

GeViSoft SDK

Dokumentation | Documentation | Documentation | Documentación

GEUTEBRÜCK

knowledge base

Version 2012_1.7 | Date 19.07.2012

GeViSoft SDK

Introduction

The GeViSoft SDK allows integrating your custom solutions and products with Geutebrück's GeViSoft suite. It includes an application programming interface (API) with all necessary DLLs, headers, example projects, and documentation to help you getting started with your integration easily.

The SDK supports C++ and Delphi. Furthermore a .Net wrapper is included which allows you to use the SDK from C#. It provides various example projects and solutions in these languages.

GeViSoft

GeViSoft is Geutebrück's central management system for video control. Its main function is the switching of video signals between different cameras, monitors and DVRs by controlling a video matrix system. Alarm handling as well as the remote control of pan/tilt and dome cameras is a further functionality of GeViSoft.

GeViSoft can also be used to handle general purpose digital inputs and outputs and thus allows integrating custom sensor technology and actuating elements to the Geutebrück system.

Furthermore, different peripherals common to video control systems, like video motion analysis or operator consoles, can be managed.

GeViSoft Architecture

The architecture of GeViSoft follows the client-server paradigm. The server software (GeViServer) usual runs on a dedicated PC. This hardware platform is called GeViStation. The combined system of software and hardware is called GeViControl.

At least one IO client must handle connections to the peripherals. This client is communicating with the GeViSoft server and runs on the same machine. It is called GeViIO.

The GeViIO client provides the interfaces for the communication to the attached peripherals like a VX3 matrix or a PTZ. These peripherals can also be virtualized.

GeViServer and GeViIO can be configured from the GeViSet application. The configuration is described in detail in chapter [Configuration of GeViSoft](#).

The following figure shows a setup of GeViSoft with an attached VX3, digital IO and two PTZ devices.

GeViSoft Example Configuration

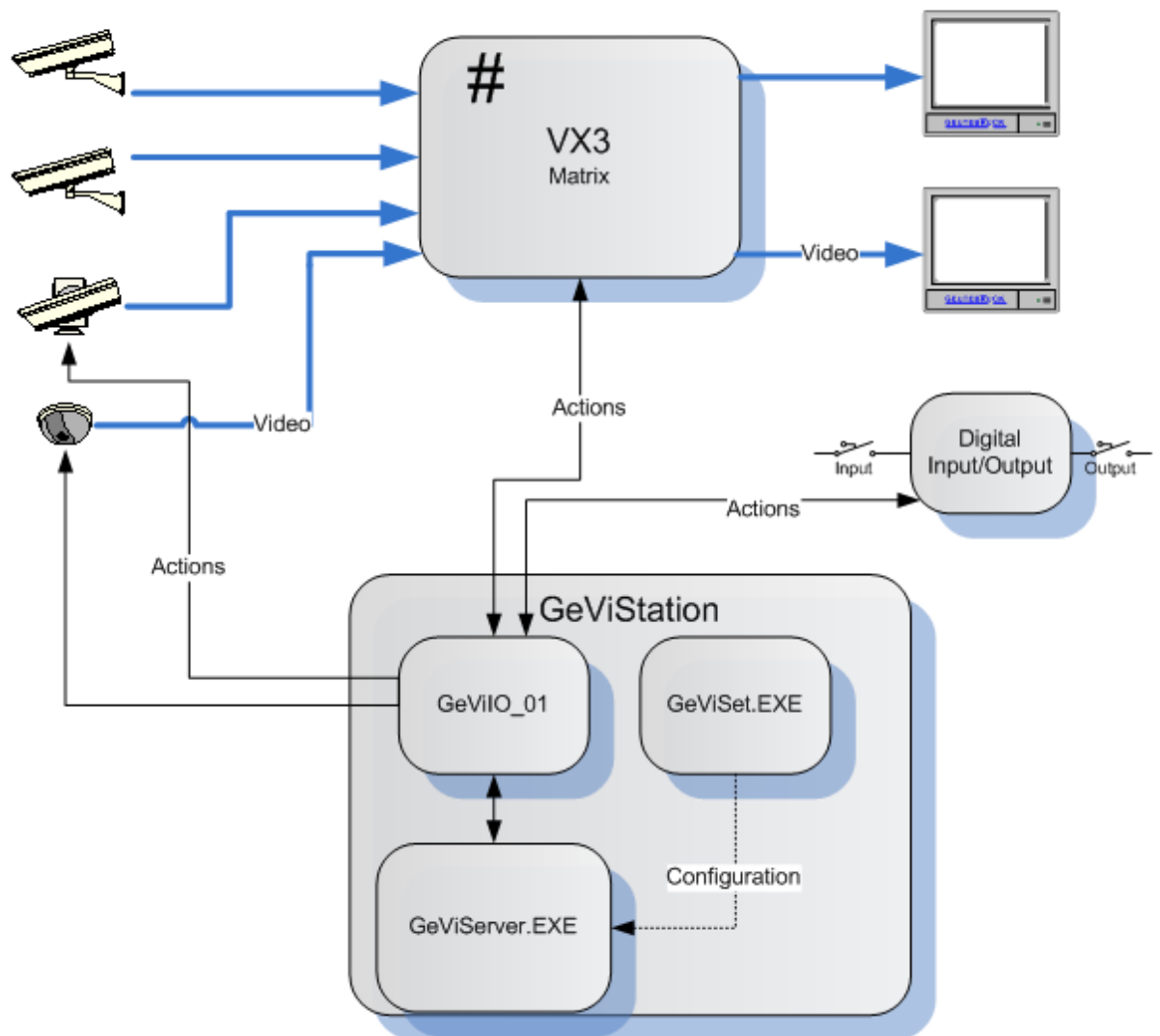


Figure 1 - GeViSoft Example Configuration

Historically, there has been a demand to control a large number of video feeds with a limited number of monitors in surveillance systems. This has lead to the invention of video matrixes like the VX3, which allowed different camera signals to be dynamically routed to the attached monitors. The routing could be user initiated or triggered by external events like alarms or digital inputs.

Besides the video routing it was necessary to allow the operator to remote control PTZ and dome cameras from a central console to react on alarms or other events.

A configuration like the one described above is reflected in the setup according to figure 1.

Nowadays analogue video cameras and monitors are getting replaced by IP cameras and PCs running software viewers like GSCView. GeViSoft allows the handling of these modern setups as well so that they can be integrated seamlessly into existing installations.

Figure 2 gives an example for a complex setup integrating analogue as well as digital components.

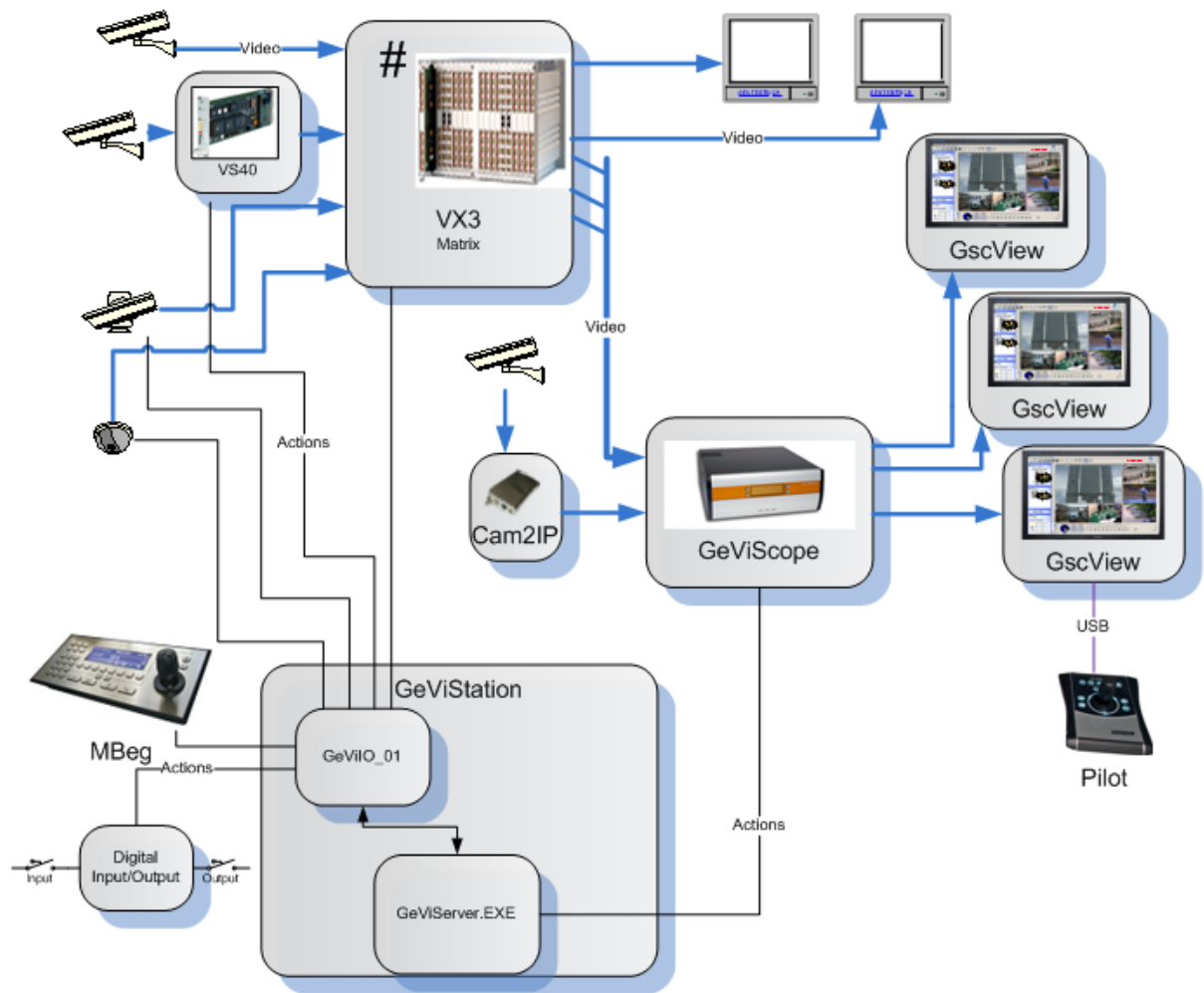


Figure 2 - Complex GeViSoft Setup

Additional to controlling the cross switching inside the matrix, GeViSoft can be used to communicate with GeViScopes. It is possible to configure GeViSoft in such a way that a GeViScope and the connected GscViews can be controlled just like an analogue video matrix, e.g. a VX3.

The next chapter gives an overview of the different components that add up to GeViSoft.

GeViServer

GeViServer is the backend server in a GeViSoft system. It also manages the internal database. GeViServer usually runs as a Windows service on production machines, but can also be started as a console application for testing purposes or debugging. If installed by the SDK setup, the GeViServer must be started from the console.

It is possible to run GeViServer in a cluster to increase reliability.

GeViAdmin

The application GeViAdmin is used to set up the GeViServer database. It can also be used to configure redundancy settings by clustering several GeViServers. Furthermore, GeViScope can be used for diagnostics and load analysis. GeViAdmin is part of the shipping product, but not of the SDK. The SDK installer automatically sets up a GeViSoft database during the installation process.

GeViIO

The GeViIO client is responsible for the communication with the external interfaces and peripherals. It runs on the same machine as the GeViServer. Other instances of GeViIO might run on separate machines.

GeViSet

GeViSet is the configuration tool for GeViServer. It can be used to configure GeViIO clients, users, events, alarms and all other functionalities of GeViServer, as well as connections to GeViScope servers. Some configuration steps and options inside GeViSet are shown in the following chapters.

GeViAPI Test Client

The GeViAPI Test Client allows testing and debugging GeViSoft applications. With this tool you can send and receive actions and alarms, query the database, and retrieve system information.

SDK Introduction

The GeViSoft SDK provides you with an open application programming interface to the GeViSoft suite and allows you to integrate your custom products with Geutebrück's.

The SDK includes the DLLs and corresponding header files required by your C++ or Delphi projects. Furthermore .Net wrapper DLLs are included which allow you to use the SDK from your C# application.

Several example applications help you getting started with the GeViSoft SDK development and may act as a foundation for your own solutions.

Files and Directory Structure

During installation, the environment variable %GEVISOFTSDKPATH% is set. It points to the root directory of the SDK installation. The variable's value is determined by the path chosen as the install directory during setup. Usually, this is "C:\GEVISOFT". All SDK directories are located inside this root directory.

This is a (partial) tree view of the standard installation:

C:\GEVISOFT

└── DATABASE

└── Documentation

└── Examples

└── DelphiXE

└── Delphi_ConsoleClient

└── Delphi_SimpleActionClient

└── Delphi_SimpleClient

└── Delphi_SimpleDatabaseClient

└── GeViScopeSDK

└── Include

└── GeViSoftSDK

└── Include

└── VS2008CPP

└── CPP_Console_Client

└── CPP_SimpleActionClient

└── CPP_SimpleClient

└── CPP_SimpleDatabaseClient

└── GeViScopeSDK

└── INCLUDE

└── LIB

└── GeViSoftSDK

└── Include

└── Lib

Please note that the directory C:/GEVISOFT/DATABASE will be created without regarding the chosen install path. This directory hosts the GeViSoft database GeViDB.mdb which is hidden by default.

Inside the %GEVISOFTSDKPATH% directory, a structure like this is created:

- A *Documentation* folder containing all GeViSoft related documentation and manuals.
- An *Examples* folder including subfolders that are named according to the corresponding IDE and programming language.
 - Inside each of these, there is a *GeViScopeSDK* and *GeViSoftSDK* folder with the respective *Include* and *Lib* folders for the programming language as well as the folders with the different examples.
 - The C++ headers are located inside the *Include* folder and the libraries inside the *Lib* folder.
 - For Delphi, the *.pas* and the *.inc* files can all be found inside the *Include* folder.

The %GEVISOFTSDKPATH% directory itself hosts all the executables, dynamic link libraries, and runtime files that are needed by GeViSoft. By default, all the example projects will output their generated binaries into this folder as well. This guarantees that all runtime dependencies are met and your compiled executables find the needed DLLs.

Additionally, the .Net wrapper assemblies Geutebrueck.GeViSoftSDKNetWrapper.dll and GscActionsNET.dll reside in this folder.

SDK Setup

Setup of Test/Build Environment

This chapter describes how to setup and configure the GeViSoft test environment.

NOTICE

Please note that you need administrative privileges on the development machine.

Installation of GeViSoft

The SDK is shipped as an executable installer. You just need to run it on your development machine in order to install the SDK.

NOTICE

It is highly recommended to install GeViSoft to the default path C:/Gevisoft.

WARNING

Please make sure that you do not install the SDK on a production GeViSoft machine as the setup will overwrite the installed GeViSoft files without notice.

Starting GeViServer

You can start GeViServer from the command prompt by issuing the command

```
%GEVISOFTSDKPATH%/geviserver.exe console
```

or by executing the `startserver.bat` script in your GeViSoft installation's root folder. The `console` argument forces the software to run as a console application and allows you to

easily monitor the server's output. On a production machine, GeViServer usually runs as a windows service.

NOTICE

Please note that without a license dongle, the server will terminate after two hours. You can directly restart it without any further restrictions.

