QuaeroSys
Medical Devices

Piezostimulator

User Manual

30.10.2010

QuaeroSys Medical Devices

Am Hohenwiesenweg 5
D-63679 Schotten

Tel.: +49 (0) 60 44 / 9 66 98 - 0
Fax.: +49 (0) 60 44 / 9 66 98 - 100
www.quaerosys.com
medicaldevices@quaerosys.com

Contents

1 Overview

The QS Piezostimulator is a mechanical tactile stimulator, specified for use in functional imaging experiments like MEG, fMRI or EEG. The stimulation is realized by using several pin matrices and other devices, which are able to apply mechanical forces.

The system consists of a master device which is able to communicate with a personal computer via USB and several stimualtion modules which are slaves of this master device. The master device has separate ports for input and output trigger signals, enabling a precise synchronization between stimulation and measurement. For more complex experiments, it is even possible to upgrade the system with up to 16 cards for stimulation or trigger input or output purposes. The timing accuracy is at 0.5 ms.

The intended purpose is the mechanical tactile stimulation of the human body for investigations of physiological processes using functional imaging methods like MEG, fMRI or EEG without making clinical diagnostics.

2 Hardware

2.1 Technical data

Power supply

Voltage	85-230 V AC
Frequency	50-60 Hz
Power	40 W

Trigger ports

Voltage	5 V DC
Trigger output (HW ver. <1.0)	
Type	Positive pulse (logical ’1’)
Length	500 ns
Trigger output (HW ver. ≥1.0)	
Type	Positive or negative pulse (logical ’1’ or ’0’)
Length	Multiples of 500 µs
Trigger input	Rising edge
Galvanic isolation	Optical

Physical size (rack version)

Height	5.25 inch = 3 RU (133 mm)
Width	19 inch (483 mm)
Depth (incl. handle bars)	10.75 inch (273 mm)
Depth (rack)	9.5 inch (243 mm)

Physical size (standalone version, without handle)

Height	7.25 inch (185 mm)
Width	13.75 inch (350 mm)
Depth	21.25 inch (540 mm)

Supported operating systems

Linux	Dynamic Shared Object (DSO)
MS Windows XP®	Dynamic Link Library (DLL)
MS Windows 2000®	Extension for Presentation (Neurobehavioral Systems)
MS DOS ®	Not supported

Connection to the stimulation computer

Connector	USB-B
Specification	USB 2.0
Speed	Full speed 12 Mbit/s
Galvanic isolation	Optical

2.2 Controller card

The controller card (figure 2.1) is the "heart" of the piezostimulator. It controls the stimulation sequence which is received via USB.

For triggering purposes there are two ports (input/output). By using the `triggerOut` command a trigger can be emitted on port 0x01, when execution of the stimulation sequence reaches this command.

The `waitForTrigger` command stops the running stimulation sequence when it is executed and waits until an incoming pulse (rising edge) on the trigger input (port 0x00) starts the sequence again.

In hardware version <1.0 trigger output is always a positive pulse (logical '1') with a width of 500 ns. In hardware version version ≥1.0 the pulse is configurable. It can be positive (logical

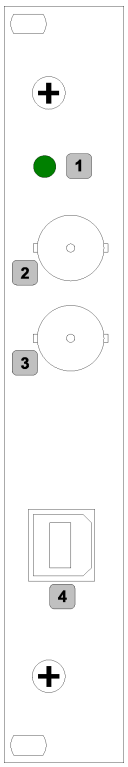

Figure 2.1: Controller card (front view). - **1** green status LED (card), **2** trigger input port (port address 0), **3** trigger output port (port address 1), **4** USB 2.0 interface.

'1') or negative (logical '0') with a variable length in multiples of 500 µs. To modify these configuration parameters the commands `setTriggerMode` and `setTriggerLength` are used.

A firmware is running on the controller card which interprets the stimulation sequences and manages communication with the host PC. From time to time there are software updates available which can be obtained from our homepage and installed by the user. Furthermore the hardware version of the controller card determines some functional aspects e. g. the trigger output. If there is an upgrade available the card has to be send in for upgrading. Please contact us, if you wish to upgrade.

The slot address of the controller is always 0x10 (hex) or 16 (decimal).

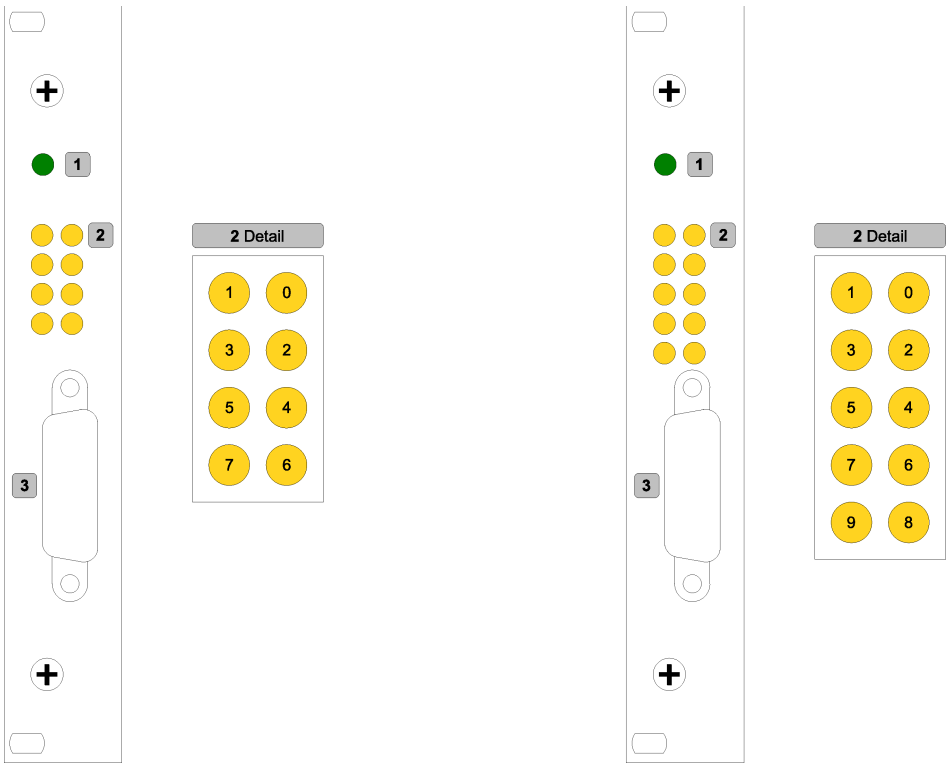

Figure 2.2: Stimulation cards with 8 and 10 pins (front view). - **1** green status LED (card), **2** yellow status LEDs (one for each pin), **3** SUB-D connector (interface to stimulation module).

2.3 Stimulation cards

The stimulation card (figure 2.2) controls the pins of a stimulation module. There are two versions available which can drive 8 or 10 pins. If one module has more pins than a single card can handle, 2 or more stimulation cards are used. The stimulation module is connected via a SUB-D connector and pin activity is visualized by yellow LEDs which belong to one pin each.

Possible slot addresses of stimulation cards range from 0x00 (hex) to 0x0F (hex) or 0 to 15 (decimal).

2.4 Stimulation modules with pin matrices

There are many stimulation devices with different pin matrices. Each of the pins can be driven out to 4096 different heights between 0 and 1.5 mm with a timing accuracy of 0.5 ms.

All matrices are named with AxB where A is the number of pins in X-direction and B is the number of pins in Y-direction. Matrices are available in normal (non-shifted) and shifted versions. In the non-shifted pin array the distance between two pins is 2.5 mm from center to center and 1.5 mm from edge to edge as the pins are 1.0 mm in diameter. In the shifted version distances are the same, but one over another column is shifted in Y-direction by 1.25 mm (see figure 2.3).

