored on several recording servers, the time ranges are protected on all of these servers.

Parameter `playbackid` defines the ID of the playback stream for which time ranges should be protected.

Parameters `begintime` and `endtime` define the time range to be protected. The time stamps passed are expected as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Note:
If further channels are assigned to a stream (see `associations` in chapter 3.7.3.12), the specified time ranges are not automatically protected for these channels. For this

purpose, the respective playback streams have to be explicitly protected by calling the `addstreamprotection` command again.

Example:
MMS→vimacc:
```
cmd=addstreamprotection;playbackid=cam0231_archiv;contextid=co
ntext#1;begintime=2014-03-31T18:00:37.890;endtime=2014-03-
31T18:10:37.890;userdata=test;contextid=<text>
```

vimacc→MMS:
```
resp=addstreamprotection;playbackid=cam0231_archiv;contextid=c
ontext#1;begintime=2014-03-31T18:00:37.890;endtime=2014-03-
31T18:10:37.890;userdata=test;answer=ok
```

3.7.4.9 Removing a Delete Protection for a Time Range of a Playback Stream

Command:
```
cmd=removestreamprotection;playbackid=<playbackid>;begintime=<
utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;
endtime=<utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;
userdata=<text>;contextid=<text>
```

Response:
```
resp=removestreamprotection;playbackid=<playbackid>;begintime=
<timestamp>;endtime=<timestamp>;userdata=<text>;userdata=<text
>;contextid=<text>;answer=ok|failed
```

This command can be used to remove the write protection from a time range of a video stream that was protected by using the `addstreamprotection` command (see chapter 3.7.4.8) so that it will be deleted during the next automated archive adjustment. If the recorded content is stored on several recording servers, the delete protection is removed for all servers.

Parameter `playbackid` defines the ID of the playback stream for which the protection should be removed from a specific time range.

The parameters `begintime` and `endtime` define the protected time range. The time stamps passed are expected as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
MMS→vimacc:

```
cmd=removestreamprotection;playbackid=cam0231_archiv;contextid
=context#1;begintime=2014-03-31T18:00:37.890;endtime=2014-03-
31T18:10:37.890;userdata=test
```

vimacc→MMS:

```
resp=removestreamprotection;playbackid=cam0231_archiv;contexti
d=context#1;begintime=2014-03-31T18:00:37.890;endtime=2014-03-
31T18:10:37.890;userdata=test;answer=ok
```

3.7.4.10 Querying the Protected Sections of a Playback Stream

Command:

```
cmd=getstreamprotectionlist;playbackid=<playbackid>;userdata=<
text>
```

Response:

```
resp=getstreamprotectionlist;playbackid=<playbackid>;userdata=
<text>;answer=ok|failed;parameterlist{\r\n
begintime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>;endtime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>\r\n
begintime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>;endtime=<utc timestamp iso 8601: yyyy-MM-
dd'T'hh:mm:ss.zzz>\r\n …
\r\n}
```

This command can be used to query the protected sections of a playback stream. Protected sections can be set by using the `addstreamprotection` command (see chapter 3.7.4.8).

If the recorded content is stored on multiple recording servers, the protected time ranges are determined on all servers and all ranges found are returned.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Parameter `playbackid` defines the ID of the playback stream whose limits should be determined.

The limits determined are returned as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

For each time range protected a line in the following format `begintime=<timestamp>\;endtime=<timestamp>\r\n` is returned in the response text.

Example:
MMS→vimacc:

```
cmd=getstreamprotectionlist;playbackid=cam0231_archiv;userdata
=test
```

vimacc→MMS:

```
resp=getstreamprotectionlist;playbackid=cam0231_archiv;userdat
a=test;answer=ok;parameterlist
{\r\nbegintime=2014-04-01T11:42:19.246\;endtime=2014-04-
01T11:49:38.727\r\n}
```

3.7.4.11 Deleting a Time Range from a Playback Stream

Command:

```
cmd=removetimespanfromstream;playbackid=<playbackid>;begintime
=<utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;
endtime=<utc timestamp iso 8601: yyyy-MM-dd'T'hh:mm:ss.zzz>;
userdata=<text>;contextid=<text>
```

Response:

```
resp=removetimespanfromstream;playbackid=<playbackid>;begintim
e=<timestamp>;endtime=<timestamp>;userdata=<text>;userdata=<te
xt>;contextid=<text>;answer=ok|failed
```

This command can be used to delete a specific time range from a recorded video stream.