Configuration of GeViSoft

In this chapter you will learn how to establish a connection to the GeViServer with the setup client GeViSet ([Setting up the server connection](#)).


After that there is a description for setting up a GeViIO client that provides a virtual video matrix and digital IO ([Configuration of the GeViIO Client](#)). You do not need to carry out the steps described in that paragraph. They are for reference only because this configuration is already done for you in the database that is delivered with the SDK.

Setting up the server connection

- 1 Start GeViServer by executing `startserver.bat` if not already done so
- 2 Start GeViSet.exe
- 3 Setup the server connection
 - a Open *File* -> *GeViSoft server connections*
 - b If a connection *localhost* exists, press *Connect* and move to step 4
 - c If no connection exists choose *Connections* -> *New Connection*
 - d Enter *localhost* as the name of the new connection and press the *Forward* button
 - e In the *Create New Server Connection* window set the computer name to *localhost*, the user name to *sysadmin*. Check *Save password* and set the password to *masterkey*. Select *Local connection* as connection type. Press the *Forward* button. Choose the *localhost* connection and press *Connect*

Configuration of the GeViIO Client (reference)

The GeViIO client's configuration is already done for you inside the database that is shipped with the SDK. The steps described here are only a reference for you if you need to adapt settings for your test environment.

1. In the *Clients* field push the *Add* button and add a new GeViIO client with the name *GeViIO_01*.
2. Select the new GeViIO client and press *Configure*.
3. Mark the client as *Active* and *Virtual*.
4. Add a new VX3 matrix by pressing *Add* in the *Interfaces* field and selecting the appropriate type (VX3/CX3). Name the interface *Virtual VX3*.
5. Select the newly created VX3 interface and press *Edit*.
6. Add 16 new video inputs to the VX3 interface by pressing the *Add* button in the *Video inputs* tab. In the *New video Input* window set *Count* to 16 and press ok. The new video input channels should show up in the *Video input* tab.
7. Add 4 new video outputs in the same manner as the inputs.
8. Add 8 new input contacts and 8 new output contacts in the same way you did for the video input.
9. Send your newly created setup to the server by choosing *File -> Setup to server* or by clicking .

Now your client window should look like this:

Client GeViIO

Client

☒ Active
Name: GeViIO_01

☒ Virtual
Description: GeViIO_01

Synchronization: ☒
Interval: 24 h

Interfaces:

Name	Port-Nr
<input checked="" type="checkbox"/> Virtual Vx3	COM1

Add ...
Edit ...
Remove

Global GeViIO resources

Video input list:

Global ID	Name
<input checked="" type="checkbox"/> 1	Video input1
<input checked="" type="checkbox"/> 2	Video input2
<input checked="" type="checkbox"/> 3	Video input3
<input checked="" type="checkbox"/> 4	Video input4
<input checked="" type="checkbox"/> 5	Video input5
<input checked="" type="checkbox"/> 6	Video input6
<input checked="" type="checkbox"/> 7	Video input7
<input checked="" type="checkbox"/> 8	Video input8
<input checked="" type="checkbox"/> 9	Video input9
<input checked="" type="checkbox"/> 10	Video input...
<input checked="" type="checkbox"/> 11	Video input...
<input checked="" type="checkbox"/> 12	Video input...
<input checked="" type="checkbox"/> 13	Video input...
<input checked="" type="checkbox"/> 14	Video input...
<input checked="" type="checkbox"/> 15	Video input...
<input checked="" type="checkbox"/> 16	Video input...

Video output list:

Global ID	Name
<input checked="" type="checkbox"/> 1	Video outp...
<input checked="" type="checkbox"/> 2	Video outp...
<input checked="" type="checkbox"/> 3	Video outp...
<input checked="" type="checkbox"/> 4	Video outp...

Input contact list:

Global ID	Name
<input checked="" type="checkbox"/> 1	Digital input1
<input checked="" type="checkbox"/> 2	Digital input2
<input checked="" type="checkbox"/> 3	Digital input3
<input checked="" type="checkbox"/> 4	Digital input4
<input checked="" type="checkbox"/> 5	Digital input5
<input checked="" type="checkbox"/> 6	Digital input6
<input checked="" type="checkbox"/> 7	Digital input7
<input checked="" type="checkbox"/> 8	Digital input8

Output contact list:

Global ID	Name
<input checked="" type="checkbox"/> 1	Digital outp...
<input checked="" type="checkbox"/> 2	Digital outp...
<input checked="" type="checkbox"/> 3	Digital outp...
<input checked="" type="checkbox"/> 4	Digital outp...
<input checked="" type="checkbox"/> 5	Digital outp...
<input checked="" type="checkbox"/> 6	Digital outp...
<input checked="" type="checkbox"/> 7	Digital outp...
<input checked="" type="checkbox"/> 8	Digital outp...

OK
Cancel

Connection to GeViScope (optional)

If you have a GeViScope server up and running, you can connect GeViSoft to it via a TCP/IP connection. If connected, actions can be exchanged between the two systems. As an

example this can be used to remote control GSCView.

Please note that you can install the GeViScope Server as a part of Geutebrück's GeViScope SDK if you have not done it yet. You can download this SDK on www.geutebrueck.com or request it from the SDK division.

Installing the GeViScope SDK is a prerequisite for the scenario and example in the chapter [Switching Video](#).

You can configure the connection to GeViScope inside GeViSet. Choose the menu Server -> GeViScope Connections and press Add in the pop-up menu. You can then configure the connection parameters inside the GeViScope connection window.

NOTICE

Please note that the Alias is used to address different GeViScope servers from inside the SDK with GSCActions. See [Action messages -> creating action messages > 4. Example of creating a GeViScope Action Message](#)

First Steps with GeViSoft

This chapter will lead you throughout your first steps with GeViSoft. You will learn how to connect to a GeViServer, send some basic actions, and customize message logging and display to your needs. If you are already familiar with GeViSoft, you can skip this chapter or skim through it.

GeViAPI Test Client

The easiest way to test your GeViSoft setup is by using the GeViAPI Test Client. You can start it from your %GEVISOFTSDKPATH% directory.

Please make sure that your GeViServer is already started. If not start it by executing the “startserver.bat” inside the GeViSoft root directory.

After startup connect to the GeViServer by adding your credentials and pressing the “Conn” button. If everything works out, the “Connected” indicator will be illuminated in green and several messages will pop up in the “Communication log”. At this point your communication is set up correctly.

If you have followed the configuration steps in chapter [Setting up GeViIO](#) you will already be able to use GeViSoft for switching your virtual video I/O.

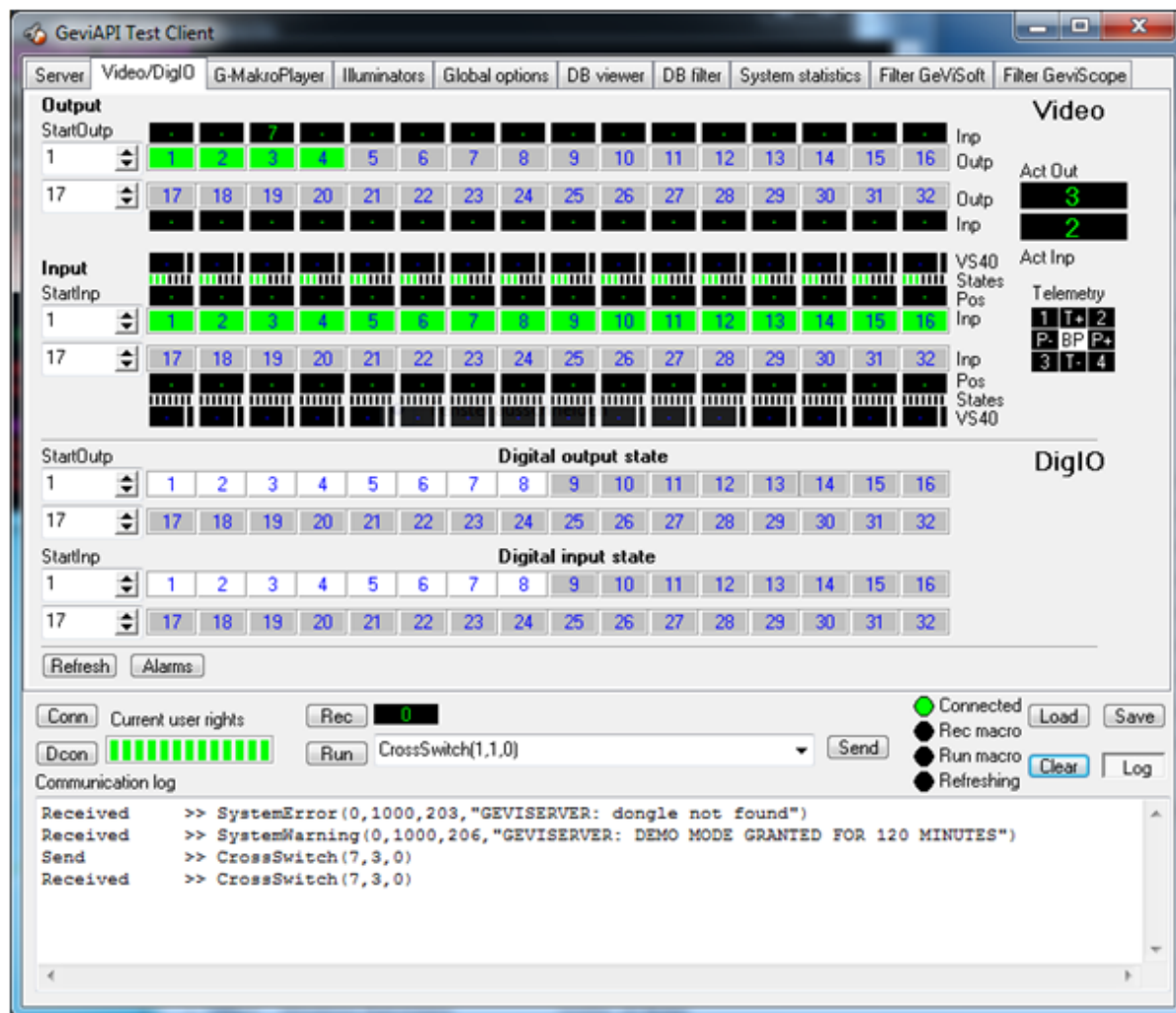
Cross Switching Video

Select the tab *Video/DigIO*. You can switch your video signal in the following way:

1. Select an active input and an active output. The signal will be switched between these two. You can see the active I/O on the windows right hand side beneath the text *Video*.
 - a) To select an active output, left-click on one of your configured video outputs in the upper window area. You should see *Act Out* changing with regard to your selection.

b) Now move the mouse over the desired input (e.g. 7) and right-click on the input. The number of your selected input should now appear in the black square above your selected output.

2. Clear a video output. Move the mouse over the output to clear and right-click on it. The number in the black square above the output should vanish.



NOTICE

When switching the output, a *CrossSwitch* action with the chosen channels is displayed in the *Communication Log* shown in the lower part of the GeViAPI Test Client’s window.

If a real VX3 would be connected to your GeViSoft and the inputs were connected to video signals, you would switch the real signal to the according output (normally a monitor). You will learn how to use these switch actions to remote control a GscView in the same way you would use an analogue matrix in the chapter [Switching Video](#).

Manipulating Digital I/O

Similar to the video signals you can switch digital outputs and generate digital input signals in your virtual test client.

Generate a simulated digital input:

To generate an input move your mouse pointer over the desired input channel. A left click will simulate a closing of the contact, a right click an opening. The contacts’ states are color coded according to this table:

Color	State
White	Unknown
Red	Closed
Green	Open
Gray	Unavailable

Generate a simulated digital output:


To generate an output move the pointer over the desired output signal. Left-clicking will set the output’s state to open, right-clicking to close. The outputs’ states are color coded according to this table:

Color	State
White	Unknown
Red	Closed

Green	Open
Yellow	Alternating (Can be set via Alternate Contact action)
Gray	Unavailable

Information

If the GeViIO client was connected to real DIO hardware, you could see the input signals changing in real time. Setting of the outputs would result in switching real loads.

StartOutp		Digital output state										
1	⬆️⬇️⬆️	1	2	3	4	5	6	7	8	9	10	11
17	⬆️⬇️⬆️	17	18	19	20	21	22	23	24	25	26	27
StartInp		Digital input state										
1	⬆️⬇️⬆️	1	2	3	4	5	6	7	8	9	10	11
17	⬆️⬇️⬆️	17	18	19	20	21	22	23	24	25	26	27
<input type="button" value="Refresh"/> <input type="button" value="Alarms"/>												
<input type="button" value="Conn"/> Current user rights		<input type="button" value="Rec"/> 0										
<input type="button" value="Dcon"/> 		<input type="button" value="Run"/> CrossSwitch(1,1,0)										
Communication log												
Send	>> CrossSwitch(7,3,0)											
Received	>> CrossSwitch(7,3,0)											
Send	>> OpenContact(2)											
Received	>> OpenContact(2)											
Send	>> InputContact(3,true)											
Received	>> InputContact(3,true)											

Actions

So far you only used GeViAPI Test Client's built-in functionality to interact with GeViServer. In this chapter you will learn to use GeViSoft actions to control the system.

GeViSoft actions can be sent by typing them into the text box in the lower middle of the GeVi-API Test Client's window. You can find a complete list of the possible actions in the documentation.

Hint

You can interactively generate actions and learn about their parameters by composing them in GeViSet. Therefore, open GeViSet, and connect to the server. Then navigate to *Server -> Named actions* and press *Add* in the window that pops up. In the window *Named action settings* you may press the button with the three dots ("...") to take you to the *Action settings* menu.

There you can choose any of the implemented actions and view their parameters and settings. To filter the actions by category choose one of the categories from the upper left list box. Hoover the mouse over any of the parameters to get a detailed description of it.

As an example select *Crossbar control* as a category and move to *CrossSwitch* to see the message's parameters on the right side.