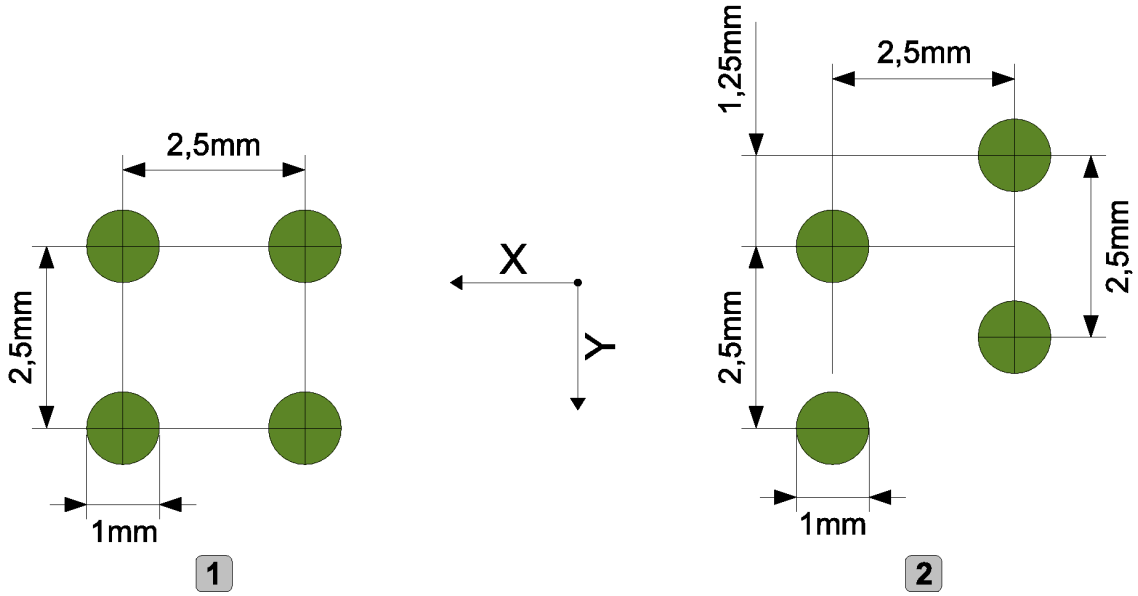

Figure 2.3: Measurements of matrices shown on a representative detail (top view). - **1** non-shifted version, **2** shifted version.

There are four types of standard matrices available which are shown in figure 2.4. The inner parts of the modules are configured with 2x4 and 2x5 pins in a non-shifted and a shifted version. By leaving out some pins they can be reduced to any other configuration like 1x4 or 1x5 pins. In an extreme example there is only one pin left leading to a 1x1 module.

These inner parts are the basis for further combination of modules. By putting two or more inner parts together, new variants get available. In figure 2.5 the combination of 2x5 matrices with matrices of the same type is shown as an example for forming bigger matrices from smaller ones. New standard matrices obtained by this method are 4x4, 4x5, 6x4 and 6x5. These matrices are driven by two or more stimulation cards, because they are albe to drive 8 or 10 pins each.

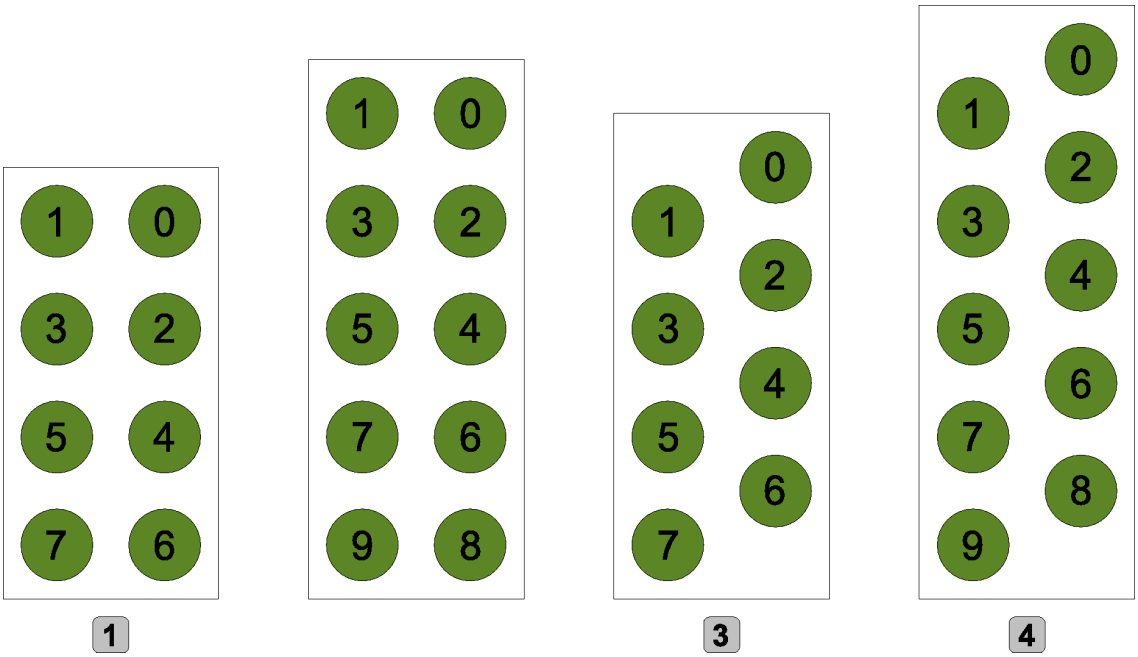

Figure 2.4: Different basic module variants (top view, cable near pin 0). - **1** 2x4 pins (non-shifted), **2** 2x5 pins (non-shifted), **3** 2x4 pins (shifted), **4** 2x5 pins (shifted).

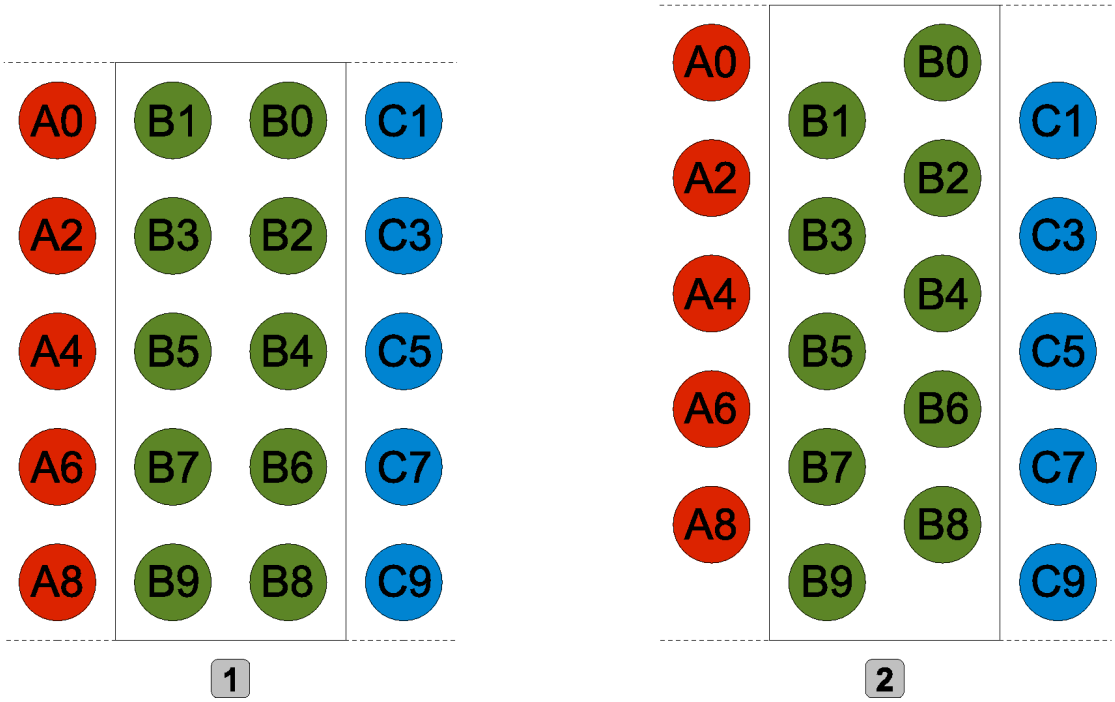

Figure 2.5: Combining 2x5 matrices in bigger modules, the neighbour modules are only drawn partially (top view, cable near pin 0). - **1** non-shifted, **2** shifted.

2.5 Two point discrimination module

The two point discrimination module (see drawing A.10) is a special module that has two pointed pins which are mounted in a way that their distance is variable. There are four pins inside the module which move on two axes. The height of the two pointed pins (Z-axis) can be controlled like on any other module with a pin matrix. The distance between the pins (X-axis) is also controllable via the master device in a range from nearly zero to several millimeters. As there are only four pins in one module, it is possible to connect two of these modules to one stimulation card via an adaptor. The configuration is carried out by special functions designed for these modules (see section 5.5).