If the recorded content is stored on several recording servers, the respective time range is deleted on each server.

Parameter `playbackid` specifies the ID of the playback stream from which the specified time range should be deleted.

Parameters `begintime` and `endtime` define the time range to be deleted. The time stamps passed are expected as UTC (Coordinated Universal Time) time stamps in the following format `yyyy-MM-dd'T'hh:mm:ss.zzz`.

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is transferred as assignment characteristic when the command is logged in the **vimacc** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
MMS→vimacc:

```
cmd=removetimespanfromstream;playbackid=cam0231_archiv;context
id=context#1;begintime=2014-03-31T18:00:37.890;endtime=2014-
03-31T18:10:37.890;userdata=test
```

vimacc→MMS:

```
resp=removetimespanfromstream;playbackid=cam0231_archiv;contex
tid=context#1;begintime=2014-03-31T18:00:37.890;endtime=2014-
03-31T18:10:37.890;userdata=test;answer=ok
```

3.7.4.12 Setting a Text Mark for a Live Stream

Command:
`cmd=setbookmarkforstream;deviceid=<deviceid>;text=<text>;userdata=<text>;contextid=<text>`

Response:
`resp=setbookmarkforstream;deviceid=<deviceid>;text=<text>;userdata=<text>;contextid=<text>;answer=ok|failed`

This command can be used to store a text mark (bookmark) for a certain stream so that a certain position can be easily found if a subsequent evaluation is required.
The text is passed to the **vimacc®** documentation layer together with the associated point in time.

Parameter `deviceid` specifies the ID of the stream to which the `text` passed should be assigned.

The entry passed in the parameter `text` should be included in inverted commas (`"<text>"`) so that it can also include spaces and special characters. (Semicolons, colons and inverted commas have to be escaped.)

Parameter `contextid` is used to identify and reference the command to previous commands and events. The parameter itself is not evaluated. Instead, it is passed as assignment characteristic when the command is logged in the **vimacc®** documentation layer.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

Example:
MMS→vimacc:
`cmd=setbookmarkforstream;deviceid=cam0231;contextid=context#1;text="Dies ist der Text eines Lesezeichens";userdata=test`

vimacc→MMS:
`resp=setbookmarkforstream;deviceid=cam0231;contextid=context#1;text="Dies ist der Text eines Lesezeichens";userdata=test;answer=ok`

3.7.5 VIMACC_CONTROL_ALL

3.7.5.1 General

This protocol adds commands for writing so called "data points" of the **vimacc** system to the VIMACC_CONTROL_DEVICES_ ALARMS_SCENARIOS protocol.

3.7.5.2 Writing a Data Point

Command:
`cmd=writedp;contextid=<text>;datapointname=<text>;datapointvalue=<text>`

Response:
`resp=writedp;contextid=<text>;datapointname=<text>;datapointvalue=<text>;answer=ok|failed`

This command can be used to write any data point of the **vimacc** system.

Parameter `datapointname` defines the data point to be written in the **vimacc** Config, starting from the root node. Thus, the complete path of the data point has to be specified.

Parameter `datapointvalue` defines the content to be written to the data point passed.

Parameter `contextid` is not evaluated but used for logging and returned within the responses to ensure a better differentiation of the messages received.

Note:
Here, key-value pairs have to be passed as content for parameter `datapointvalue` as well, e.g. command `cmd=show;source=cam0231;videodlg=1.`
For a correct protocol evaluation the text to be passed of parameter `datapointvalue` has to be escaped three times with special character '\' '.
This is because otherwise no special characters like ('), (\), (\r), (\n) and (\t) could be passed and key-value pairs contained could not be correctly determined.
As the key-value pairs of this parameter are setup the same way as the entire command, i.e. by using a (=), and the single key-value pairs are separated by a semicolon (;), this structure could not be determined from the command without the escape character.

Thus, the text to be passed has to be "escaped" in this parameter as follows:

1. As the commands are identified by the key-value pair `cmd=<Kommando>` character '=' has to be escaped first.
2. Since the key-value pairs are separated by a ';', the ';' character has to be escaped in the resulting text.
3. Since the protocol parser "de-escapes" the received text once as well, all special characters have to be escaped in the resulting text again.