The complete message is:

```
CrossSwitch(ID VideoInput, ID VideoOutput, Switchmode).
```

Cross Switching Video

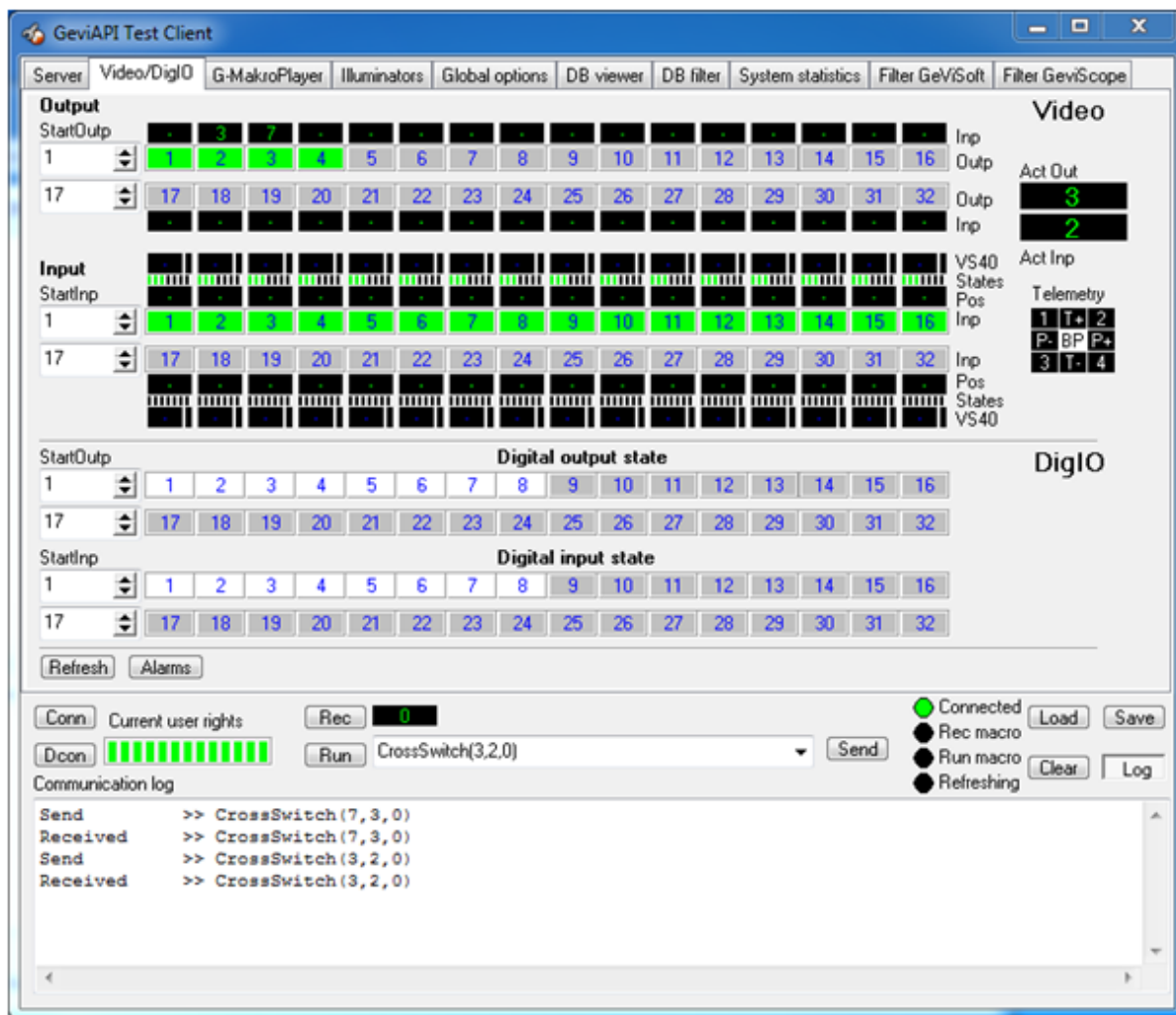
1. Route video from an input to an output -- To send the video from input 7 to output 3, do the following:

a) Type this action into the text box in the lower middle of the GeViAPI Test Client window and send it: `CrossSwitch(7, 3, 0)`

b) Make sure that the signal is routed accordingly by checking the output in the tab Video/DigIO

c) Route video input 3 to output channel 2. (`CrossSwitch(3, 2, 0)`)

2. Clear video output 2: `ClearVideoOutput(2)`



Cross switching video 1

Manipulating Digital I/O

1. Open contact 1 and close contact 2 -- The actions `OpenContact (ContactNumber)` and `CloseContact (ContactNumber)` can be used to set the digital outputs from GeViSoft.

a) To open contact 1 send the action: `OpenContact (1)`

b) In the Tab `Video/DigIO` of GeViAPI Test Client make sure that the indication of output one has turned to green

c) To close contact 2 send the action: `CloseContact (2)`

d) Make sure that the output turned red.

2. Simulate a closing of the input contact 3 and an opening of the input contact 5

a) `InputContact (3, true)`

b) Make sure that input 3 is signaling *closed* (red indication)

c) `InputContact (5, false)`

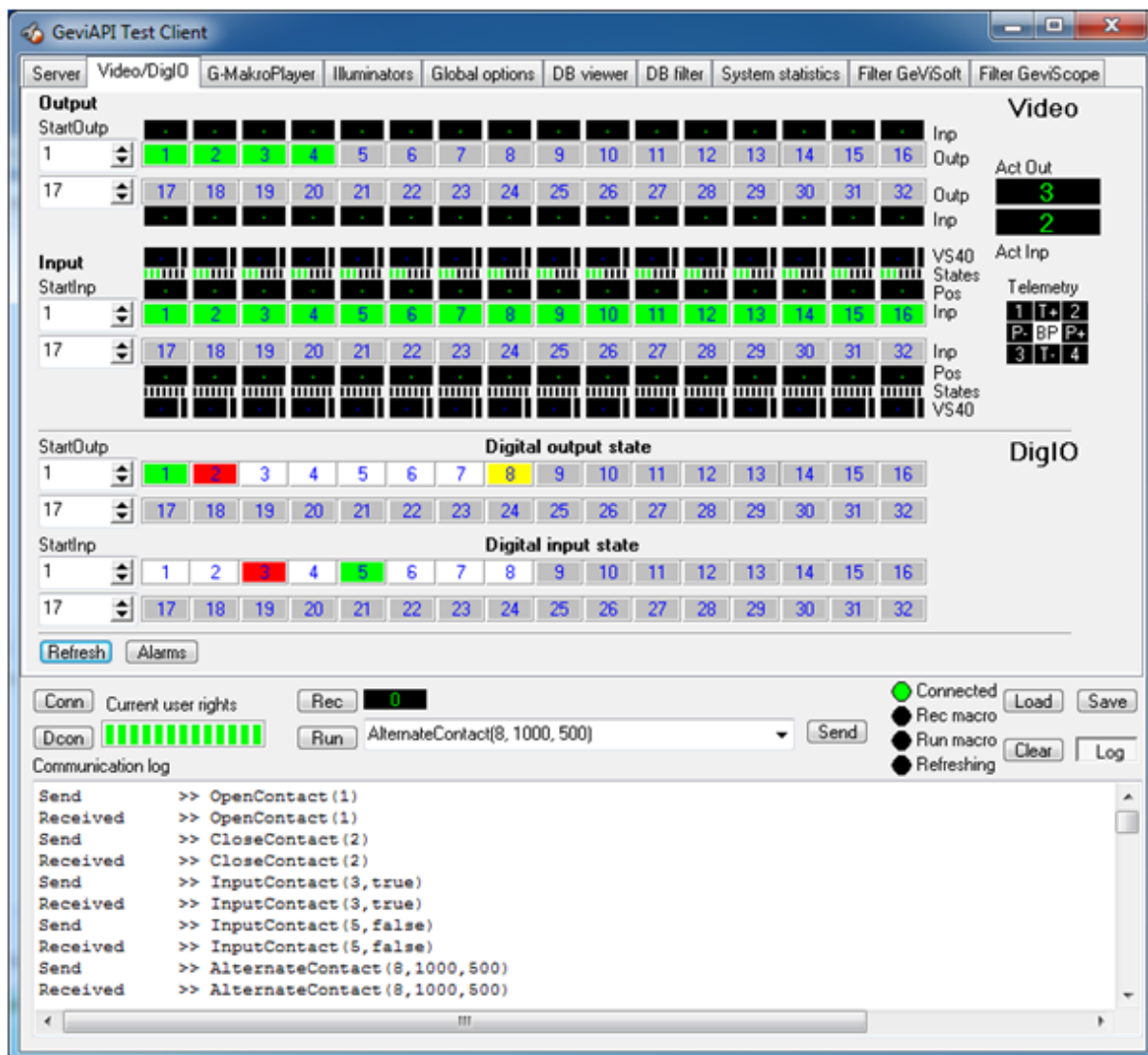
d) Make sure that input 5 is signaling *open* (green indication)

3. Alternating a contact -- Simulate a flash light on output 8

a) To alternate a contact, you can use the action `AlternateContact (ContactID, BlinkPeriod_in_ms, BlinkOnTime_in_ms)`

b) Send the command to flash the light with a frequency of 1 Hz and a duty cycle of 500ms: `AlternateContact (8, 1000, 500)`

c) Check that the contact is alternating – after pressing the *Refresh* button, the output 8 state should be *alternating* (yellow).



Manipulating digital IO

GETAS

In this chapter you will learn about GETAS, the Geutebrück Telnet Action Server. The GETAS component allows you to send and receive GeViSoft actions via telnet. Multiple clients can connect to one GeViServer at a time.

The telnet interface allows you to easily integrate simple applications into your GeViSoft infrastructure. Furthermore it offers an option to connect non-Windows platforms.

CAUTION

By default, GETAS is not active. To activate GETAS, open GeViSet and navigate to *Server -> GETAS*. In the *GETAS settings* window, you can then activate the component by checking *Enable TCP port*. By default GETAS will listen to port 7707. Leave the other settings unmodified and press *OK*. Send the altered setup to the server afterwards (*File -> Setup to server*).

CAUTION

To connect to GETAS, you need a telnet client. You can either use the Windows telnet client or a third party application like putty.

ADVICE

If you are using Windows 7, the telnet client is not activated by default. To activate it go to *Start -> Control Panel -> Programs and Features* and select the *Telnet Client* from the list box.

Now you can connect to GeViServer and send some actions.

Basic GETAS Usage

1. Connect to GeViServer via GETAS – Open a command window (cmd.exe) and start telnet. In a command window type: `telnet localhost 7707`

2. Make sure that your input is echoed locally by entering `set localecho`
3. You may want to press enter once to clear your screen if necessary.
4. Make sure that you started your GeViAPI Test Client and connected it to the GeViServer
5. Send an action to the server:
 - a) `CustomAction(42, "Hello GETAS")`
 - b) If you receive an echo of your action preceded by a 4; from the GeViSoft server, your configuration is working
 - c) Verify that you can also see the action in the GeViAPI Test Client's communication log. If you cannot see the message, make sure you are connected and your filter settings in the tab *Filter GeViSoft* are set correctly. To be sure, set the filter to accept all messages.
6. Monitor actions sent by other clients in your telnet session:
 - a) Send an action from GeViAPI Test Client: `CustomAction(23, "Hello GETAS client")`
 - b) Verify that you received the action in your telnet window.

Video and IO Control with GETAS

1. Now control your virtual VX3 by using GETAS – Make sure that GeViAPI Test Client is running while you issue commands via telnet and you can see the *Video/DigIO* tab. Your GeViIO_01 configuration should be the same as in chapter [Setting up GeViIO](#).

2. Route video input 7 to video output 2:

a) Connect to the GeViServer via telnet in a command window if not done it yet.

b) Send: `CrossSwitch(7,2,0)`

c) Make sure that the video signal is routed correctly in the *Video/DigIO* tab of GeVi-API Test Client

3. Copy the video signal from output 2 to have it also on output 4:

a) Send: `CopyCameraOnMonitor(2,4)`

b) Make sure, that input 7 is routed to output 2 and 4 in the GeViAPI Test Client

4. Clear the video output on channel 2:

a) Send: `ClearVideoOutput(2)`

b) Make sure the command worked (GeViAPI Test Client)

5. Close digital output contact 5:

a) Send: `CloseContact(5)`

b) Verify the result of the command in GeViAPI Test Client

GETAS Limitations

GETAS can be used for simple applications or integration of non-Windows clients. Nevertheless there is one limitation. The telnet connection is only established to one server at a

time. If GeViServer is running in a redundancy setup, the actions are not forwarded between the different servers.

WARNING

If you plan to integrate your GETAS solution into a complex setup with mirroring, you have to take care of the communication with the different servers on your own.

This has not to be considered if you are using the SDK functionality as described in chapter [SDK Usage](#). The SDK functions will take care of communicating with the various systems.

Action Mapping

Action mapping comes in handy if you need to trigger one action by another. Assume you want to switch on a beacon if a door is opened. The beacon is connected to a digital output 2 and the door to a digital input 3 (which opens together with the door). These actions can be mapped in the following way.

1. In GeViSet select *Server* → *Action mapping*
2. Press *Add* in the *Action mapping* window. The *Action mapping settings* window will open.
3. Press the ... button to add a new input action and choose the *Digital contacts* category in the *Action settings* window
4. Select the *InputContact* action and set the parameters *GlobalContactID* to 3 and *ChangedTo* to *false*. Set *Caption* to *door contact has opened* and press *OK*.
5. Press the + button to set the output action in the *Action mapping settings* window
6. To flash a beacon, the output signal must alternate between on and off. This can be achieved with the *AlternateContact* action.
7. Set the *AlternateContact* action's parameters to *GlobalContactID* = 2, *BlinkPeriod* = 1000ms, and *BlinkOnTime* = 500ms. Enter *blink the beacon* as *Caption*.
8. Send the setup to the GeViServer
9. Test the mapping by sending the action `InputContact(3, false)` either by GETAS or inside GeViAPI Test Client. You should see the mapped action `AlternateContact(2, 1000, 500)` delivered by the GeViServer directly afterwards. You can also check the output's status in GeViAPI Test Client's *Video/DigIO* tab after hitting the refresh button.

10. To switch off the beacon after the door has closed, you need to map another action pair in GeViSet. For that, map `InputContact(3, true)` to `CloseContact(2)` and send the setup to the GeViServer.

11. Check if the beacon is switched off immediately after sending the action `InputContact(3, true)`

Please note that you can map multiple actions to one input. This allows you to realize more complex setups.

Timer (optional)

GeViSoft allows you to configure timers which can schedule actions, giving you a versatile tool for custom applications. You can configure different types of timers in GeViSet. Timers internally count in “ticks”. Each tick equals one millisecond.

Types of Timers

- **Once timer (single shot)** – this timer counts down for the number of main ticks after being started and at the end fires the configured action
- **Periodical timer** – this timer allows triggering actions periodically every time it reaches the number of main ticks. After firing the action, the timer restarts counting ticks from zero.
- **Periodical timer with embedded tick** – this timer acts similar to the simple periodical timer. In addition, it can fire a second action on reaching the “embedded tick” count. As an example, you could realize switching on and off of a beacon light with a timer like this. Simply close the output at the embedded tick and open it at the main tick.

System timer settings

Active: ☒

Name: BeaconTimer

Description: Timer to switch the beacon off and on

Timer ID: 1

Timer type:

☐ Once

☐ Periodical

☒ Periodical with embedded tick

Main tick on: 1000 ms

Embedded tick on: 500 ms

On embedded tick: Light Beacon Edit ... Remove

On main tick: Turn Beacon off ☐ On Start Edit ... Remove

OK Cancel

Timer 1

Configuring a Timer

1. To realize the beacon's timer select „*Server* -> „*Timer* in GeViSet.
2. Add a new timer. Make sure, the *Active* checkbox is ticked. Name the timer *BeaconTimer* and describe it as *Timer to toggle a beacon on digital output 2*
3. Set the *Timer type* to *Periodical with embedded tick*, the main tick to occur every 1000ms, and the embedded tick to occur every 500ms. This will generate a timer with two ticks per second and a spacing of 500 ms in between.
4. Press the *Edit* button *On embedded tick* to set the action that shall occur with every embedded tick. Chose *OpenContact* for the *GlobalContactID 2* and give the action a caption like *turn of beacon*.
5. For the main tick, set the action to *CloseContact* for the same output and the caption to *turn on beacon*.
6. Send the new setup to the server and switch to GeViAPI Test Client
7. You can start the timer by sending the *StartTimer* action. This action takes two parameters, the *TimerID* and the *TimerName*. If you want to address a timer by its ID, just send an empty *TimerName*. Send `StartTimer(1, "BeaconTimer")`
8. In the *Video/DigIO* tab, you should see that output 2 toggles with a frequency of 1 Hz.
9. To stop the timer, send a `StopTimer(1, "BeaconTimer")` action.