The distance of pin tips (X-axis) has to be calibrated to get reproducible results. Calibration is carried out by QuaeroSys, which returns a list of calibration parameters. They are valid for a given operational range and can be entered in the program. After calibration the module can be used within this range by setting distances measured in millimeters. To get another distance range it is possible to rotate the pins inside of the module after unmounting the upper part. The pins are asymmetrical (see detail of drawing A.10), which makes it possible to get three different operational ranges by turning the pin tips to the outer or inner side of the module . Calibration cannot be guaranteed after changing pin orientation. Additionally this task may only be performed by qualified personnel.

3 Installation

3.1 Hardware

1. Unpack the device carefully and check for all items listed on the delivery note. If there are any discrepancies, feel free to contact us.
2. Connect the power cord to the back of the stimulator. The device accepts input voltages in different countries without changing any setting.
3. Depending on your operating system you will have to install different drivers and connect the device at different times (see below). The USB cable is connected to the PC and to the front of the controller card.

3.2 Software

3.2.1 Linux

1. Make sure you have a properly working USB device and you have permission to access it as the user you are working as. Working as root is discouraged.
2. Download the most current versions of the Linux USB driver package and the stimulator library package for Linux from our website.
3. Unpack the USB driver package and copy the current version of `libftd2xx.so` (including symlinks) to e. g. `/usr/lib` or any other path which is configured as a search path for libraries.
4. Unpack the stimulator library package and copy the current version of `libstimlib.so` (including symlinks) to e. g. `/usr/lib` or any other path which is configured as a search path for libraries.

5. Run `ldconfig` (as root)
6. Copy `stimlibrel.h` to a path accessable by programs linking to the stimulator library.

3.2.2 Windows®

Basic installation

1. Login as local administrator to have sufficient permissions to install drivers.
2. Download the most current versions of the Windows USB driver package and the stimulator library package for Windows from our website.
3. Unpack the two packages to seperate temporary directories.
4. Connect the device to the PC (it may be switched on or off).
5. Install the windows USB drivers when prompted and choose the temporary directory of the USB driver package as installation source.
6. Copy the current version of the stimulator library `stimlib.dll` to the directory `C:\windows\system32`.

Matlab®

1. Follow the steps of basic installation. After that you may switch to another user account.
2. Copy `stimlibrel.h` to a directory of your choice.
3. To use the stimulator in Matlab® you have to load the library and the header file once by using the `loadlibrary` command.

Notice: Matlab® does not handle whitespace in any path or file name in the right way. Even the Matlab® installation path must not contain any spaces, otherwise parts of the program will fail with weird error messages.

Presentation®

1. Follow the steps of basic installation and keep logged in as administrator.
2. Download the most current version of the Presentation® extension from our website.
3. Unpack the package to a temporary directory.
4. Copy the current version of the extension DLL `qspiezoext.dll` to the directory `C:\windows\system32`
5. Run the command `regsvr32 C:\windows\system32\qspiezoext.dll` (with exact version number).
6. Switch to the user account who will run Presentation®.
7. Add the registry information for the extension DLL `presextension.reg`
8. Copy `stimlibrel.pcl` to a directory of your choice.
9. To use the stimulator in Presentation® you have to include `stimlibrel.pcl` into your Presentation® file.

4 Programming

4.1 General

The Stimulator is internally clocked with 2 kHz (0.5 ms cycle duration). The clock controls the chronology of commands, which is structured by the command `wait`. Every call of `wait` completes one cycle and emits one or more internal trigger signals - all commands waiting for these trigger signals will be synchronously executed, whereas some commands do not wait for a trigger signal - they are simply executed at the next call of `wait`.

The Stimulator has eight DAC (Digital to Analog Converter) with 4096 different output voltages. These eight different voltages represent the possible pin heights which can be multiplexed to every single pin.

There are two methods to control the Stimulator. You could first set the DAC and let the pins switch between the DAC, which is a good method for stimulating with a rectangular signal. Or you could set the pins to one DAC an change the voltage of this DAC for example to stimulate with a sinus signal. You can also combine these two methods.

A stimulation program at least consists of `initStimulator`, `startStimulation` and `closeStimulator`. The stimulation sequence normally can be found between `initStimulator` and `startStimulation`.

4.2 External triggers

The trigger input and output ports serve a possibility to synchronize the Stimulator with the measurement. Therefore the two commands `waitForTrigger` and `triggerOut` are used (see sections 5.4.3 and 5.4.4).

If a single stimulation sequence consists of 5 s of a 20 Hz sinus followed by 5 s pause a `triggerOut` can be emitted when the sinus starts. Every time this sequence is repeated a trigger is emitted showing exactly the point where the pins started moving. In this way every single measurement can be synchronized to other trials and averaging can be done easily.

If the stimulation should start immeadiately after the subject is ready, but the time from starting the program until stimulation occurs is to long, `waitForTrigger` is the solution. To complete the example a `waitForTrigger` could be placed in front of the repeating stimulation sequence. The program is loaded to the stimulator (depending on size completely or just the beginning), but execution does not start until a trigger (rising edge by e. g. shorting and releasing the input) is received.

If the hardware version of the controller card is ≥ 1.0 the trigger output can be configured by using `setTriggerMode` and `setTriggerLength` (see sections 5.4.1 and 5.4.2). The trigger output signal can be configured as rising edge or a falling edge which is simultaneous with the trigger event. The time the signal is held high or low is given by `setTriggerLength`.

4.3 Data buffering

The stimulator has an integrated buffer (remote buffer), which is filled by the stimulation PC little by litte. If the remote buffer is full, stimulation data is generated and put into the local buffer on the stimulation PC. If this buffer is also full and the stimulation has not been started through `startStimulation` an error is generated, because there would be no chance to deplete the buffer. The best practice is to allocate a local buffer large enough to fit the whole program (by using `setProperty` with `local_buffer_size`) and send the start command in the end.

5 Command reference

5.1 General commands

5.1.1 `initStimulator`

Opens a new connection via USB to transmit data and commands. This command must be issued at the beginning of each stimulation program. Before you may use the stimulator with the different programs and operation systems you have to load and include some of the library files.

C++

```
#include "stimlibrel.h"
int initStimulator('<licence-code>')
```

PCL

```
include_once "C:\\<yourpath>\\stimlibrel.pcl"
loadLibrary();
initStimulator("<licence-code>");
```

Matlab®

```
loadlibrary('stimlib0.dll',
   'C:\<yourpath>\stimlibrel.h')
int calllib('stimlib0', 'initStimulator',
   '<licence-code>')
```

Return codes:

`OK`	Stimulator successfully initialized
`E_NOT_ENOUGH_MEMORY`	The stimulation PC has not enough memory to satisfy the property `local_buffer_size`
`E_NO_LICENSE`	The licence code is not set or incorrect

5.1.2 closeStimulator

Terminates the connection to the stimulator.

C++
```
int closeStimulator()
```

PCL
```
closeStimulator()
```

Matlab®
```
int calllib('stimlib0', 'closeStimulator')
```

Return codes:

OK	Connection to stimulator successfully closed

5.1.3 resetStimulator

Reboots the stimulator.

C++
```
int resetStimulator()
```

PCL
```
resetStimulator()
```

Matlab®
```
int calllib('stimlib0', 'resetStimulator')
```

Return codes:

OK	Stimulator successfully reset

5.1.4 setProperty

Sets the given property.

C++
```
int setProperty(char* property, int value)
```

PCL
```
setProperty('<property>', int value)
```

Matlab®
```
int calllib('stimlib0', 'setProperty',
  '<property>', int value)
```

Properties:

`local_buffer_size`	PC buffer in bytes (default 50 000). This property must be set before `initStimulator`. See chapter 4.3 for details on usage.

Return codes:

`OK`	Property successfully set to value
`E_NO_PROPERTY`	Property name unknown
`E_READONLY_PROPERTY`	Property is read-only and cannot be set
`BUFFER_SIZE_OUT_OF_RANGE`	Only for `local_buffer_size`: specified value out of range

5.1.5 `getProperty`

Reads the given property. These properties can be read at any time the library is loaded. They are not depending on a connection with the stimulation device, so you can get e.g. the final number of remote$_{b}uffer_{u}nderrunsafterclosingtheconnectiontothestimulator$.