If e.g. command `cmd=show;source=cam0231;videodlg=1` should be passed with the `datapointvalue` parameter, the following text is passed:

1. Escape character '=':
   ➔ `cmd\=show;source\=cam0231;videodlg\=1`
2. Escape ';' character in the resulting text:
   ➔ `cmd\\=show\;source\\=cam0231\;videodlg\\=1`
3. Escape all special characters in the resulting text again:
   ➔ `cmd\\\\=show\\;source\\\\=cam0231\\;videodlg\\\\=1`

Example:

- MMS→vimacc:
  `cmd=writedp;contextid=1234;datapointname=ActiveDeviceList/4000/command/request;datapointvalue=cmd\\\\=show\\;source\\\\=10000\\;videodlg\\\\=VD1`

  vimacc→MMS:
  `resp=writedp;contextid=1234;datapointname=ActiveDeviceList/4000/command/request;datapointvalue=cmd\\\\=show\\;source\\\\=10000\\;videodlg\\\\=VD1;answer=ok`

3.7.5.3 Writing the Command Data Point of a vimacc Device

Command:
`cmd=writecommanddp;contextid=<text>;deviceid=<deviceid>;datapointvalue=<text>`

Response:
`resp=writecommanddp;contextid=<text>;deviceid=<deviceid>;datapointvalue=<text>;answer=ok|failed`

This command can be used to write the command data point (the `command/request` data point) of a **vimacc®** device.

Parameter `deviceid` defines the respective **vimacc®** device. It is checked whether the device has already been activated within the **vimacc®** system. If not, `answer=failed,device not available` is returned.

Parameter `datapointvalue` defines the content to be written to the data point passed. Here, the transferred text has to be escaped as well (→command `writedp`).

Parameter `contextid` is not evaluated but used for logging and returned within the responses to ensure a better differentiation of the messages received.

Example:

- MMS→vimacc:
  ```
  cmd=writecommanddp;contextid=1234;deviceid=4000;datapoint
  value=cmd\\\\=show\\;source\\\\=10000\\;videodlg\\\\=VD1
  ```

  vimacc→MMS:
  ```
  resp=writecommanddp;contextid=1234;deviceid=4000;datapoin
  tvalue=cmd\\\\=show\\;source\\\\=10000\\;videodlg\\\\=VD1
  ;answer=ok
  ```

3.7.5.4 Reading a Data Point

Command:

```
cmd=readdp;userdata=<text>;datapointname=<text>
```

Response:

```
resp=readdp;userdata=<text>;datapointname=<text>;answer=ok,<da
tapoint value>|failed
```

This command can be used to read any data point of the **vimacc®** system.

Parameter `datapointname` defines the data point to be read in the **vimacc®** Config file, starting from the root node. Thus, the complete path of the data point has to be specified.

Parameter `userdata` is not evaluated but returned within the responses to ensure a better differentiation of the messages received.

The value of the data point read is returned in the answer.

Note:
To be able to extract the returned content of the read data point from the response text, (special) characters like ("), (\), (\r), (\n), (\t), (=), (,) and (;) are preceded by the escape character (\).

Example:

- MMS→vimacc:
  ```
  cmd=readdp;userdata=1234;datapointname=ActiveDeviceList/401/command/response
  ```

  vimacc→MMS:
  ```
  resp=readdp;userdata=1234;datapointname=ActiveDeviceList/401/command/response;answer=ok,cmd\=acceptalarm\;contextid\=11\;error\=rejected
  ```

3.7.6 VIMACC_CONTROL_FALLBACK

3.7.6.1 General

This protocol provides only a limited set of commands. It is used to obtain some information about the status of a **vimacc®** system when it can no longer be operated correctly.

This might be the case when the required licence has not yet been activated or is expired. Demo setups can usually be used for up to 12 weeks. After that, the **vimacc®** components will be deactivated.
To ensure that a higher level management system is at least informed about the key error states, this protocol is activated by default.

3.7.6.2 Querying Available Commands

Command:
```
cmd=help;userdata=<text>
```

Response:
```
resp=help;userdata=<text>;answer=ok,parameterlist{\r\nbefehl#1
\r\n...befehl#n\r\n}
```

For more information, refer to chapter 3.7.3.2.

3.7.6.3 Monitoring the Control Connection

Command:
```
cmd=keepalive;userdata=<text>
```

Response:
```
resp=keepalive;userdata=<text>;answer=ok
```

For more information refer to chapter 3.7.3.3.