NOTICE

Hint for using *StartTimer* and *StopTimer* actions:

GeViSoft first tries to evaluate the timer by *TimerName* and only if no name is given by ID. If you

use a non existing name, the timer will not be started, even if you state the right ID. If you want to start a timer by ID, send an empty string as name (e.g. *StartTimer(1, "")*).

Events (optional)

Events can be used to control complex behavior caused by a start condition as a trigger. The GeViSoft event handling implementation allows a very flexible setup of event triggers and resulting actions.

Event settings

Active: ☒

Name:

Description:

Event ID:

Trigger

☒ Enabled

☒ Repeat Actions

☒ Adjust auto stop time

☒ Adjust start time

☒ Stop before

Auto stop

☒ Enabled

Stop after: ms

Time ranges

☒ Autostop on leave valid time ranges

☒ rest of time

Start by

+

..

-

▲

▼

Stop by

+

..

-

▲

▼

On start

+

..

-

▲

▼

On stop

+

..

-

▲

▼

OK

Cancel

You can add events in GeViSet (*Server -> Events -> Add*).

Options for Events

Option	Description
Active	Events can only be triggered if marked <i>Active</i>
Trigger Enabled	If <i>trigger</i> is enabled, the event is restarted if the <i>Start by</i> condition occurs again.
Repeat Actions	If the <i>Start by</i> condition occurs, the <i>On start</i> action is invoked
Adjust auto stop time	If the <i>Start by</i> condition occurs, the elapsed <i>Auto stop</i> time is reset to zero
Adjust start time Stop before	If checked, the start time is adjusted on retriggering
Auto stop Enabled	If enabled, the event stops after the time frame set in <i>Stop after</i>
Stop after	Period of time after which the event automatically stops
Auto Stop on leave of valid time ranges	Events can be activated for certain time ranges only. If this option is checked, the event automatically stops if the valid time ranges are left
Time range field	List of all the time ranges where the event is activated. Note that if no time range is given, the event cannot be triggered!
Start by	List of actions that trigger the event. If multiple actions are configured, any of the actions will trigger the event on its own (logical OR operation)
Stop by	List of actions that terminate the event. If multiple actions are configured, any of the actions will stop the event on its own (logical OR operation)
On start	List of actions that are all executed on event start (logical AND)
On Stop	List of actions that are all executed on event termination (logical AND)

Configuring an Event

1. Here is an example how to configure an event that routes video signals based on digital input – closing of contact 3 triggers the routing of video input 3 to video output 2. After 5 seconds, the event stops and video output 2 is cleared. The event will be configured for automatic retriggering. Here are the settings:

Event settings

Active: ☒

Name:

Description:

Event ID:

Trigger

☒ Enabled

☒ Repeat Actions

☒ Adjust auto stop time

☐ Adjust start time

☐ Stop before

Time ranges

☐ Autostop on leave valid time ranges

☒ rest of time

Auto stop

☒ Enabled Stop after: ms

Start by

+ .. - ▲ ▼

Stop by

+ .. - ▲ ▼

On start

+ .. - ▲ ▼

On stop

+ .. - ▲ ▼

OK Cancel

Example of an Event

2. The actions for *Start by*, *On start*, and *On stop* are:

a) Start by: Contact 3 closed -> `InputContact(3, true)`

b) On start: Route video In 3 to Video out 2 -> `CrossSwitch(3,2,0)`

c) On stop: Clear video output 2 -> `ClearVideoOutput(2)`

3. After the setup has been sent to the GeViServer, the event can be tested with the GeViAPI Test Client

4. If you left click input contact 3 the event is started. You will see that video input stream 3 is routed to video output 2. After 5 seconds the output is cleared again. You can also see the event being started in the communication log.

5. The event can be retriggered. If you left click input 3 again while the event is running, the 5 second auto stop time starts over again.

6. You can also start the event by sending a *StartEvent* message (`StartEvent(ID, "MessageName")`).

Alarms (optional)

Due to the large amount of video cameras connected to modern video surveillance systems, operators cannot observe all the streams at the same time. Moreover, only certain incidents are of interest or need action. Therefore, it is helpful that a preselection of the video material shown to the user is carried out by the system. Often special actions have to be taken if a particular situation is happening. As an example assume that a parking lot with a barrier at the entrance is being monitored. The operator is supposed to open the barrier after making sure that a waiting vehicle is allowed to enter. Normally, the operator would have to watch the stream of the camera permanently and act on it. In cases like this Geutebrück systems can assist by providing alarms. Alarms are very similar to events, but offer more versatile options for customizing and defining required user interaction.

Alarm Options

GeViSet offers several options for alarms.

Option	Description
Name	Alarm name – can be used in actions
Description	Field for the description of an alarm
Alarm ID	Alarm identifier -- can be used in actions
Active	Alarms can only be triggered if marked Active
Priority	Alarms can have a priority from 1 (high) to 10(low). A higher priority alarm will displace a lower priority one if configured to be shown on the same monitor group
Monitor Group	Several monitors that are addressed as a group for easier administration
Cameras	List of cameras that are relevant for the alarm. Their pictures are shown on the monitor group in case an alarm occurs
Retriggerable	If checked, the alarm can be retriggered by its initial activator.
Popup (Retrigger)	

Option	Description
Undo acknowledge (Retrigger)	If set, the alarm has already been acknowledged and the alarm is retriggered, the state will be reset to not acknowledged.
User specific (Retrigger)	If checked, a custom list of actions can be added which will be executed on a retrigger event of the alarm
Start by Action	List of actions. Any of the actions will start the alarm (logical OR)
On start Action	List of actions. All of the actions will be sent on start (logical AND)
Acknowledge by Action	List of actions. Any of the actions will acknowledge the alarm (logical OR)
On acknowledge Action	List of actions. All of the actions will be sent on acknowledge (logical AND)
Quit by Action	List of actions. Any of the actions will quit the alarm (logical OR)
On quit Action	List of actions. All of the actions will be sent on quit (logical AND)

Configuring an Alarm

Configure an alarm for the parking lot scenario as described above. Assume that the detection of a vehicle is done by a sensor on digital input 1 (vehicle is detected on close). After checking if the vehicle may enter the operator must open the barrier. To do so he acknowledges the alarm by pushing a button connected to digital input 2. As the barrier is controlled by digital output 1 the *On acknowledge* action must open this contact. After the vehicle has passed, the operator must quit the alarm by pushing a button connected to digital input 3. *On quit* the barrier must be closed by closing digital output 1. The parking lot is surveilled by two cameras on inputs 4 and 7. During the alarm, these must be routed to outputs 1 and 2.

1. Alarms are displayed in *Monitor Groups*. First define one In GeViSet.

- a) *Server -> Monitor groups -> Add*
- b) Set the group's Name and Description to *MonitorGroup1*
- c) *Add* video outputs 1 and 2 to the group

d) Leave the rest of the settings as they are

2. Add a new alarm in GeViSet: *Server -> Alarms -> Add*

a) In the General tab, set *Name* and *Description* to *ParkingLot*

b) Press the *Monitor group* button and add *MonitorGroup1*

c) Add *Video input 4* and *Video input 7* to Cameras

Alarm settings

☒ Active Name: ParkingLot

Description: ParkingLot

Alarm ID: 1

General Actions

Priority: 1

Monitor group: MonitorGroup1

Cameras:

Global ID	Name
4	Video input4
7	Video input7

Add ... Remove ▲ ▼

Retrigger

☐ Retriggerable

☒ Popup

☐ Undo acknowledge

☐ User specific:

OK Cancel

Alarm Settings 1

d) In the *Actions* tab, set the *Start by* action to *InputContact*, the *GlobalContactID* to 1 and *ChangedTo* to true. Add the *Caption vehicle detected*

e) Set the *Acknowledge by* action to *InputContact*, the *GlobalContactID* to 2 and *ChangedTo* to true. Add the *Caption button acknowledged pressed*

f) Set the *On acknowledge* action to *OpenContact*, and the *GlobalContactID* to 1. Add the *Caption opening barrier*

g) Set the *Quit by* action to *InputContact*, the *GlobalContactID* to 3 and *ChangedTo* to true. Add the *Caption button quit pressed*

h) Set the *On quit* action to *CloseContact*, and the *GlobalContactID* to 1. Add the *Caption closing barrier*

Alarm settings

☒ Active Name: ParkingLot

Description: ParkingLot

Alarm ID: 1

General Actions

Start by	Acknowledge by	Quit by
InputContact(1, true) - vehicle detected	InputContact(2, true) - button ack press...	InputContact(2, true) - button quit pressed
<div>+ .. - ▲ ▼</div>	<div>+ .. - ▲ ▼</div>	<div>+ .. - ▲ ▼</div>
On start	On acknowledge	On quit
	OpenContact(1) - opening barrier	CloseContact(1) - closing barrier
<div>+ .. - ▲ ▼</div>	<div>+ .. - ▲ ▼</div>	<div>+ .. - ▲ ▼</div>

OK Cancel

Alarm Settings 2

3. Send the setup to the server

4. Test the new alarm in GeViAPI Test Client

a) Clear video outputs 1 and 2 by right-clicking on them.

b) Simulate the arrival of the vehicle by left-clicking input 1.

c) Check if the alarm is triggered by verifying that streams 4 and 7 are displayed on monitors 1 and 2. Note that the outputs' color changed to red which indicates an alarm feed. You should also find the `AlarmStarted()` action in the *Communication log*

d) Acknowledge the alarm and open the barrier by left-clicking input contact 2. Make sure that this leads to the opening of output 1 and an `AlarmAcked()` action appearing in the log.

e) Quit the alarm by left-clicking input contact 3. The video outputs' color should change to green as the alarm has finished. The barrier (output 1) should have closed.

Switching Video

Though monitor groups date back to analogue video recording, the idea behind them comes in handy when complex situations are to be presented to operators. In modern CCTV systems most of the sources are digital ones and the viewers run as software on dedicated consoles. Nevertheless the concept of monitor groups can still be reproduced with Geutebrück's systems. The standard viewer – GscView – can be remote controlled to show predefined scene setups in a way similar to monitor groups.

In this chapter you will learn how to switch between two user defined GscView scenes by triggering a GeViSoft alarm. You will have a normal 4-by-4 scene displaying 16 channels of live footage from a GeViScope. On triggering an alarm in GeViSoft, GscView will be switched to a 2-by-2 scene displaying predetermined video channels.

Scenario

Assume the following situation, which is closely related to [Configuring an Alarm](#) in chapter *Alarms*:

Configure an alarm for the parking lot scenario. Assume that the detection of a vehicle is done by a sensor on digital input 1 (vehicle is detected on close). After checking if the vehicle may enter, the operator must open the barrier. This happens on acknowledging the alarm by pushing a button connected to digital input 2. As the barrier is controlled by digital output 1, the *On acknowledge* action must open this contact. After the vehicle has passed, the operator must quit the alarm by pushing a button connected to digital input 3. *On quit* the barrier have to be closed by closing digital output 1. The parking lot is surveilled by two cameras on inputs 4 and 7. During the alarm, these must be routed to outputs 1 and 2 of a 2-by-2 scene *MyScene* in GscView. Before and after the alarm, all 16 GeViScope channels should be displayed in a 4-by-4 scene *MyStartScene* in GscView.

Prerequisites

1. Please setup the GeViScope SDK on your development machine if you have not done it yet.
2. Configure GscView as described in the chapter *Remote control GscView by action* in the GeViScope SDK documentation. Check that from GSCPLCSimulator you can switch between the scenes.
3. Configure a connection to GeViScope as described in [Connection to GeViScope \(optional\)](#).
4. You should now have a GscView setup with two scenes: *MyStartScene* and *MyScene* that can be remote controlled.

Configuring the Alarm

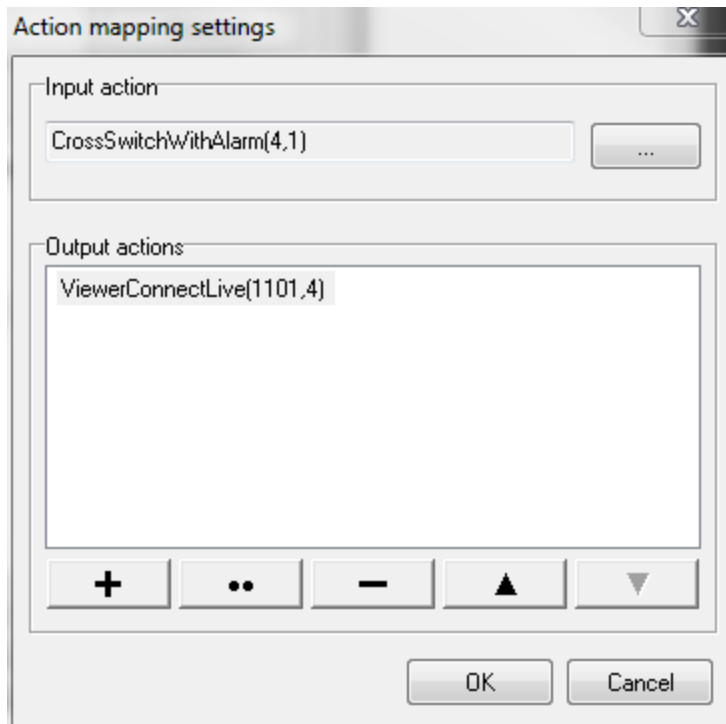
1. Configure the alarm as described in [Configuring an Alarm](#).
2. After that, the monitor group must be mapped to a GscView Scene. GeViSoft uses the *CrossSwitchWithAlarm* action to route the video to the monitor group internally. Therefore these actions must be mapped to GSCViewer Control actions (e.g. VC Change Scene By Name). This is done in GeViSet by adding a new *Action mapping*:
 - c) This changes the scene in the viewer. After that, channel 4 must be routed to viewer 1101 in the scene. For that, add another output action to the set by pressing the + button:
 - d) Add the *Viewer connect live* action with the *GeviScope alias* = *GEVISCOPe*, the *viewer* = *1101*, the *channel* = *4*, and *Caption* = *ViewerConnectLive (1101,4)*

a) As *Input action* select *CrossSwitchWithAlarm* with *VideoInput* = 4, *VideoOutput* = 1, and *Caption* = *CrossSwitchWithAlarm(4, 1)*

b) To change the scene in the viewer there are different possibilities. You can either call *VCChangeSceneByName* or directly connect a live stream to a viewer number. This is done by sending a *ViewerConnectLive* action. Here, channel 4 must be routed to viewer 1101 in the scene. For that, add an output action to the set by pressing the + button:

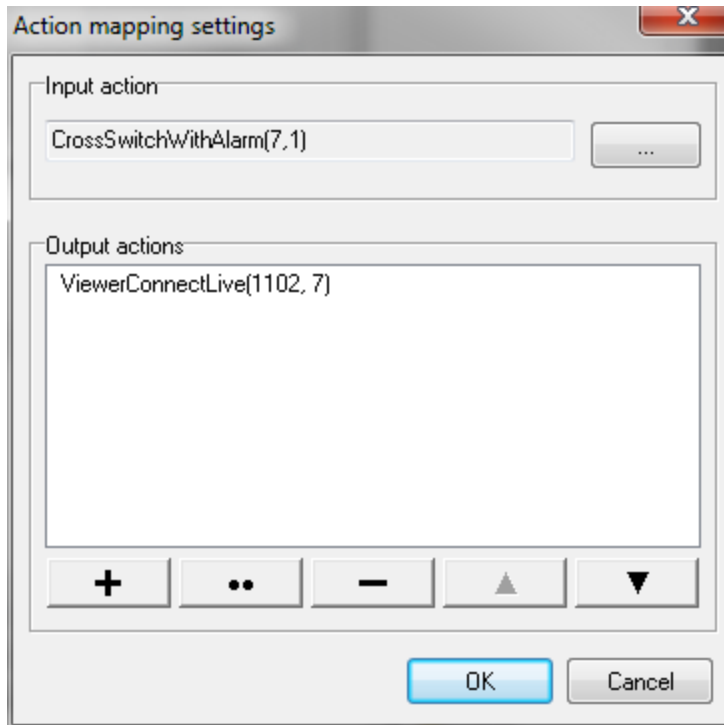
c) Add the *Viewer connect live* action with the *GeviScope alias* = *GEVISCOPe*, the *viewer* = 1101, the *channel* = 4, and *Caption* = *ViewerConnectLive (1101, 4)*

d) The Action mapping settings window should look like this:



Switching Video 1

e) Now repeat the process for the *CrossSwitchWithAlarm* action for video input 7 and viewer 1102.



f) If executed, the mappings above will switch the scene to *MyScene* in GscView and route the video channels to the respective viewer. Execution of the `CrossSwitchWithAlarm` actions takes place at the moment of triggering the alarm in GeViSoft.