C++
```
int getProperty(char* property)
```

Matlab®
```
int calllib('stimlib0', 'getProperty',
  '<property>')
```

Properties:

`local_buffer_size`	Available local buffer in bytes
`local_buffer_status`	Filling of the PC buffer in bytes
`remote_buffer_underruns`	Number of steps of 0.5 ms the remote buffer (stimulator) was empty
`remote_buffer_status`	Number of bytes in remote buffer (possibly inaccurate due to program execution)

5.1.6 `startStimulation`

Starts the eradication of the stimulation program while starting the internal clock of the stimulator.

C++

```
int startStimulation()
```

PCL

```
startStimulation()
```

Matlab®

```
int calllib('stimlib0', 'startStimulation')
```

Return codes:

`OK`	Stimulation successfully started
`E_NO_LICENSE`	The licence code is incorrect or empty

5.1.7 `stopStimulation`

Stops the eradication of the stimulation program.

C++

```
int stopStimulation()
```

PCL

```
stopStimulation()
```

Matlab®

```
int calllib('stimlib0', 'stopStimulation')
```

Return codes:

OK	Stimulation successfully stopped
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.2 Pin movement

5.2.1 setDAC

Sets one of eight DACs (0 to 7) to a value between 0 (pin down) and 4095 (pin up). After a reset, all DACs are set to 0, so all pins should be down.

Avoid using DAC 0, because all pins refer at startup to this DAC.

The command will be executed instantly when calling wait (no waiting for a special internal trigger signal).

C++

```
int setDAC(uint8_t dac, uint16_t value)
```

PCL

```
setDAC(int dac, int value)
```

Matlab®

```
int calllib('stimlib0', 'setDAC', int dac, int
   value)
```

Return codes:

OK	DAC set to new value
E_DAC_OUT_OF_RANGE	DAC channel not valid (0-7)
E_VALUE_OUT_OF_RANGE	DAC value out of range (0-4095)
E_DAC_LOCKED	DAC is locked for special use
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.2.2 setPinBlock8

Requires: Stimulation card for eight pins

Sets the pins of the chosen eight pin stimulation card (first parameter) to a DAC of 0 to 7. The command will be executed as soon as the defined internal trigger signal (second parameter) is emitted by wait.

C++

```
int setPinBlock8(uint8_t slot, uint8_t
   int_trigger, uint8_t pin0, uint8_t pin1,
   uint8_t pin2, uint8_t pin3, uint8_t pin4,
   uint8_t pin5, uint8_t pin6, uint8_t pin7)
```

PCL

```
setPinBlock8(int slot, int int_trigger, int
   pin0, int pin1, int pin2, int pin3, int pin4,
   int pin5, int pin6, int pin7)
```

Matlab®

```
int calllib('stimlib0', 'setPinBlock8', int
   slot, int int_trigger, int pin0, int pin1,
   int pin2, int pin3, int pin4, int pin5, int
   pin6, int pin7)
```

Return codes:

OK	Pins successfully assigned to DAC channels
E_SLOT_OUT_OF_RANGE	Slot of stimulation card not valid (0-15)
E_TRIGGER_OUT_OF_RANGE	Trigger number out of range (0-15)
E_DAC_OUT_OF_RANGE	DAC channel not valid (0-7)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.2.3 setPinBlock

This function is an alias to setPinBlock8. It only exists for compatibility reasons.

5.2.4 setPinBlock10

Requires: Stimulation card for ten pins

Sets the pins of the chosen ten pin stimulation card (first parameter) to a DAC of 0 to 7. The command will be executed as soon as the defined internal trigger signal (second parameter) is emitted by wait.

C++

```
int setPinBlock10(uint8_t slot, uint8_t
   int_trigger, uint8_t pin0, uint8_t pin1,
   uint8_t pin2, uint8_t pin3, uint8_t pin4,
   uint8_t pin5, uint8_t pin6, uint8_t pin7,
   uint8_t pin8, uint8_t pin9)
```

PCL

```
setPinBlock10(int slot, int int_trigger, int
   pin0, int pin1, int pin2, int pin3, int pin4,
   int pin5, int pin6, int pin7, int pin8, int
   pin9)
```

Matlab®

```
int calllib('stimlib0', 'setPinBlock10', int
   slot, int int_trigger, int pin0, int pin1,
   int pin2, int pin3, int pin4, int pin5, int
   pin6, int pin7, int pin8, int pin9)
```

Return codes:

`OK`	Pins successfully assigned to DAC channels
`E_SLOT_OUT_OF_RANGE`	Slot of stimulation card not valid (0-15)
`E_DAC_OUT_OF_RANGE`	DAC channel not valid (0-7)
`E_BUFFER_OVERFLOW`	Command cannot be queued in local buffer (see property `local_buffer_size`)

5.2.5 `send`

This function does not exist anymore. It has been replaced by `wait`. Due to offering the driver for more operating systems and applications this command had to be renamed. Please change your old programs accordingly.

5.2.6 `wait`

Completes the actual clock cycle and emits one or more (count) internal trigger signals. Commands waiting for this signal will be executed immediately.

C++

```
int wait(uint8_t int_trigger, unsigned long int
   count)
```

PCL

```
wait(int int_trigger, int count)
```

Matlab®

```
int calllib('stimlib0', 'wait', int int_trigger,
   int count)
```

Return codes:

OK	Trigger OK
E_TRIGGER_OUT_OF_RANGE	Trigger number out of range (0-15)
E_VALUE_OUT_OF_RANGE	Trigger length out of range (≥1)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3 Variables and I/O

5.3.1 setVar

Sets the given variable when program execution reaches the command. In the stimulator there a 100 user defined variables of 16 bit available which are numbered from 0 to 99.

C++

```
int setVar(uint8_t var, uint16_t value);
```

Matlab®

```
int calllib('stimlib0', 'setVar', int var, int
   value)
```

Return codes:

OK	Variable successfully set
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.2 setVarImmediate

Sets the given variable immediately.

This command does not queue in the command buffer, but is executed directly.

C++

```
int setVarImmediate(uint8_t var, uint16_t value);
```

Matlab®

```
int calllib('stimlib0', 'setVarImmediate', int
   var, int value)
```

Return codes:

OK	Variable successfully set
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.3 getVarImmediate

Fetches the value of the given variable from the stimulator.

This command does not queue in the command buffer, but is executed directly.

C++

```
uint16_t getVarImmediate(uint8_t var);
```

Matlab®

```
uint16_t calllib('stimlib0', 'getVarImmediate',
   int var)
```

5.3.4 incVar

Increments the given variable by one.

C++

```
int incVar(uint8_t var);
```

Matlab®

```
int calllib('stimlib0', 'incVar', int var)
```

Return codes:

OK	Variable successfully increased
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.5 decVar

Decrements the given variable by one.

C++

```
int decVar(uint8_t var);
```

Matlab®

```
int calllib('stimlib0', 'decVar', int var)
```

Return codes:

OK	Variable successfully decreased
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.6 outPort8

Requires: I/O card

Output the given value of 8 bit to a I/O card port.

C++

```
int outPort8(uint8_t slot, uint8_t port, uint8_t
   value);
```

PCL

```
outPort8(int slot, int port, int value);
```

Matlab®

```
int calllib('stimlib0', 'outPort8', int slot,
   int port, int value);
```

Return codes:

OK	I/O command successful
E_SLOT_OUT_OF_RANGE	Slot of I/O card not valid (0-15)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.7 outPort16

Requires: I/O card

Output the given value of 16 bit to two I/O card ports.

C++

```
int outPort16(uint8_t slot, uint8_t portH,
   uint8_t portL, uint16_t value);
```

PCL

```
outPort16(int slot, int portH, int portL, int
   value);
```

Matlab®

```
int calllib('stimlib0', 'outPort16', int slot,
   int portH, int portL, int value);
```

Return codes:

OK	I/O command successful
E_SLOT_OUT_OF_RANGE	Slot of I/O card not valid (0-15)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.8 outPortVar16

Requires: I/O card

Output the 16 bit value of var to two I/O card ports.

If only 8 bit are used set the other port to 0xFF.

C++

```
int outPortVar16(uint8_t slot, uint8_t portH,
   uint8_t portL, uint8_t var);
```

PCL

```
outPortVar16(int slot, int portH, int portL, int
   var);
```

Matlab®

```
int calllib('stimlib0', 'outPortVar16', int
   slot, int portH, int portL, int var);
```

Return codes:

OK	I/O command successful
E_SLOT_OUT_OF_RANGE	Slot of I/O card not valid (0-15)
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.3.9 inPortVar16

Requires: I/O card

Reads the value of two I/O card ports into the internal 16 bit variable.