3.7.6.4 Requesting Status Information from the vimacc System

Command:
```
cmd=subscribesystemstatus;userdata=<text>;activate=<0|1>
```

Response:
```
resp=subscribesystemstatus;userdata=<text>;activate=<0|1>;answ
er=ok|failed
```

Status message:
```
resp=systemstatus;userdata=<text>;property=<property>;content=
<status text>
```

For more information refer to chapter 3.7.3.22.

4 vimacc Live

Access to live streams of the system can be enabled by implementing the **vimacc®** video widget or by using the **vimacc®** RTSP server.
See also *chapter 3.7.3.11 Querying Configured Cameras*

4.1 Video Widget

The video widget is a C++/QT basic implementation for displaying video streams of a **vimacc®** system. Video streams are retrieved directly via TCP/IP from the **vimacc®** interface process and displayed. The corresponding codecs are already contained. The implementing body has to register and pay for the respective patent portfolio usage rights at MPEGLA.

SIMATIC WinCC OA Video EWO is an implementation variant of the **vimacc®** video widget.
An autonomic video widget is not subjected to the **vimacc®** Rights Management when it comes to displaying streams as it is not connected to the central **vimacc®** configuration. Thus, no user rights can be assigened.
The retrieving body has to provide for an access protection by means of respective mechanisms.

4.2 RTSP Server

The **vimacc®** RTSP server is a basic setup component and part of all **vimacc®** editions. It has to be activated via the licence files and configured by the **vimacc®** AdministrationCenter, see **vimacc®**Administrator's Guide.

Access is performed by using the regular access notation via the following URL structure:
`rtsp://<Rechnername>:<Port>/<StreamID>/live`

`<Rechnername>`	Name or IP address of the server/PC
`<Port>`	Port number (default: 5544)
`<StreamID>`	**vimacc®**Stream-ID

See also *chapter 3.7.3.11 Querying Configured Cameras*

Example: rtsp://VideoServer1:5544/cam_axis2025_0001/**live**

The RTSP server can only be used to retrieve streams in H.264 or MPEG4 format. For accessing streams with other codecs the **vimacc®** HTTP server can be used. It provides all stream types in a uniform and transcoded MJPEG stream.

5 vimacc Playback

Access to live streams of the system can be enabled by implementing the **vimacc®** video widget or by using the **vimacc®** RTSP server.
See also *chapter 3.7.3.12 Querying Available Playback Streams*

5.1 Video Widget

The video widget is a C++/QT basic implementation for displaying video streams of a **vimacc®** system. Video streams are retrieved directly via TCP/IP from the respective **vimacc®** server process and displayed. The corresponding codecs are already contained. The implementing body has to register and pay for the respective patent portfolio usage rights at MPEGLA.

SIMATIC WinCC OA Video EWO is an implementation variant of the **vimacc®** video widget.
An autonomic video widget is not subjected to the **vimacc®** Rights Management when it comes to displaying streams as it is not connected to the central **vimacc®** configuration.
The retrieving body has to provide for an access protection by means of respective mechanisms.

5.2 RTSP Server

The **vimacc®** RTSP server is a basic setup component and part of all **vimacc®** editions. It has to be activated via the licence files and configured by the **vimacc®** AdministrationCenter, see **vimacc®**Administrator's Guide.

Access is performed by using the regular access notation via the following URL structure:
`rtsp://<Rechnername>:<Port>/<StreamID>/`**`playback`**

`<Rechnername>`	Name or IP address of the server/PC
`<Port>`	Port number (default: 5544)
`<StreamID>`	**vimacc®**Stream-ID

See also *chapter 3.7.3.12 Querying Available Playback Streams*

Example: rtsp://VideoServer1:5544/cam_axis2025_0001/**playback**

The RTSP server can only be used to retrieve streams in H.264 or MPEG4 format. For accessing streams with other codecs the **vimacc®** HTTP server can be used. It provides all stream types in a uniform and transcoded MJPEG stream.

6 Support / Hotline

More information: www.vimacc.de

Do you have any questions about **vimacc®**?

- Send an email to **support@accellence.de**
  or
- Call our hotline: **+49 (0)511 277 2490**

Our staff is happy to help you from 09:00 AM to 5:00 PM CET/CEST on business day.

7 Index