3. After quitting the alarm the 4-by-4 scene *MyStartScene* must be reloaded in GscView, according to the `scenario`. This can be done as an *On quit* action of the GeViSet alarm:

a) In the GeViSet *Alarm settings* of the *ParkingLot* alarm, add a *VC change scene by name* action to the *On quit* list.

b) Chose the action from *GSC: Viewer action* and set *GeviScope alias* to *GEV-ISCOPE*, *viewer* to *1000*, *scene* to *MyStartScene*, and *Caption* to *VCChangeSceneByName(1000, MyStartScene)*).

c) Send the setup to the server.

4. Open GeViAPI Test Client and GscView to test your new configuration. On starting the alarm by left clicking input 1 in GeViSet, the scene should switch to *MyScene* in GscView with channel 4 being displayed in viewer 1101 and channel 7 in viewer 1102. On quitting the alarm by left clicking input 3 in GeViAPI Test Client, the scene should switch back to *MyStartScene*.

SDK Usage

Introduction

It is recommended to be familiar with the GeViSoft system, the possibilities of modern video surveillance systems and video management systems in general. Before starting programming your custom GeViSoft application, you should understand the basics of action, alarm, and event handling in GeViSoft, as well as the principles of client-server network communication.

The following sections support you with some suggestions and hints about using the SDK interfaces.

General Hints

You can always monitor the actions sent by the GeViServer or your application inside the GeViAPI Test Client. Furthermore, this application allows you to send actions and database queries. It is located in the `%GEVISOFTSDKPATH%`.

NOTICE

On a development system it is recommended to start GeViServer with the `startserver.bat` script or from a command prompt in console mode (`geviserver.exe console`). This allows you to monitor the server's output during your development.

WARNING

Make sure to delete all objects that are created inside of DLLs. The SDK offers a `DeleteObject()` method for these objects.

NOTICE

Callback functions which are called out of the SDK DLLs are called from threads. These were created inside the DLLs. Variables and pointers that are passed as arguments of the callback may not be used outside the callback context. They are only valid for the duration of the callback call.

NOTICE

Structures that are used as arguments for SDK functions should always be initialized by use of the function `memset()`. If the structure contains a size or `structsize` element, then it has to be initialized with the `sizeof()` function.

Overview of the SDK's Interfaces for C++ and Delphi users

NOTICE

The following paragraphs describe the SDK usage from C++ and Delphi. For a description of the .Net Interfaces see chapter [C# and .Net specifics](#)

GeViProcAPI

The SDK is based on two DLLs and the corresponding headers. The GeViProcAPI.dll in connection with the GSActions.dll implements all the SDK's functionality. GSActions.dll is used to allow the interoperability between GeViSoft and GeViScope. It makes the GeViScope actions available to your GeViSoft application. The corresponding headers and Pascal files allow you to access the functions provided by these DLLs directly.

GeViSoft is a client/server architecture and based on a central database managed by the GeViServer. This is reflected in the function calls provided by the SDK. There are four major function types declared in GeViProcAPI. They can be distinguished by their prefixes:

GeViAPI_Database_

These database functions allow you to interact with GeViSoft server directly. They are the ones you normally use for developing your application.

Example: GeViAPI_Database_Connect() is used to connect your application to the server.

GeViAPI_DeviceClient_

The DeviceClient functions are used by the GeViIO client internally. They are usually not of interest for SDK developers.

GeViAPI_SetupClient_

The SetupClient functions are used by GeViSet to change the server setup. They are usually not of interest for SDK developers.

GeViAPI_

These are general helper functions needed for carrying out standard tasks in your application.

Example: GeViAPI_FreePointer() to release memory allocated by your objects inside the DLL threads.

GeViProcAPI provides flat function calls for communicating with a GeViServer. To give you a more convenient and object oriented option to develop your application, another abstraction layer has been added to the SDK. This layer hides the flat function calls to the GeViProcAPI from you. Its functionality can be found in the GeViAPIClient headers and C++ files.

For a comprehensive description of these functions, please consult the GeViSoft API Documentation which is delivered with the GeViSoft API SDK.

GeViAPIClient

GeViAPIClient as an abstraction layer uses the flat functions provided by GeViProcAPI and encapsulates them into a CGeViAPIClient class. You can instantiate an object of this class and use its provided methods to handle the communication with the GeViServer.

For a comprehensive description of these functions, please consult the GeViSoft API Documentation which is delivered with the GeViSoft API SDK.

Configuring your IDE for GeViSoft Projects

Visual Studio 2008, C++

- 1.) Add GeViSoft's header and cpp files to your project.
(You can do this by dragging and dropping the GeViScopeSDK\Include folder and the GeViSoftSDK\Include folder from %GEVISOFTSDKPATH%\Examples\VS2008CPP to your project.)
- 2.) Add the SDK's include files to your project by adding
`$(GEVISOFTSDKPATH)\Examples\VS2008CPP\GeViScopeSDK\Include`
and
`$(GEVISOFTSDKPATH)\Examples\VS2008CPP\GeViSoftSDK\Include`
to your *Configuration Properties -> C/C++ -> General -> Additional Include Directories*
- 3.) In the *Configuration Properties -> C/C++ -> Preprocessor* tab add the *Preprocessor Definition* `GEVI_GSC_INCLUDE`
- 4.) In the project's properties TAB *Configuration Properties -> Linker -> General* add
`$(GEVISOFTSDKPATH)\Examples\VS2008CPP\GeViScopeSDK\lib`
and
`$(GEVISOFTSDKPATH)\Examples\VS2008CPP\GeViSoftSDK\lib`
to the *Additional Library Directories* of your project
- 5.) In the project's properties TAB *Configuration Properties -> Linker -> Input -> Additional Dependencies* add `GeViProcAPI.lib` and `GscActions.lib`
- 6.) Make sure that your output file can find the path to GeViProcAPI and GscActions DLLs. It is recommended to set *Configuration Properties -> Linker -> General -> Output File* to `$(GEVISOFTSDKPATH)\$(ProjectName).exe` or copy the DLLs into the application's folder.
- 7.) Set the *Configuration Properties -> Debugging -> Command* to your executables name:

`$(GEVISOFTSDKPATH) \$(TargetName) $(TargetExt)`

NOTICE

Please make sure that you select the correct configuration when setting properties. Best practice is to adopt the GeViSoft settings to *All Configurations*

NOTICE

Please notice that Visual Studio refers to environment variables in the form \$(VAR) whereas Windows uses the %VAR% notation. Take this into account if you use the GEVISOFTSDKPATH variable.

Visual Studio 2010, C++

The following guide is suitable for console projects or MFC projects. If you wish to build Windows Forms or C++/CLI applications more configurations might be necessary.

- 1.) Add GeViSoft's header and cpp files to your project.
(You can do this by dragging and dropping the `GeViScopeSDK\Include` folder and the `GeViSoftSDK\Include` folder from `%GEVISOFTSDKPATH%\Examples\VS2010CPP` to your project.
- 2.) Add the SDK's include files to your project by adding
`$(GEVISOFTSDKPATH) \Examples\VS2010CPP\GeViScopeSDK\Include`
and
`$(GEVISOFTSDKPATH) \Examples\VS2010CPP\GeViSoftSDK\Include`
to your *Configuration Properties -> VC++ Directories -> Include Directories*
- 3.) Add the SDK's library files to your project by adding
`$(GEVISOFTSDKPATH) \Examples\VS2010CPP\GeViScopeSDK\lib`
and

`$(GEVISOFTSDKPATH)\Examples\VS2010CPP\GeViSoftSDK\lib`
to your *Configuration Properties -> VC++ Directories -> Library Directories*

4.) In the project's properties TAB *Configuration Properties -> Linker -> Input -> Additional Dependencies* add `GeViProcAPI.lib` and `GscActions.lib`

5.) Make sure that your output file can find the path to GeViProcAPI and GscActions DLLs. It is recommended to set *Configuration Properties -> Linker -> General -> Output File* to `$(GEVISOFTSDKPATH)\$(ProjectName).exe` or copy the DLLs into the application's folder.

6.) Set the *Configuration Properties -> Debugging -> Command* to your executables name:
`$(GEVISOFTSDKPATH)\$(TargetName)$(TargetExt)`

Common Tasks

This chapter describes several common tasks you might need to carry out during your development.

These are described in pseudo code and C++. For a description of the .Net API see chapter [C# and .Net specifics](#).

Connecting to a GeViServer

The first example shows you how to connect to a GeViServer by using the flat API calls from GeViProcAPI. The second and recommended method shows you how to establish the connection with the help of a GeViAPIClient object.

Connecting using GeViProcAPI calls

Pseudo code

1. Declare a database handle
2. Encrypt the password string
3. Create a remote database object inside the DLL
4. Connect to the database object created inside the DLL

C++, direct GeViProcAPI calls:

```
// declare a string to hold the password hash
// (32 byte + '\0')
char encodedPassword[33];
```

```

// declare a database handle
GeViAPI_Namespace::HGeViDatabase database;

// encode the password
GeViAPI_EncodeString(encodedPassword, "masterkey",
    sizeof(encodedPassword));

// create a remote database object inside the DLL
// to access a GeViSoft database
GeViAPI_Database_Create(database, "localhost",
    "127.0.0.1", "sysadmin",
    encodedPassword, "", "");

if (database) // database successfully created
{
    // Connect functions result
    TConnectResult result;

    // Connect to the database object.
    GeViAPI_Database_Connect(database, result,
        NULL /* your callback here! */,
        NULL /* your instance here! */);
    if(result == connectOk)
        std::cout << "Connection established!";
}

```

Connecting using GeViAPIClient Objects (recommended)

Pseudo code

1. Declare a GeViAPIClient wrapper object
2. Declare and define a connection callback function to monitor the connection progress (this function will be called from inside the DLL and return a progress state in its arguments)

3. Encrypt the clear text password
4. Create an instance of the GeViAPIClient wrapper object
5. Call the wrapper's connect method
6. Check If the connect method returned a success

C++, GeViAPIClient calls:

1. Connection handling

```
// wrapper around a GeVi API client object
GeViAPIClient* m_APIClient;

// declare a string to hold the password hash
char encodedPassword[33];
GeViAPIClient::EncodePassword(encodedPassword,
                               "my password",
                               sizeof(encodedPassword) );

// create an new instance of the wrapper
m_APIClient = new GeViAPIClient("My GeViServer",
                                "127.0.0.1", "sysadmin",
                                encodedPassword, NULL, NULL);

if(m_APIClient)
{
    // connect to the server - ConnectProgressCB is your callback
    TConnectResult ConnectResult =
    m_APIClient->Connect(ConnectProgressCB, this);
    if(ConnectResult == connectOk)
    {
        // Connection successfully established
        // Do your work here.
    }
}
```

```
}
```

2. Callbacks

```
// Callback function for connect progress display
bool __stdcall ConnectProgressCB(void* Instance,
                                int Percentage,
                                int Percent100)
{
    if(Instance == NULL)
    {
        return (true) ;
    }
    // Call the callback method of your class
    // object's instance
    CYourClass* yourClass = (CYourClass*)Instance;
    return ( yourClass->ConnectProgress(
        Percentage, Percent100) ) ;
}

// Your class's callback
bool CYourClass::ConnectProgress(int percentageLower,
                                int percentageUpper)
{
    // Do s.th., e.g. show a Progress Ctrl.
    return (true) ;
}
```

Connection Monitoring

GeViSoft offers methods to monitor if your connection is still established. It is advisable to monitor the connection from your application and try a reconnect if it breaks down.

You can use the `sendPing()` method for connection monitoring which returns *true* if the connection is still established and *false* if not.

Best practice is to cyclically call `sendPing()` from a separate thread and handle the reconnection from inside this thread if necessary.

Monitoring connections is implemented in the SDK's example `CPP_MonitoredConnectionClient`.

Monitoring a Connection

Pseudo code

1. Create a separate thread inside your application if a connection should be established
2. Inside this thread, DO:
 - a. Send a ping to the server
 - b. IF the result of the ping is NOT true: try a reconnect
 - c. Sleep for a given time (e.g. 10s)
3. UNTIL the connection should be terminated

C++ Example

```
// Prerequisite:
// GeViAPIClient* m_APIClient
// must already be created and connected.
//
// Run this method inside a separate Thread!
int MonitorConnection()
{
    const int reconnectionPeriod_in_ms = 10000;
    bool result;
    while(true) {
```

```
result = m_APIClient->SendPing();  
if (result == false)  
{  
    // TODO: notify your user here.  
    // Try a reconnect:  
    m_APIClient->Connect(YourConnectCallbackCB, this);  
}  
Sleep(reconnectionPeriod_in_ms);  
}  
return 0;  
}
```

Message Handling

After you have established your connection you are ready to exchange messages with the server.

Message Representation

There are two possible representations of messages in GeViSoft. They are either stored in a binary form or as an ASCII string. The API offers methods to convert between these two representations. These methods are defined in the MessageBase header, C++, and Pascal files.

Table of conversion methods between message representations.