If only 8 bit are used set the other port to 0xFF.

C++

```
int inPortVar16(uint8_t slot, uint8_t portH,
   uint8_t portL, uint8_t var);
```

PCL

```
inPortVar16(int slot, int portH, int portL, int
   var);
```

Matlab®

```
int calllib('stimlib0', 'inPortVar16', int slot,
   int portH, int portL, int var);
```

Return codes:

OK	I/O command successful
E_SLOT_OUT_OF_RANGE	Slot of I/O card not valid (0-15)
E_VAR_OUT_OF_RANGE	Number of variable out of range (0-99)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.4 Triggers

5.4.1 setTriggerMode

Sets the trigger output mode to one of the following modes.

Trigger mode is changed immediately after sending the command. It should only be called once at the beginning of the program to apply the right settings for the receiving device.

The command only affects HW versions ≥1.0, otherwise (default or HW version <1.0) trigger mode is RISING_EDGE.

C++

```
int setTriggerMode(uint8_t slot, uint8_t port,
   uint8_t mode);
```

PCL

```
setTriggerMode(int slot, int port, int mode);
```

Matlab®

```
int calllib('stimlib0', 'setTriggerMode', int
   slot, int port, int mode)
```

Trigger modes:

FALLING_EDGE	Mode 0: Trigger starts with falling edge (1 → 0)
RISING_EDGE	Mode 1: Trigger starts with rising edge (0 → 1)

When using Presentation or Matlab® you cannot use the constant names as these do not know about constants. You will have to use the mode numbers as stated above.

Return codes:

OK	Trigger mode successfully applied
E_SLOT_OUT_OF_RANGE	Slot of controller card or I/O card not valid (0-16)
E_MODE_OUT_OF_RANGE	Mode not correct (see above)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.4.2 setTriggerLength

Sets stimulator to generate trigger outputs of a given length. The length is measured in steps of 0.5 ms.

Trigger length is changed after finishing the last trigger. It should only be called once at the beginning of the program to apply the right settings for the receiving device.

This command only affects HW versions ≥1.0, otherwise trigger length is 0.5 ms (default) or 500 ns (HW version <1.0).

C++

```
int setTriggerLength(uint8_t slot, uint8_t port,
   uint8_t length);
```

PCL

```
setTriggerLength(int slot, int port, int length);
```

Matlab®

```
int calllib('stimlib0', 'setTriggerLength', int
   slot, int port, int length)
```

Return codes:

OK	Trigger length successfully applied
E_SLOT_OUT_OF_RANGE	Slot of controller card or I/O card not valid (0-16)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.4.3 waitForTrigger

Waits for an external trigger signal (rising edge) on the selected port. The device pauses command execution when reaching this command. It does not block the stimulation PC or any program on it until the trigger signal arrives.

If there is only a single controller card the only possible arguments for this function are slot 0x10 (decimal 16) and port 0x00.

C++

```
int waitForTrigger(uint8_t slot, uint8_t port);
```

PCL

```
waitForTrigger(int slot, int port)
```

Matlab®

```
int calllib('stimlib0', 'waitForTrigger', int
   slot, int port)
```

Return codes:

OK	Stimulator will wait for incoming trigger
E_SLOT_OUT_OF_RANGE	Slot of controller card or I/O card not valid (0-16)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.4.4 triggerOut

Emits an external trigger signal on the selected port.

In hardware version <1.0 trigger output is always a positive pulse (logical '1') with a width of 500 ns. In hardware version version ≥1.0 the pulse is configurable. It can be positive (logical '1') or negative (logical '0') with a variable length in multiples of 500 µs. To modify these configuration parameters the commands setTriggerMode and setTriggerLength are used.

The command will be executed instantly on calling wait, where the number of wait cycles has to be at least as high as the current trigger length for complete execution of the trigger.

If there is only a single controller card the only possible arguments for this function are slot 0x10 (decimal 16) and port 0x01.

C++

```
int triggerOut(uint8_t slot, uint8_t port);
```

PCL

```
triggerOut(int slot, int port)
```

Matlab®

```
int calllib('stimlib0', 'triggerOut', int slot,
   int port)
```

Return codes:

OK	Stimulator will emit a trigger
E_SLOT_OUT_OF_RANGE	Slot of controller card or I/O card not valid (0-16)
E_BUFFER_OVERFLOW	Command cannot be queued in local buffer (see property local_buffer_size)

5.5 Two point discrimination

5.5.1 set2PDProperties

Sets basic parameters for two point discrimination modules.

A call to this function is needed for the usage of each two point discrimination module identified by the first parameter. The slot gives the card where the module is connected, subslot specifies the connection on the adaptor (if applicable, otherwise this is 1). The following three parameters assign DACs to the axes of the module. The first DAC controls the distance and is occupied by the driver if X-calibration is used (see `set2PDCalibrationX`). DACs assigned to the pins moving in Z-direction (up/down) are user specific like pins on normal matrix modules. If both pins should always move in the same way, they can simply bound to the same DAC. If Z-calibration (see `set2PDCalibrationZ`) is used there have to be given two different DACs.

C++

```
int set2PDProperties(uint8_t module, uint8_t
   slot, uint8_t subslot, uint8_t dac_x, uint8_t
   dac_z0, uint8_t dac_z1)
```

PCL

```
set2PDProperties(int module, int slot, int
   subslot, int dac_x, int dac_z0, int dac_z1)
```

Matlab®

```
int calllib('stimlib0', 'set2PDProperties', int
   module, int slot, int subslot, int dac_x, int
   dac_z0, int dac_z1)
```

Return codes:

`OK`	Stimulator accepted parameters
`E_2P_MODULE_OUT_OF_RANGE`	Module identification number out of range (0-1)
`E_SLOT_OUT_OF_RANGE`	Slot or subslot number out of range
`E_DAC_OUT_OF_RANGE`	DAC channel not valid (0-7)
`E_BUFFER_OVERFLOW`	Command cannot be queued in local buffer (see property `local_buffer_size`)

5.5.2 set2PDCalibrationX

Assigns calibration data for X-axis to a specific module.

Calibration data is given as 10+1 values used for the right interpretation of the distance given by `set2PDDistance`. The module has to be identified with the first parameter as each individual module has a set of calibration data. As the driver needs full control over the DAC assigned to the X-axis, it will be locked and cannot be used otherwise.

C++

```
int set2PDCalibrationX(uint8_t module, uint16_t
   homeDACPos, double co0, double co1, double
   co2, double co3, double co4, double co5,
   double co6, double co7, double co8, double
   co9)
```

PCL

```
set2PDCalibrationX(int module, int homeDACPos,
   double co0, double co1, double co2, double
   co3, double co4, double co5, double co6,
   double co7, double co8, double co9)
```

Matlab®

```
int calllib('stimlib0', 'set2PDCalibrationX',
   int module, int homeDACPos, double co0,
   double co1, double co2, double co3, double
   co4, double co5, double co6, double co7,
   double co8, double co9)
```

Return codes:

`OK`	Stimulator accepted parameters
`E_DAC_LOCKED`	DAC is locked for special use

5.5.3 set2PDCalibrationZ

Assigns calibration data for both Z-axes to a specific module.

Calibration data is given as two values for each axis used for the right interpretation of the pin height given by `set2PDHeight`. The module has to be identified with the first parameter as each individual module has a set of calibration data. As the driver needs full control over both DACs assigned to the Z-axes, they will be locked and cannot be used otherwise. Please be sure that there have been assigned two different DACs in `set2PDProperties` otherwise there will be an error.

C++

```
int set2PDCalibrationZ(uint8_t module, double
   Z0_co0, double Z0_co1, double Z1_co0, double
   Z1_co1);
```

PCL

```
set2PDCalibrationZ(int module, double Z0_co0,
   double Z0_co1, double Z1_co0, double Z1_co1)
```

Matlab®

```
int calllib('stimlib0', 'set2PDCalibrationZ',
   int module, double Z0_co0, double Z0_co1,
   double Z1_co0, double Z1_co1)
```

Return codes:

`OK`	Stimulator accepted parameters
`E_DAC_LOCKED`	DAC is locked for special use

5.5.4 `set2PDDistance`

Move pins of a two point discrimination module to a given distance.