CGeV- iMessage::ReadASCIIMessage	Converts an ASCII string into a CGeViMessage
CGeV- iMessage::WriteASCIIMessage	Converts a CGeViMessage into an ASCII string
CGeViMessage::ReadBinMessage	Converts a binary representation of a message into a CGeV- iMessage
CGeV- iMessage::WriteBinMessage	Converts a CGeViMessage into its binary representation

Action Messages

Creating Action Messages

You can create an action message in two ways. One is by calling its predefined action constructor directly. The other is by converting an ASCII or binary representation into a new action object. The predefined constructors are located in the Actions header, C++, and Pascal files.

Actions can be considered as either being direct commands from the client to the GeViServer to control its periphery or as notifications which are sent from the server to the client to indicate state changes of logical or physical components. In contrast to actions, there are state queries and database queries. These are treated separately in the chapters [State Queries](#) and [Database Queries](#).

1. Example for a directly created CustomAction message (constructor from Actions.h/cpp)

```
CGeViMessage* gevimessage = new  
    CActCustomAction(123, "Hello GeViSoft!");
```

2. Example for a CustomAction message created from a string

```
int bytesRead;  
std::string buffer("CustomAction(123, \"Hello GeViSoft!\")");  
CGeViMessage* gevimessage =  
    CGeViMessage::ReadASCIIIMessage(buffer.c_str(),  
                                     buffer.size(),  
                                     bytesRead );
```

3. Example for the ASCII output of a binary action message:

```
// gevimessage is the binary message
```

```

char* buffer;
const int bufferlength = GEVI_MAXACTIONLENGTH;
int numBytesReceived;
buffer = new char[bufferlength];
gevimessage->WriteASCIIMessage(buffer,
                                bufferlength,
                                numBytesReceived);
std::cout << buffer << std::endl;

```

4. Example of creating a GeViScope Action Message

GeViScope messages can also be created inside GeViSoft directly. This is needed to allow the interoperability of GeViSoft and GeViScope.

The GeViScope message constructors can be found in the GscActions header. They are implemented inside the *GscActions* DLL. GscActions can be created by calling the CActGscAction constructor:

```

CGeViMessage* gevimessage = new CActGscAction(
    "YourGscServerName",
    GscAct_CreateCustomAction(1, L"Hello GeViScope!") );

```

NOTICE

Please note that "GscServerNameAlias" is the alias name you configured for the connection in GeViSet.

Sending Action Messages

The next example shows you how to send a message to the GeViSoft server. As a prerequisite, a GeViAPIClient object must already be created and connected to the server.

C++ Example:

```

GeViAPIClient* m_APIClient

```

```

//must already be created and connected
/*
...
*/
CGeViMessage* gevimessage = new CActCustomAction(
    123, "Hello GeViSoft!");
if (gevimessage)
{
    m_APIClient->SendMessage(gevimessage);
    // Don't forget to delete objects you create inside the DLL
    gevimessage->DeleteObject();
}

```

Receiving Action Messages

This example shows you how to receive a message from GeViSoft. As a prerequisite, a GeVi-APIClient object must already be created and connected to the server. Furthermore, a database notification callback function must be defined. This callback function will be called from inside the GeViProcAPI DLL whenever a notification from the server is received.

Pseudo code

1. Define the callback
2. Define the callback's handler method
3. Register your callback with the GeViAPIClient connection's object.
4. Handle the received notifications in you handler method.

C++ Example:

1. Define the callback

```

void __stdcall GeViDatabaseNotificationCB(void* Instance,
                                         TServerNotification Notification,
                                         void* Params)
{
    if(Instance == NULL)
        return;
    // calling the callback method of yourClass object's instance.
    // As an example, CYourClass might be CMainWin for an MFC Application
    CYourClass* yourClass = (CYourClass*)Instance;
    yourClass->DatabaseNotification(Notification, Params);
}

```

2. Define the callback's method

```

void DatabaseNotification(TServerNotification Notification,
                         void* Params)
{
    // Check if we received a message. It might also be another
    // notification like a change of setup or shutdown of the server
    if(Notification == NFServer_NewMessage)
    {
        // create the message if possible
        // (the message is freed again in the main thread context)
        CGeViMessage* gevimessage;
        TMessageEntry* messageEntry =
            reinterpret_cast<TMessageEntry*>(Params);
        int noOfBytesRead = 0;
        gevimessage = CGeViMessage::ReadBinMessage(
            messageEntry->Buffer,
            messageEntry->Length,
            noOfBytesRead);
        if(gevimessage)
        {
            // You received a message! Now you need to handle it.
            // This can be done here.
        }
        else
    }
}

```

```

        {
            // Message could not be created. Handle the error here.
        }
    }
    else
    {
        // If we are here, we received another type of notification
    }
}

```

3. Register your callback with the connection object.

```

m_APIClient = new GeViAPIClient( ... );
if(m_APIClient)
{
    // connect to the server
    TConnectResult ConnectResult =
        m_APIClient->Connect(ConnectProgressCB, this);
    if(ConnectResult == connectOk)
    {
        // Connection established! Now register your callback!
        m_APIClient->SetCBNotification(
            GeViDatabaseNotificationCB, this);
    }
}

```

Disconnecting from a GeViServer

When disconnecting from the server, you should unregister your notification callback and delete the GeViAPIClient object.

C++ Example:

```

void DisconnectFromServer()
{

```

```
if (m_APIClient != NULL)
{
    // Unregister the notification callback
    m_APIClient->SetCBNotification(NULL, NULL);
    m_APIClient->Disconnect();
    delete m_APIClient;
    m_APIClient = NULL;
}
}
```

State Queries

State Queries are messages sent from the client to the server to get information about the state of logical and physical components of the GeViSoft system well as virtual resources. An example of such information would be an enumeration of all the video inputs available at a GeViServer.

Creating State Queries

You can create a state query by calling its predefined constructor. All the state queries' constructors are located in the *StateQueries* header, C++, and Pascal files.

State queries can then be sent with the `SendStateQuery()` method of the `GeViAPIClient` class. This method returns a `CStateAnswer` object with the GeViServer's response.

```
CStateAnswer* StateAnswer = m_APIClient->SendStateQuery(  
    GetFirstVideoInputQuery, INFINITE);
```

The second parameter of the method is the timeout for a server answer in milliseconds. By sending `INFINITE`, you can prevent the call from timing out.

Creating, sending, and receiving state queries is implemented in the SDK's example Delphi/CPP_SimpleClient.

Enumeration of all video inputs

Pseudo code

1. Create a state query to get the first video input (class `CSQGetFirstVideoInput`)
2. Send the query to the server

3. If the answer is a valid input channel then
4. REPEAT
 - a) Get the actual channel's information from the answer and process it as needed (e.g. print it out, store it to a list)
 - b) Create a state query to get the next video Input (class CSQGetNextVideoInput)
 - c) Send the query
5. UNTIL there is no more video input left

C++ Example:

```
void CMainWin::FillVideoInputsList()
{
    if(m_APIClient == NULL)
        return;

    // Enumerate all available video inputs with the help of state queries.
    // Create a new state query that will return the first video input channel:
    CStateQuery* getFirstVideoInputQuery = new CSQGetFirstVideoInput(
        true, // show only active channels
        true); // show only enabled channels
    if(getFirstVideoInputQuery)
    {
        // Send the query to the server
        CStateAnswer* stateAnswer = m_APIClient->SendStateQuery(
            getFirstVideoInputQuery,
            INFINITE); //Timeout
        // Don't forget to free the memory inside the DLL...
        getFirstVideoInputQuery->DeleteObject();
        if(stateAnswer)
        {
            // Iterate through all available video input channels
```

```

while (stateAnswer->m_AnswerKind != sak_Nothing)
{
    // Get the channels info
    CSAVideoInputInfo* videoInputInfo =
    reinterpret_cast<CSAVideoInputInfo*>(stateAnswer);
    // create a video input descriptor
    TVideoInputDescriptor* newVideoInput = new
        TVideoInputDescriptor(videoInputInfo->m_GlobalID,
            videoInputInfo->m_Name,
            videoInputInfo->m_Description,
            videoInputInfo->m_HasPTZHead,
            videoInputInfo->m_HasVideoSensor,
            videoInputInfo->m_HasContrastDetection,
            videoInputInfo->m_HasSyncDetection);
    // Do something with the channel information. Here:
    // Add the channel information to a
    // CListBox lbVideoInputs
    int newIndex = lbVideoInputs.AddString(
        newVideoInput->m_Name.c_str());
    lbVideoInputs.SetItemDataPtr(newIndex, newVideoInput);
    // Create a query to get the next input channel
    CStateQuery* getNextVideoInputQuery = new
        CSQGetNextVideoInput(true, true,
            videoInputInfo->m_GlobalID);
    stateAnswer->DeleteObject();
    stateAnswer = NULL;
    if(getNextVideoInputQuery)
    {
        stateAnswer =
            m_APIClient->SendStateQuery(
                getNextVideoInputQuery, INFINITE);
        getNextVideoInputQuery->DeleteObject();
        if(!stateAnswer)
            break;
    }
    else //No more video input channel detected!
        break;
}
if(stateAnswer)

```

```
        {
            stateAnswer->DeleteObject();
            stateAnswer = NULL;
        }
    }
}
```

Database Queries (optional)

Database queries allow you to fetch datasets from the action or alarm table of the GeViSoft activity database. All the actions that have been received and all the alarm events that occurred are stored inside the database. To specify and narrow down your query results, several filter operations are available as well.

To get familiar with the possibilities of GeViSoft's database queries, and especially its filtering options, please have a look at the GeViAPI Test Client's "Database Viewer" and "Database Filter" tabs.

Creating Database Queries

You can create a database query by calling its predefined constructor. All the database queries' constructors are located in the DatabaseQueries header, C++, and Pascal files.

Database queries can then be sent with the `SendDatabaseQuery()` method of the GeViAPIClient class. This method returns a `CDataBaseAnswer` object with the GeViServer's response.

```
CDataBaseQuery* geviquery = new CDBQCreateActionQuery(0);  
  
CDataBaseAnswer* dbAnswer = m_APIClient->SendDatabaseQuery(geviquery, INFINITE);
```

The second parameter of the method is the timeout for a server answer in milliseconds. By sending `INFINITE`, you can prevent the call from timing out.

Database Query Session Handling

Action sending and state querying did not need any form of session handling. This is different for database querying. Usually you want to collect several records that are connected in

some form, e.g. applying the same filter set to subsequent queries. To signal the database engine that your queries are associated, you pass a unique query handle with them. The query handle is the result you receive from a `CDBQCreateActionQuery` or `CDBQCreateAlarmQuery`. Therefore these queries are the first you send when interacting with the database.

C++ Example for getting a query handle:

```
// Create a new Action Query
CDataBaseQuery* geviquery = new CDBQCreateActionQuery(0);

// Send the Action Query to the server
CDataBaseAnswer* dbanswer = m_APIClient->SendDatabaseQuery(geviquery, INFINITE);

                                geviquery->DeleteObject();

if (dbanswer->m_AnswerCode == dbac_QueryHandle)
{
    // Extract the query handle from the answer
    CDBAQueryHandle* handle = reinterpret_cast<CDBAQueryHandle*>(dbanswer);
}
```

Iterating over Database Records

You can send a group of associated database queries after having obtained the query handle. Please note that the GeViSoft architecture always returns one single answer for every query. As a consequence, you might need to issue several database queries consecutively to get your desired pieces of information.

This can be illustrated by an example database query. Imagine you want to retrieve the two latest actions inside the database:

Example: Retrieving of the two latest actions inside the database

Pseudo code

1. Create a new CDBQCreateActionQuery
2. Send the query to GeViServer and retrieve the handle from the answer
3. Create a new CDBQGetLast query with the handle as the argument
4. Send the query and fetch the latest action as an answer
5. Extract the latest action's primary key from the answer
6. Create a new CDBQGetPrev query with the handle and the latest action's primary key as an argument
7. Send the query and fetch the second latest action as an answer

C++:

```

// Declare a query handle
CDBAQueryHandle* handle;
__int64 primaryKey;

// Create a new Action Query
CDataBaseQuery* geviquery = new CDBQCreateActionQuery(0);

// Send the Action Query to the server
CDataBaseAnswer* dbanswer = m_APIClient->SendDatabaseQuery(geviquery, INFI-
NITE);
geviquery->DeleteObject();

if (dbanswer->m_AnswerCode == dbac_QueryHandle)
{
    // Extract the query handle from the answer
    handle = reinterpret_cast<CDBAQueryHandle*>(dbanswer);
}

// Create a database query for the latest action entry
CDataBaseQuery* getEntry = new CDBQGetLast(handle->m_Handle);

// Send the query to the GeViServer
dbanswer = m_APIClient->SendDatabaseQuery(getEntry, INFINITE);
getEntry->DeleteObject();

// Check if an action entry is in the database
if (dbanswer->m_AnswerCode == dbac_ActionEntry)
{
    // Do s.th. with the answer here...
    // Get the primary key which is used to
    // address the records internally
    primaryKey = reinterpret_cast<CDBAActionEntry*>(dbanswer->m_PK);
} //TODO: Add error handling if no action is in the database

//Create a database query to get the second latest action entry
getEntry = new CDBQGetPrev(handle->m_Handle, primaryKey);
// Send the query to the GeViServer
dbanswer = m_APIClient->SendDatabaseQuery(getEntry, INFINITE);
getEntry->DeleteObject();

```

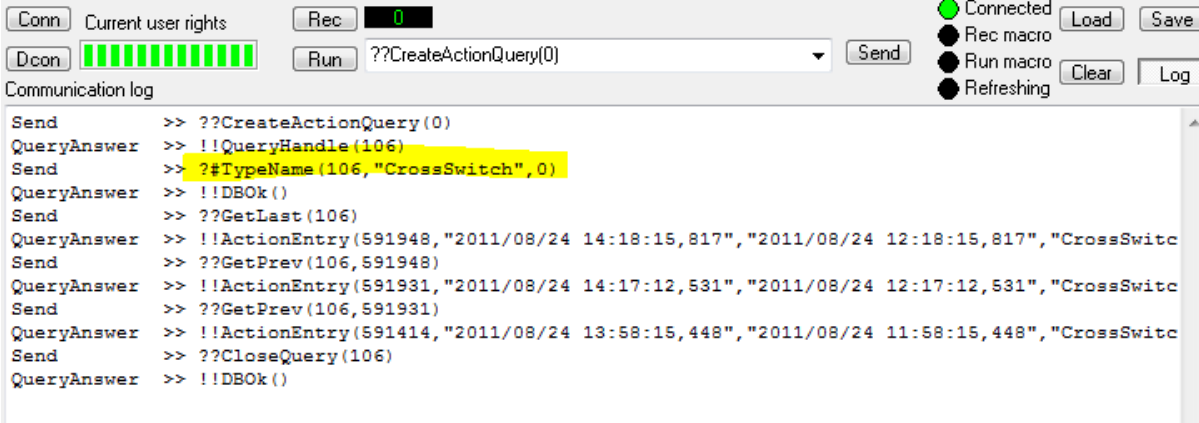
```
// Check if an action entry is in the database
if (dbanswer->m_AnswerCode == dbac_ActionEntry)
{
    // Do s.th. with the answer here...
} //TODO: Add error handling if no action is in the database
dbanswer->DeleteObject();
```

Filtering Database Queries

GeViSoft supports various filters allowing you to specify your queries in a more precise way. For example, you can narrow down your search to certain action types or senders. All the available filters are declared in the DatabaseQueries header file.

To set the filtering on the GeViServer, you have to send a database query for every filter element after you have obtained the query handle. You can monitor the processing of the queries inside the GeViAPI Test Client.

Here is a screenshot of a database query sequence which sets a filter for the action type name *CrossSwitch*. The message setting the filter is highlighted. The filter has been defined in the *Database Filter* tab of the GeViAPI Test Client. Afterwards, the fetch operation was started from the *Database Viewer* tab.



The screenshot shows the GeViAPI Test Client interface. At the top, there are buttons for 'Conn', 'Dcon', 'Rec', 'Run', 'Send', 'Load', 'Save', 'Clear', and 'Log'. The 'Rec' button is highlighted with a red background. Below the buttons, there is a 'Communication log' section. The log displays a sequence of messages and responses:

```
Send >> ??CreateActionQuery(0)
QueryAnswer >> !!QueryHandle(106)
Send >> ??TypeName(106, "CrossSwitch", 0)
QueryAnswer >> !!DBOk()
Send >> ??GetLast(106)
QueryAnswer >> !!ActionEntry(591948, "2011/08/24 14:18:15,817", "2011/08/24 12:18:15,817", "CrossSwitc
Send >> ??GetPrev(106, 591948)
QueryAnswer >> !!ActionEntry(591931, "2011/08/24 14:17:12,531", "2011/08/24 12:17:12,531", "CrossSwitc
Send >> ??GetPrev(106, 591931)
QueryAnswer >> !!ActionEntry(591414, "2011/08/24 13:58:15,448", "2011/08/24 11:58:15,448", "CrossSwitc
Send >> ??CloseQuery(106)
QueryAnswer >> !!DBOk()
```

Composing Filtered Queries

In this paragraph you will learn how to compose simple filters first and finally extent the exam-
ple from above ([Iterating over Database Records](#)) with a filter that will only return

`CustomAction` messages with certain primary keys.

Prerequisite for a test on your system is that there are `CrossSwitch`, `CustomAction`, and several other action type entries stored inside your database. To populate your database with these, you can send them with the GeViAPI Test Client. Doing a fetch in the *Database Viewer*'s tab allows you to verify that they are stored correctly afterwards.

Example Filters

Example for a filter that will only return `CustomActions`:

```
CDataBaseFilter* myActionNameFilter =  
    new CDBFTypeName(handle->m_Handle, "CustomAction", dbc_LIKE);
```

After creating your filters, you can send them with GeViAPIClients `SendDatabaseQuery` method.

```
CDataBaseAnswer* dbanswer =  
    m_APIClient->SendDatabaseQuery(myActionNameFilter, INFINITE);
```

Make sure to verify that the answer code is `dbac_DBOk` and to call the `DeleteObject` method for your filter after sending it.

Composing complex filters:

You can compose a complex filter by sending a sequence of multiple single filters to the database. These filters will then be treated as a conjunction (logical AND) by GeViServer.

Here is an example for a complex filter that only returns actions with primary keys between 500 and 600. This filter has to be composed by sending two simple filters sequentially:

```
CDataBaseFilter* myMinFilter =  
    new CDBFPK_GrtEqu(handle->m_Handle, 500);
```

```
CDataBaseFilter* myMaxFilter =
    new CDBFPK_LowEqu(handle->m_Handle, 600);
```

Complete Example of a Filtered Database Query

The example [Iterating over Database Records](#) will be extended to filter for `CustomActions` with a primary key between 500 and 600 in this paragraph. To achieve this, the filter messages have to be sent directly after retrieving the database handle. The filters are created in a method called `setActionFilter`. This message is then called after obtaining the database handle.

Example filtering method:

```
void setActionFilter(CDBAQueryHandle* handle)
{
    // Create a vector with your filter messages
    std::vector<CDataBaseFilter*> filterList;
    filterList.push_back( new CDBFPK_GrtEqu(handle->m_Handle, 500) );
    filterList.push_back( new CDBFPK_LowEqu(handle->m_Handle, 600) );
    filterList.push_back( new CDBFTypename(handle->m_Handle,
                                           "CustomAction", dbc_LIKE) );

    //Send the filters
    for (vector<CDataBaseFilter*>::iterator it =
        filterList.begin(); it != filterList.end();
        it++)
    {
        CDataBaseAnswer* dbanswer = m_APIClient->SendDatabaseQuery(*it, INFI-
        NITE);
        if (dbanswer->m_AnswerCode != dbac_DBOk)
        {
            // Do error handling here!
            (*it)->DeleteObject();
            return;
        }
    }
}
```

```

        (*it)->DeleteObject();
    }
}

```

Now you can call that method in your example from above:

```

/*
...
*/
CDataBaseAnswer* dbanswer = m_APIClient->SendDatabaseQuery(
    geviquery, INFINITE);
geviquery->DeleteObject();

if (dbanswer->m_AnswerCode == dbac_QueryHandle)
{
    // Extract the query handle from the answer
    handle = reinterpret_cast<CDBAQueryHandle*>(dbanswer);

    //Send Filter here
    setActionFilter(handle);
}
/*
...
*/

```

As a result, you should see the two latest `CustomAction` records with a primary key between 500 and 600. If you do not get any results, you need to adopt the filtering criteria to match records in your database.

Database queries and filtering is implemented in the SDK's example Delphi/CPP_ SimpleDatabaseClient.

C# and .Net specifics

This chapter deals with the GeViSoft SDKs .Net capabilities and specifics. It describes the architecture of the wrappers and the specifics of the usage.

Architecture

The GeViSoft SDK is delivered with a .Net-Wrapper, allowing you to design applications in C# or other .Net languages. The Geutebrueck.GeViSoftSDKNET.Wrapper.dll encapsulates all the native GeViProcAPI.dll calls. Additionally, the GscActionsNet.dll from the GeViScopeSDK is needed to allow for GeViScope interoperability. This wrapper encapsulates the GscActions.dll which itself uses the GscDBI.dll.

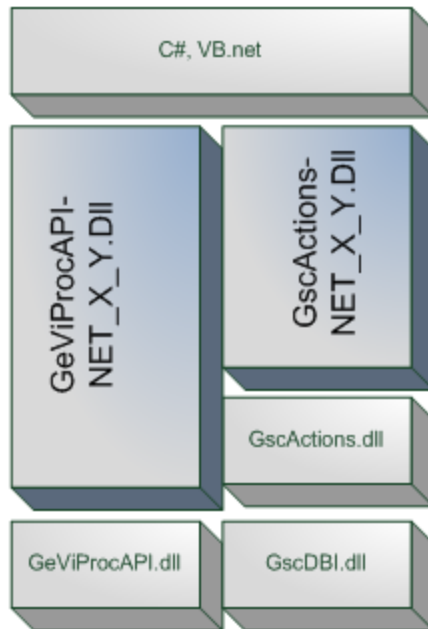


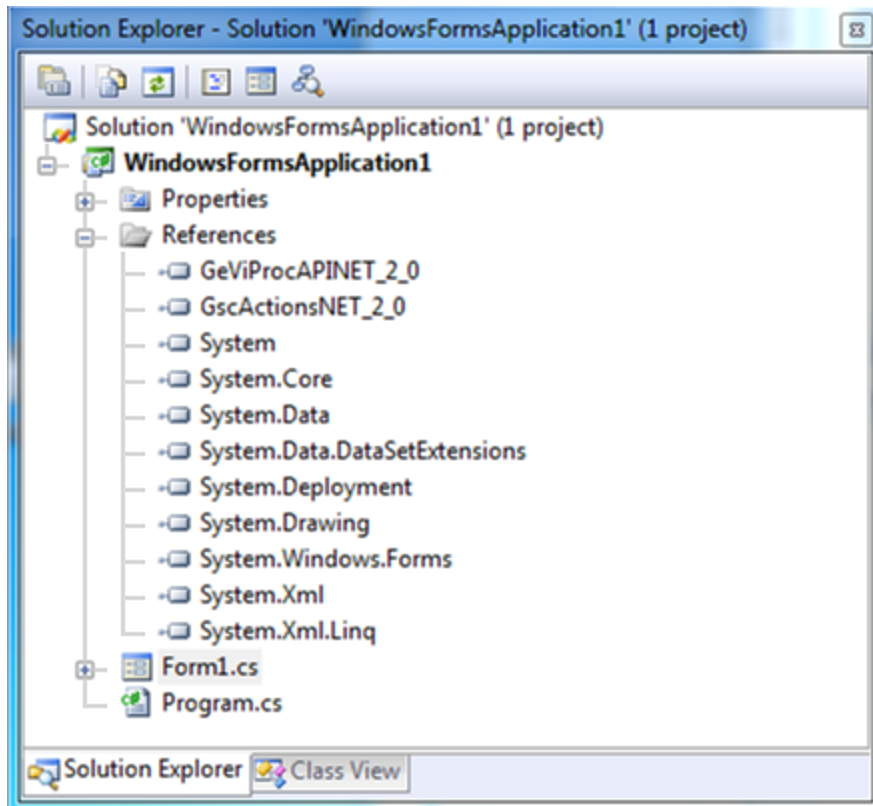
Diagram of the GeViSoft .Net wrappers

Configuring your IDE for GeViSoft .Net Projects

Visual Studio 2008, C#

1.) Add the .Net wrappers to your project's references.

(You can do this by right-clicking on *References* in your *Solution Explorer*. After pressing *Add Reference* browse to your %GEVISOFTSDKPATH% and add GeViProcAPINET_2_0.dll. If you plan to use GeViScope Actions also add GscActionsNET_2_0.dll.



2.) Change the platform settings to generate X86 code if it is not already set.
In the Configuration Manager select *Platform* -> *New* -> *X86* and use this platform for the Debug and Release configurations.

Application

Build

Build Events

Debug

Resources

Services

Settings

Reference Paths

Signing

Security

Publish

Code Analysis

Configuration: Active (Debug) Platform: Active (x86)

General

Conditional compilation symbols:

☒ Define DEBUG constant

☒ Define TRACE constant

Platform target: x86

☐ Allow unsafe code

☐ Optimize code

Errors and warnings

Warning level: 4

Suppress warnings:

Treat warnings as errors

☒ None

☐ Specific warnings:

☐ All

Output

Output path: c:\gevisoft\

4.) Add the required using directives to your project's source files.
For a project that only uses GeViSoft actions, you need at least.

```
GEUTEBRUECK.GeViSoftSDKNET.ActionsWrapper
```

as well as

```
GEUTEBRUECK.GeViSoftSDKNET.ActionsWrapper.ActionDispatcher
```

and, additionally for actions from every action class you use,

```
GEUTEBRUECK.GeViSoftSDKNET.ActionsWrapper.YourActionClass
```

If you also want to use GeViScope actions, make sure to include

```
GEUTEBRUECK.GeViScope.Wrapper.Actions.ActionDispatcher
```

and

```
GEUTEBRUECK.GeViScope.Wrapper.Actions.YourActionClass
```

You can find descriptions of the actions and their respective action classes in the API documentation or by inspecting the assemblies with the Object Browser.

Visual Studio 2010, C#

Configure your project as described in paragraph [Visual Studio 2008, C#](#)

Common Tasks with C#

This chapter describes several common tasks you might need to carry out during your development. The tasks are described in pseudo code and C#.

Connecting to a GeViServer

This paragraph shows you what tasks are needed for connecting to a GeViServer.

Connecting

Pseudo Code

1. Implement the connect callback method
2. Create an instance of a database connection object
3. Call the create() method of your database connection object
4. Add your callback delegate method to the invocation list
5. Register your callback method
6. Call the connect method of your database connection object

C#

```
// This is the connect progress callback method.
// It is called from within GeViSoft during the connection progress
void myConnectProgress(object sender, GeViSoftConnectProgressEventArgs e)
{
    Console.WriteLine("Connecting... {0} of {1}", e.Progress, e.Effort);
}

// myDB is the database object that encapsulates
// all GeViSoft interaction.
GeViDatabase myDB = new GeViDatabase();

// Set the server name, user name and password of your
// GeViSoft connection
myDb.Create("localhost", "sysadmin", "masterkey");
```

```
// Add your callback delegate to the invocation list
myDb.ConnectProgress += new GeViSoftConnectProgressEventHandler(
    myConnectProgress);

// Register the callback inside GeViSoft
myDb.RegisterCallback();

// Now you can connect to the GeViSoft Server...
myDB.Connect();
```

A straightforward implementation for establishing a connection can be found in example `cs_ConsoleClient`.

Message Handling

After having established the connection, you are ready to exchange messages and actions with the server.

Creating and Sending of GeViSoft Messages

There are two approaches that can be taken to create and send GeViSoft messages. You can either create a message instance by calling its constructor and then send this instance, or you can directly send a string representation of a message without instantiating it first.

Example 1 – Creating an instance of a message and sending it afterwards

```
// Create a CrossSwitch Action and switch
// input 7 to output 1
GeViAct_CrossSwitch myAction = new GeViAct_CrossSwitch(
    7, 1, GeViTSwitchMode.sm_Normal);

// Send the action
```

```
myDB.SendMessage(myAction);
```

NOTICE

Make sure you have included your action's corresponding action class namespace in your using directives. See [Configuring your IDE for GeViSoft.Net Projects -> VS2008, C#](#)

Example 2 – Directly sending a message from a string

```
myDB.SendMessage("CrossSwitch(7,1,0)");
```

Receiving of GeViSoft Actions

GeViSoft action dispatching is event based. Internally, for every action that is received, an event is fired. If you want to process certain GeViSoft action messages inside your application, you can register an event handler for this particular action. The event handler is called whenever a new action of that type is received.

If you want to process CrossSwitch actions in your application, you can proceed as shown in this example.

Pseudo code:

1. Implement a method that is called whenever the event is fired (action received)
2. Register your method as an event handler for the particular action
3. Register your event handler inside the GeViSoft connection object.

C#:

```
// Method to be called on receiving a CrossSwitch Action
void myDB_ReceivedCrossSwitch(object sender, GeViAct_CrossSwitchEventArgs
e)
{
```

```

        String receivedMessage = "CrossSwitch(" +
            e.aVideoInput + "," +
            e.aVideoOutput + "," +
            e.aSwitchMode + ")";
    }

    // Event handler for CrossSwitch Actions
    myDB.ReceivedCrossSwitch += new
        GeViAct_CrossSwitchEventHandler(myDB_ReceivedCrossSwitch);

    // Don't forget to register the handler inside the GeViSoft connection
    object
    myDB.RegisterCallback();

```

Receiving of GeViSoft Notifications

Besides actions, GeViSoft also sends messages regarding the server status, the database notifications. You can receive these notifications by registering a GeViSoftDatabaseNotificationEventHandler. The procedure is similar to the action subscription as described above.