This function actually positions the pins to the given distance measured in millimeters. It maintains retreating pins of the Z-axis inside of the case by smoothly setting the DACs assigned by `set2PDProperties` to zero. Then it moves X-axis back to the starting position and after that to the newly given position. It does not move pins of the Z-axis up, as this should be done by the user. The whole positioning process takes one second, therefore it has influence on the timing of the program as there was a `wait(1, 2000)`.

C++

```
int set2PDDistance(uint8_t module, double
    distance)
```

PCL

```
set2PDDistance(int module, double distance)
```

Matlab®

```
int calllib('stimlib0', 'set2PDDistance', int
    module, double distance)
```

Return codes:

OK	Stimulator accepted parameters
E_VALUE_OUT_OF_RANGE	Distance value out of range

5.5.5 set2PDHeight

Preset pin height measured in promille of full extend.

This function works like setDAC with the difference that it honours the calibration for Z-axis. This function does not actually move the pins out, but waits for the next internal trigger as setDAC does. The value of 0 has a special meaning as it does not take calibration data into account, but sets the corresponding DAC to 0.

C++

```
int set2PDHeight(uint8_t module, uint16_t
    promille_Z0, uint16_t promille_Z1);
```

PCL

```
set2PDHeight(int module, int promille_Z0, int
    promille_Z1)
```

Matlab®

```
int calllib('stimlib0', 'set2PDHeight', int
    module, int promille_Z0, int promille_Z1)
```

Return codes:

`OK`	Stimulator accepted parameters
`E_VALUE_OUT_OF_RANGE`	Height value out of range

A Technical drawings

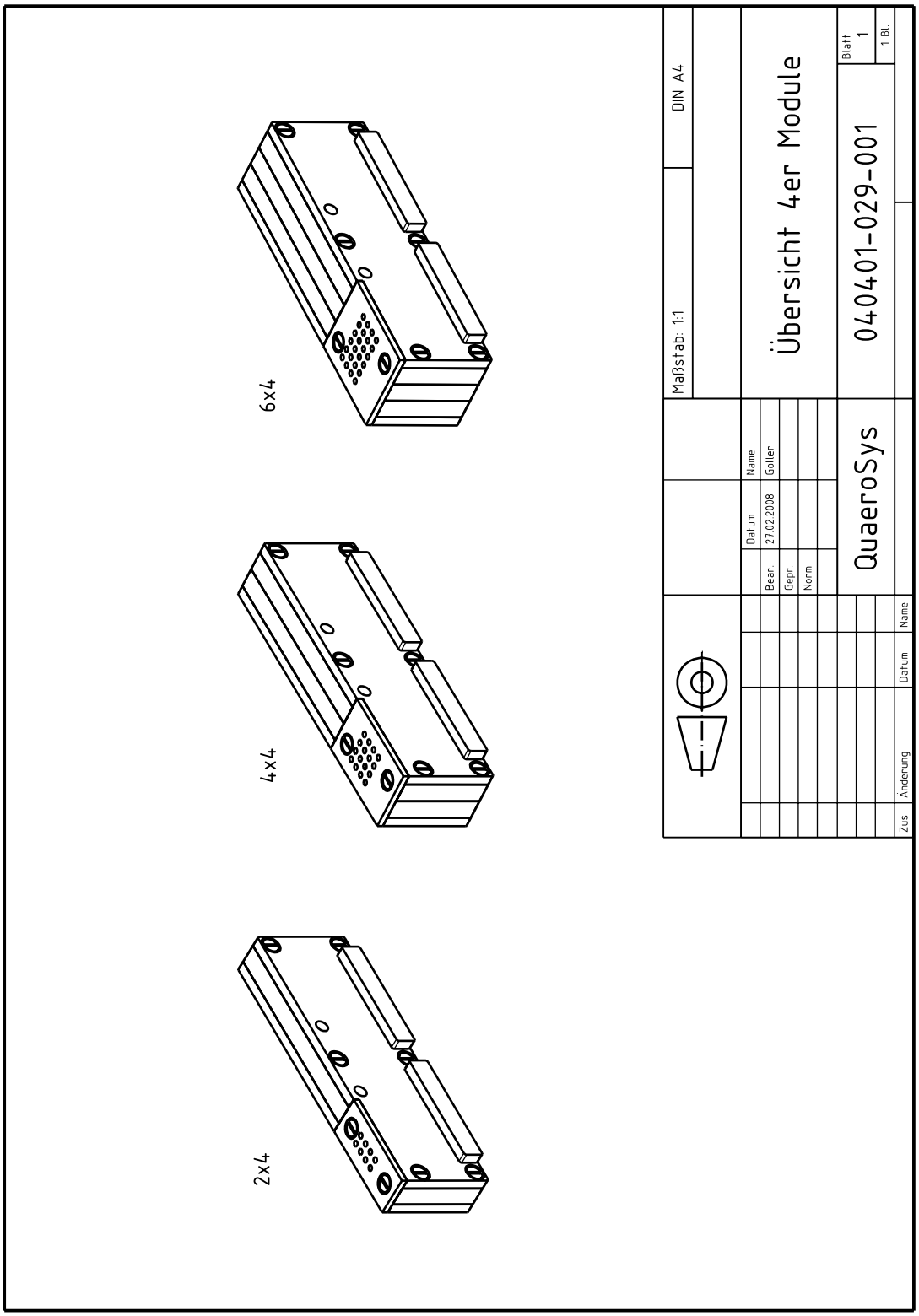

Figure A.1: Different module variants with 4 pins in a row.

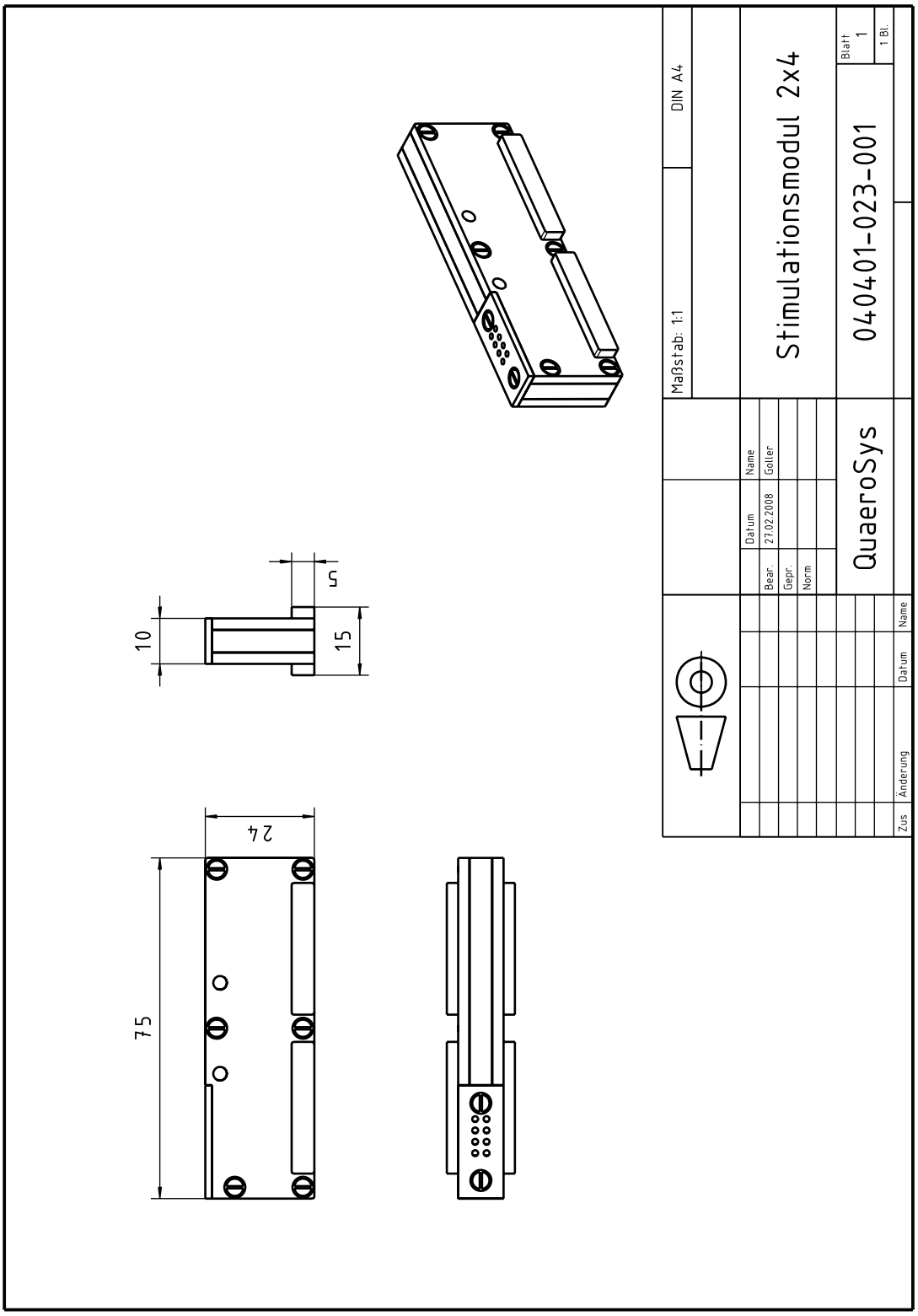

Figure A.2: Module variant 2 rows of 4 pins.