Here is an example:

```

void myDB_DatabaseNotification(object sender,
                                GeViSoftDatabaseNotificationEventArgs e)
{
    switch (e.ServerNotificationType)
    {
        case GeViServerNotification.NFServer_Disconnected:
            // ("Disconnected from Server");
            break;
        case GeViServerNotification.NFServer_GoingShutdown:
            // ("Server is shutting down");
            break;
    }
}

```

```

        case GeViServerNotification.NFServer_SetupModified:
            // ("Server setup has been modified");
            break;
        case GeViServerNotification.NFServer_NewMessage:
            // An "ordinary" action has been received.
            // Handle this action in a separate Event-
            // Handler
            // (like myDB_ReceivedCrossSwitchAction)
            break;
    }

    // You register the handler as described before
    myDB.DatabaseNotification += new
        GeViSoftDatabaseNotificationEventHandler(myDB_DatabaseNotification);
    myDB.RegisterCallback();

```

NOTICE

Please note that the `e.ServerNotificationType` equals ***GeViServerNotification.NFServer_NewMessage*** with every GeViSoft action that is received, regardless if you already subscribed for it with another event handler.

Handling of GeViScope Actions

You can also send GeViScope actions from your GeViSoft application. Sending GeViScope actions is very similar to sending GeViSoft actions. The only difference is that you need to add the GeViScope server alias to the `SendMessage()` method's parameters.

Sending GeViScope Actions

Example – sending a GeViScope message:

Prerequisite:

1. Connection to GeViScope has been configured with GeViSet
2. The appropriate namespaces are included

```
// Example for GeViScope namespace needed to handle
// a GeViScope Custom Action
using GEUTEBRUECK.GeViScope.Wrapper.Actions;
using GEUTEBRUECK.GeViScope.Wrapper.Actions.SystemActions;

// Create the GeViScope action
GscAct_CustomAction myGscAction = new
    GscAct_CustomAction(23, "Hello GeViScope!");

// Send the Action to the "GeViScope_Alias" server
myDB.SendMessage("GEVISCOPE_ALIAS", myGscAction);
```

Receiving GeViScope Actions

Receiving GeViScope actions is a little different from handling GeViSoft actions. Due to architectural constraints, whenever a GeViSoft Action arrives, it is encapsulated into a special GeViSoft action. This action is called GscAction.

The GscAction itself is retrieved like any other GeViSoft action by implementing a method that processes the GscAction and that is added as an event handler for receiving the GscAction.

This method receives the original GeViScope Action embedded into its EventArgs whenever it is called.

Normally you then would have to identify and parse the GeViScope action and handle it as needed by hand. For your convenience, the Geutebrueck SDKs provide you with a dispatcher that can simplify that task:

There is a Dispatcher method for GeViScope actions that works very similar to the dispatching mechanism used by GeViSoft. You can simply add event handlers for the GeViScope actions you are interested in and process them in there.

Example – Receiving and Dispatching GeViScope Actions inside GeViSoft:

Pseudo Code:

1. Create an instance of the GscActionDispatcher class that will dispatch the GeViScope actions to your handlers
2. Create an event handler that receives the GeViSoft GscAction. Inside this event handler, call the dispatch method of your GscActionDispatcher instance for every received GscAction.
3. Register the GeViSoft GscAction event handler with your GeViSoft database connection object.
4. Create an event handler method for any GeViScope action you want to process
5. Register your GeViScope actions event handler at the dispatcher.

C#:

```
// Create an instance of the GscActionDispatcher class
GscActionDispatcher myGscDispatcher = new GscActionDispatcher();

// GscAction event handler that dispatches the GscAction
void myDB_ReceivedGscAction(object sender, GeViAct_GscActionEventArgs e)
{
    myGscDispatcher.Dispatch(e.m_GscAction);
}

// Add the handler for GscAction (this is called for any newly received GeViScope action)
myDB.ReceivedGscAction += new
    GeViAct_GscActionEventHandler(myDB_ReceivedGscAction);

// Don't forget to register the callbacks!
myDB.RegisterCallback();

// Event handler method for the GeViScope Action
```

```

void myGscDispatcher_OnCustomAction(object sender, GscAct_CustomActionEventArgs e)
{
    Console.WriteLine "Received GEVISCOPe CustomAction("+ e.aInt + ", "
        + e.aString + ")";
}

// Register the GeViScope CustomAction event handler with the dispatcher
myGscDispatcher.OnCustomAction += new
GscAct_CustomActionEventHandler(myGscDispatcher_OnCustomAction);

```

NOTICE

You can find a complete example application that sends and receives GeViScope actions in CS_SimpleGscActionClient.

State Queries in C#

This paragraph describes how you can send and receive State Queries from within C#. For an introduction to State Queries in general see chapter [SDK-Usage->State Queries](#).

Creating and Sending State Queries

You can create State Queries with their respective constructors and send them afterwards by calling the `SendQuery()` method of your database connection instance. The `SendQuery()` method returns the GeViSoft State Answer via an out parameter.

```

// myAnswer is filled by the SendQuery() method
// with the State Answer.
GeViMessage myAnswer;
// This is your query
GeViMessage myQuery = new GeViSQ_GetFirstVideoInput(true, true);

```

```
// Send the query
myDB.SendQuery(myQuery, out myAnswer);
if (myAnswer is GeViSA_VideoInputInfo)
{
    // Do something with my answer here...
}
```

Setting the State Query Timeout

The method *SendQuery()* blocks until the database answer is retrieved from the GeViServer. If the server does not answer, this leads to a deadlock. A maximum timeout timer for the *SendQuery* exists to prevent waiting endlessly for a database answer. By default, the timeout is set to 3000 ms. You can change this timeout globally by calling the *SetQueryTimeoutInMs()* method of your database connection instance.

Example – Setting the SendQuery timeout to one second:

```
myDB.SetQueryTimeoutInMs(1000);
```

Enumeration of all video inputs

Pseudo code

1. Create a state query to get the first video input (class GeViSQ_GetFirstVideoInput)
2. Send the query to the server
3. If the answer is a valid input channel then
4. REPEAT
 - a) Get the actual channel's information from the answer and process it as needed (e.g. print it out, store it to a list)
 - b) Create a state query to get the next video Input (class GeViSQ_

GetNextVideoInput)
c) Send the query

5. UNTIL there is no more video input left

C# Example:

```
private List<GeViSA_VideoInputInfo> getVideoInputsList()
{
    List<GeViSA_VideoInputInfo> myVideoInputs =
        new List<GeViSA_VideoInputInfo>(0);
    if (myDB != null)
    {
        GeViMessage myAnswer;
        myDB.SendQuery(new GeViSQ_GetFirstVideoInput(true, true),
            out myAnswer);
        while (myAnswer is GeViSA_VideoInputInfo)
        {
            int tempID = (myAnswer as GeViSA_VideoInputInfo).sGlobalID;
            myVideoInputs.Add(myAnswer as GeViSA_VideoInputInfo);
            myDB.SendQuery(
                new GeViSQ_GetNextVideoInput(true, true, tempID),
                out myAnswer);
        }
    }
    return myVideoInputs;
}
```

NOTICE

You can find a complete example application that sends State Queries and receives State Actions in CS_SimpleClient. This example shows you how to enumerate video in- and outputs as well as digital IO.

Supported Development Platforms

The SDK is designed for and tested to work with the following development environments:

- Microsoft Visual Studio 2008, C++
- Microsoft Visual Studio 2010, C++
- Microsoft Visual Studio 2008, C#
- Microsoft Visual Studio 2010, C#
- Embarcadero RAD Studio XE, Delphi

Examples

The SDK is shipped with various examples showing you how to implement common tasks. The examples are grouped by functionality and platform.

By Functionality

Connecting/disconnecting to GeViServer

- CPP_SimpleActionClient (VS2008/VS2010, C++)
- CPP_SimpleClient (VS2008/VS2010, C++)
- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CPP_MonitoredConnectionClient (VS2008/VS2010, C++)
- CPP_ConsoleClient (VS2008/VS2010, C++)
- CS_SimpleActionClient (VS2008/VS2010, C#)
- CS_SimpleClient (VS2008/VS2010, C#)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- CS_SimpleGscActionClient (VS2008/VS2010, C#)
- Delphi_SimpleActionClient (RAD Studio XE)
- Delphi_SimpleClient (RAD Studio XE)
- Delphi_SimpleDatabaseClient (RAD Studio XE)
- Delphi_ConsoleClient (RAD Studio XE)

Monitoring a GeViSoft connection

- CPP_MonitoredConnectionClient (VS2008/VS2010, C++)

Automatically reconnecting a GeViSoft connection on loss

- CPP_MonitoredConnectionClient (VS2008/VS2010, C++)

Sending/receiving of GeViSoft actions/messages

- CPP_SimpleActionClient (VS2008/VS2010, C++)
- CPP_MonitoredConnectionClient (VS2008/VS2010, C++)
- CPP_SimpleClient (VS2008/VS2010, C++)

- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CPP_ConsoleClient (VS2008/VS2010, C++)
- CS_SimpleActionClient (VS2008/VS2010, C#)
- CS_SimpleClient (VS2008/VS2010, C#)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- CS_ConsoleClient (VS2008/VS2010, C#)
- Delphi_SimpleActionClient (RAD Studio XE)
- Delphi_SimpleClient (RAD Studio XE)
- Delphi_SimpleDatabaseClient (RAD Studio XE)
- Delphi_ConsoleClient(RAD Studio XE)

Sending/receiving of GeViScope actions/messages

- CS_SimpleGscActionClient (VS2008/VS2010, C#)

Receiving and dispatching of server notifications

- CPP_SimpleActionClient (VS2008/VS2010, C++)
- CPP_MonitoredConnectionClient (VS2008/VS2010, C++)
- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CPP_ConsoleClient (VS2008/VS2010, C++)
- CS_SimpleActionClient (VS2008/VS2010, C#)
- CS_SimpleClient (VS2008/VS2010, C#)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- CS_SimpleGscActionClient (VS2008/VS2010, C#)
- CS_ConsoleClient (VS2008/VS2010, C#)
- Delphi_SimpleActionClient (RAD Studio XE)
- Delphi_SimpleClient (RAD Studio XE)

Converting between ASCII and binary representation of messages

- CPP_SimpleActionClient (VS2008/VS2010, C++)
- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CPP_ConsoleClient (VS2008/VS2010, C++)
- Delphi_SimpleActionClient (RAD Studio XE)
- Delphi_SimpleDatabaseClient (RAD Studio XE)
- Delphi_ConsoleClient(RAD Studio XE)

Sending/receiving state queries and answers

- CPP_SimpleClient (VS2008/VS2010, C++)
- CS_SimpleGscActionClient (VS2008/VS2010, C#)
- Delphi_SimpleClient (RAD Studio XE)

Enumeration of video input and output channels

- CPP_SimpleClient (VS2008/VS2010, C++)
- CS_SimpleClient (VS2008/VS2010, C#)
- Delphi_SimpleClient (RAD Studio XE)

Enumeration of digital IO contacts

- CPP_SimpleClient (VS 2008/VS2010, C++)
- CS_SimpleClient (VS2008/VS2010, C#)
- Delphi_SimpleClient (RAD Studio XE)

Sending database action queries

- CPP_SimpleDatabaseClient (VS 2008/VS2010, C++)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- Delphi_SimpleDatabaseClient (RAD Studio XE)

Sending database alarm queries

- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- Delphi_SimpleDatabaseClient (RAD Studio XE)

Navigating through database entries

- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- CS_SimpleDatabaseClient (VS2008/VS2010, C#)
- Delphi_SimpleDatabaseClient (RAD Studio XE)

Converting database answers from binary to ASCII representation

- CPP_SimpleDatabaseClient (VS2008/VS2010, C++)
- Delphi_SimpleDatabaseClient (RAD Studio XE)

Building GeViSoft messages from user input

- CPP_ConsoleClient (VS2008/VS2010, C++)
- CS_ConsoleClient (VS2008/VS2010, C#)
- Delphi_ConsoleClient(RAD Studio XE)

By Platform

Microsoft Visual Studio 2008/2010, C++, MFC

CPP_SimpleActionClient

- Connecting/disconnecting to GeViServer
- Sending/receiving of actions
- Receiving and dispatching of server notifications
- Converting between ASCII and binary representation of messages.

CPP_MonitoredConnectionClient

- Connecting/disconnecting to GeViServer
- Sending/receiving of actions
- Receiving and dispatching of server notifications
- Converting between ASCII and binary representation of messages
- Monitoring a GeViSoft connection
- Automatically reconnecting a GeViSoft connection on loss

CPP_SimpleClient

- Connecting/disconnecting to GeViServer
- Sending/receiving state queries and answers
- Sending/receiving of actions
- Enumeration of video input and output channels
- Enumeration of digital IO contacts
- Receiving and dispatching of server notifications

CPP_SimpleDatabaseClient

- Connecting/disconnecting to GeViServer
- Sending database action queries
- Sending database alarm queries
- Navigating through database entries
- Receiving database entries
- Converting database answers from binary to ASCII representation

Microsoft Visual Studio 2008/2010, C++, Console

CPP_ConsoleClient

- Connecting/disconnecting to GeViServer
- Building GeViSoft messages from user input
- Receiving and displaying messages
- Converting between ASCII and binary representation of messages

Microsoft Visual Studio 2008/2010, C# WinForms

CS_SimpleActionClient

- Connecting/disconnecting to GeViServer
- Sending/receiving of actions
- Receiving and dispatching of server notifications

CS_SimpleClient

- Connecting/disconnecting to GeViServer
- Sending/receiving state queries and answers
- Sending/receiving of actions
- Enumeration of video input and output channels
- Enumeration of digital IO contacts
- Receiving and dispatching of server notifications

CS_SimpleDatabaseClient

- Connecting/disconnecting to GeViServer
- Sending database action queries
- Sending database alarm queries
- Navigating through database entries
- Receiving database entries

CS_SimpleGscActionClient

- Connecting/disconnecting to GeViServer
- Creating/Dispatching GeViScope Actions

- Creating GeViScope Action Event Handlers
- Adding/Removing Event Handlers on runtime

Microsoft Visual Studio 2008/2010, C#, Console

CS_ConsoleClient

- Connecting/disconnecting to GeViServer
- Building GeViSoft messages from user input
- Receiving and displaying messages
- Composing GeViSoft actions from strings

Embarcadero RAD Studio XE, Delphi, VCL

Delphi_SimpleActionClient

- Connecting/disconnecting to GeViServer
- Sending/receiving of actions
- Receiving and dispatching of server notifications
- Converting between ASCII and binary representation of messages.

Delphi_SimpleClient

- Connecting/disconnecting to GeViServer
- Sending/receiving state queries and answers
- Sending/receiving of actions
- Enumeration of video input and output channels
- Enumeration of digital IO contacts
- Usage of video input and output descriptors
- Usage of digital contact input and output descriptors
- Receiving and dispatching of server notifications

Delphi_SimpleDatabaseClient

- Connecting/disconnecting to GeViServer
- Sending database action queries
- Sending database alarm queries

- Navigating through database entries
- Receiving database entries
- Converting database answers from binary to ASCII representation

Embarcadero RAD Studio XE, Delphi, Console

Delphi_ConsoleClient

- Connecting/disconnecting to GeViServer
- Building GeViSoft messages from user input
- Receiving and displaying messages
- Converting between ASCII and binary representation of messages