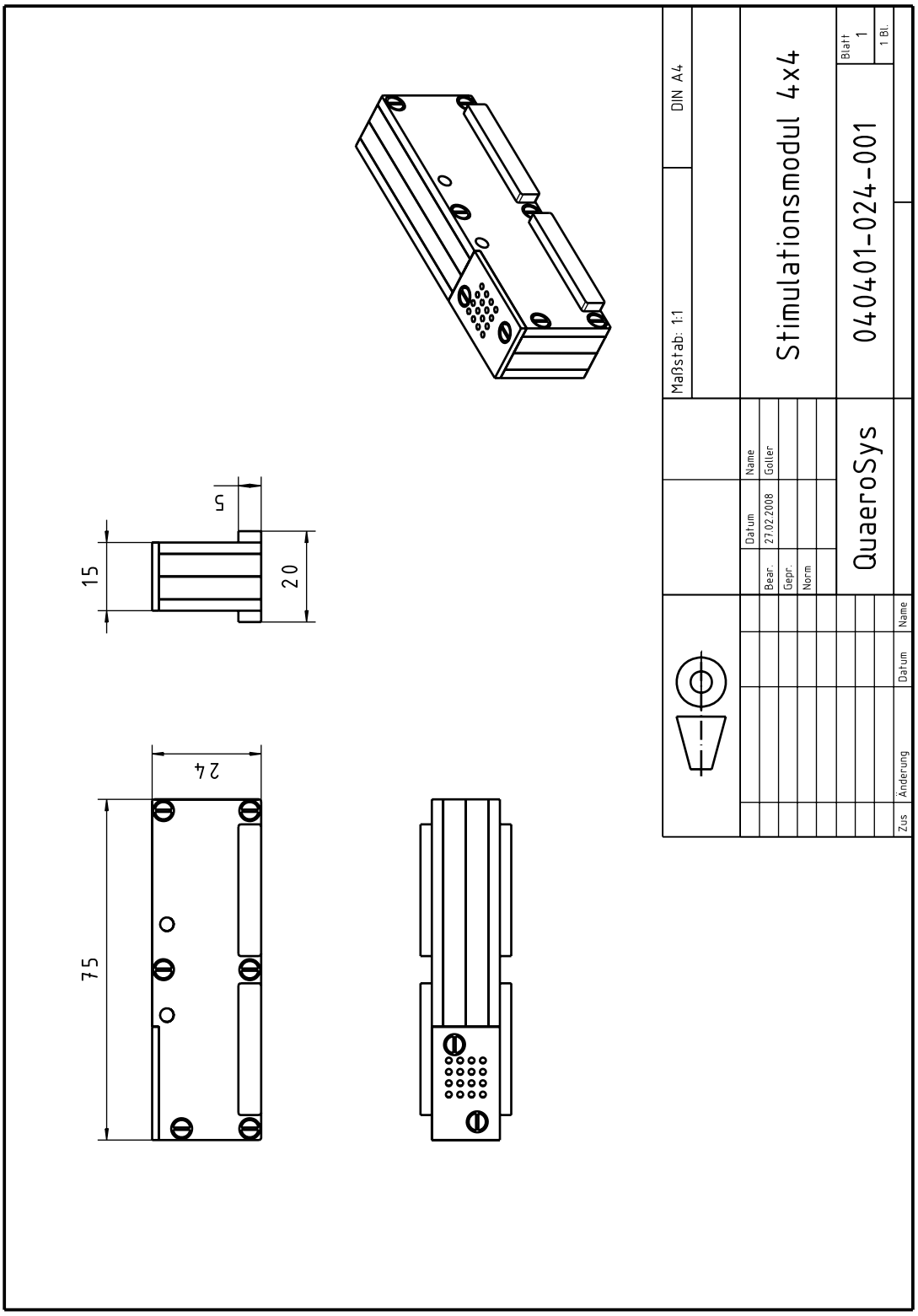

Figure A.3: Module variant 4 rows of 4 pins.

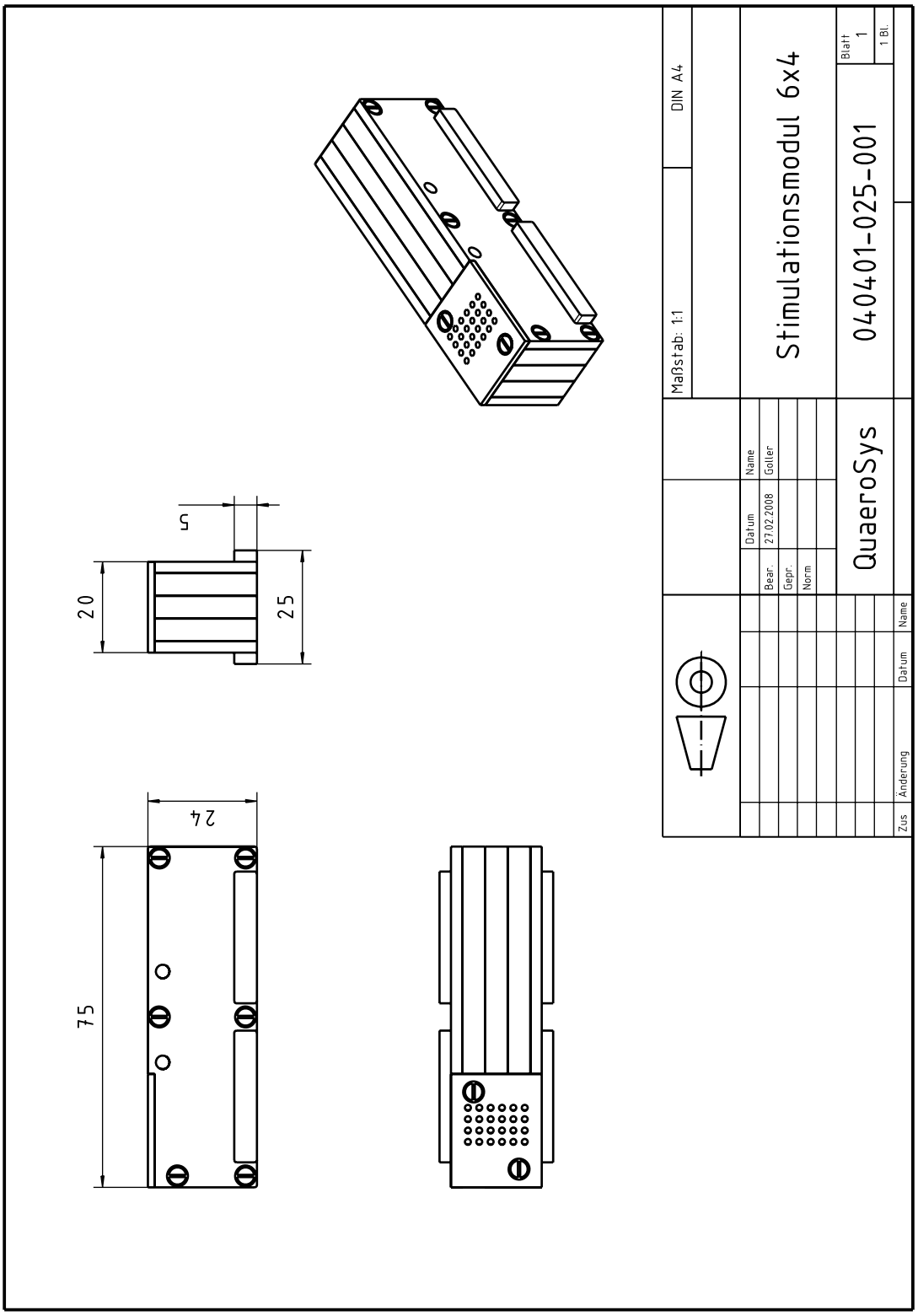

Figure A.4: Module variant 6 rows of 4 pins.

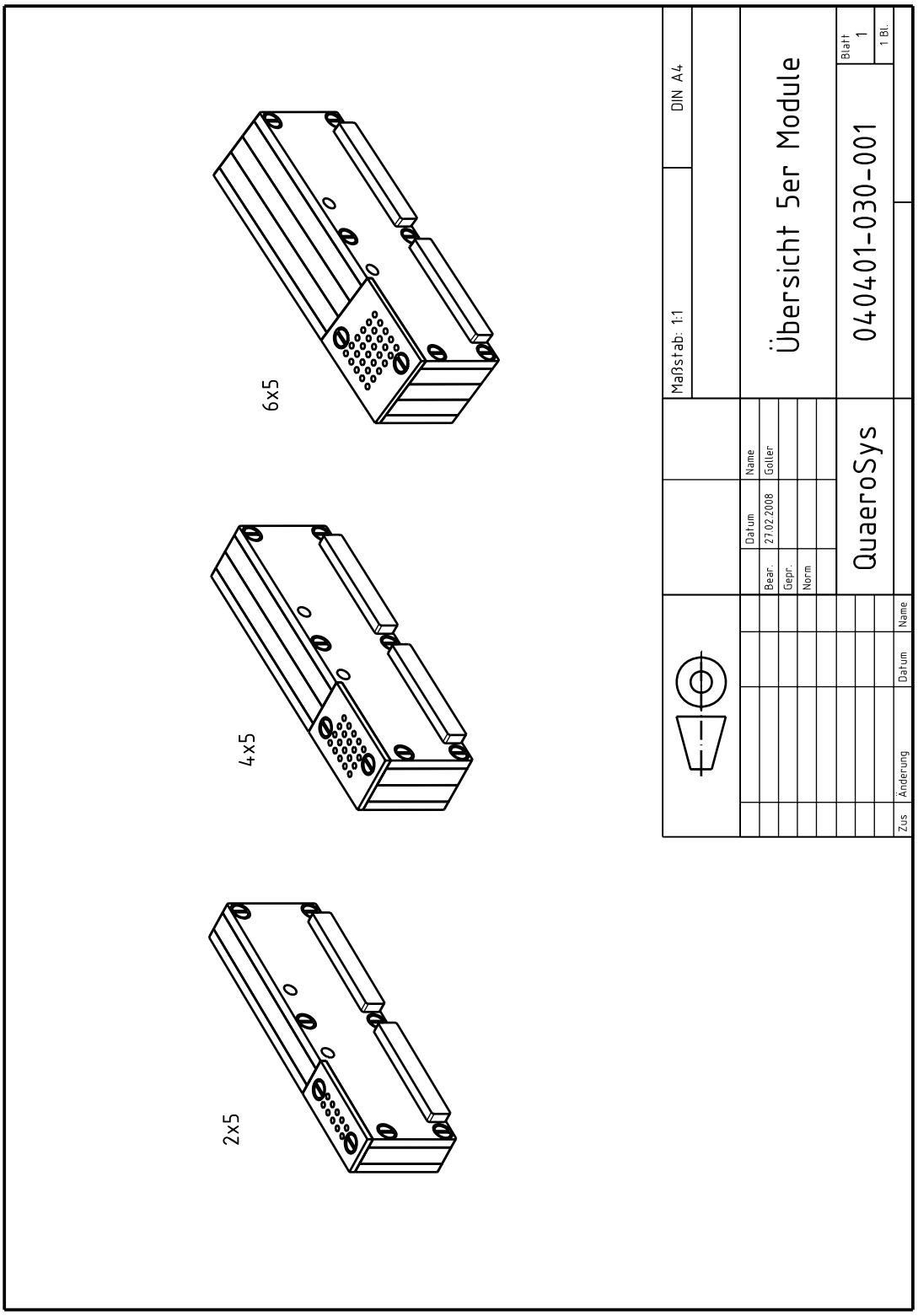

Figure A.5: Different module variants with 5 pins in a row.

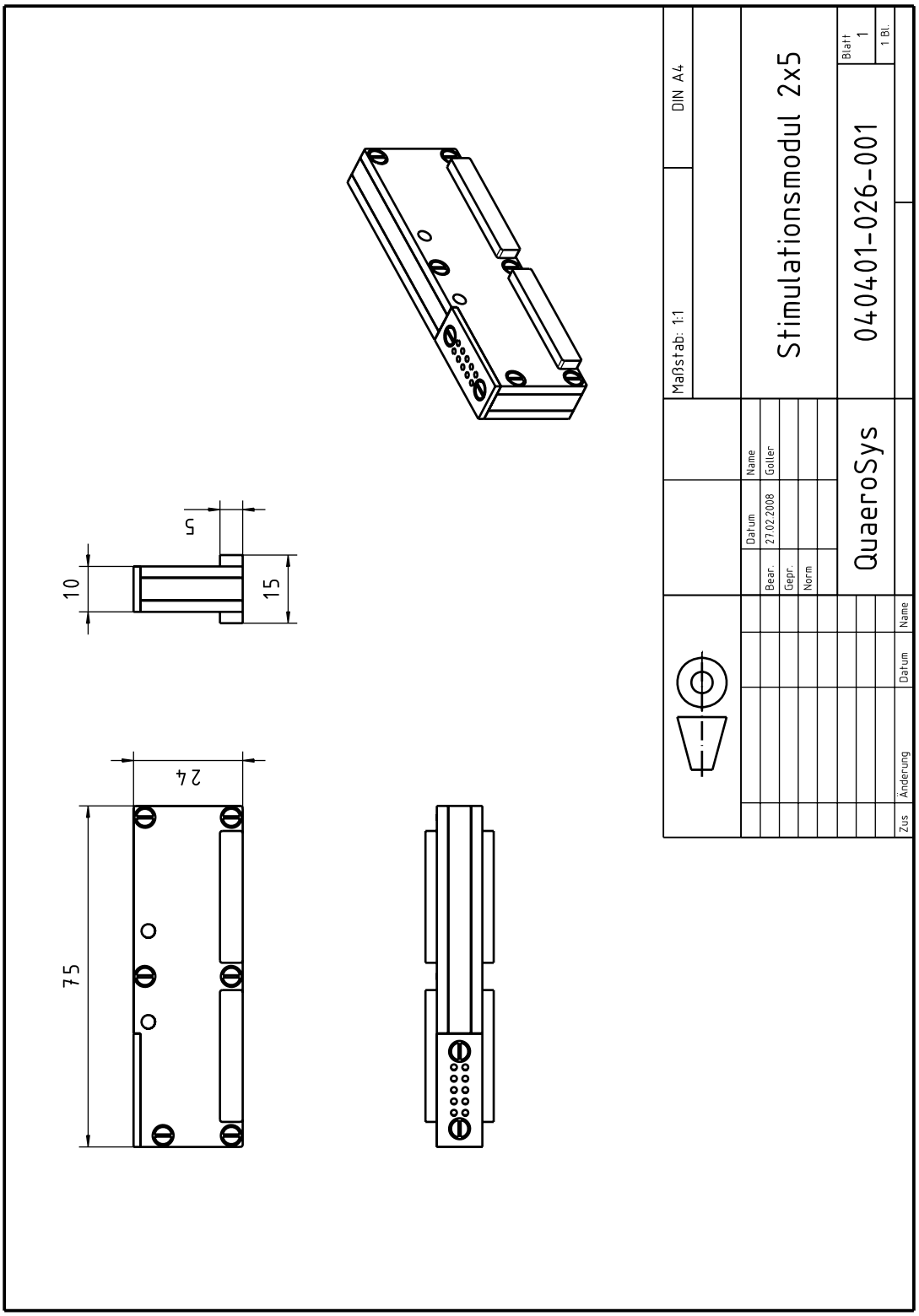

Figure A.6: Module variant 2 rows of 5 pins.

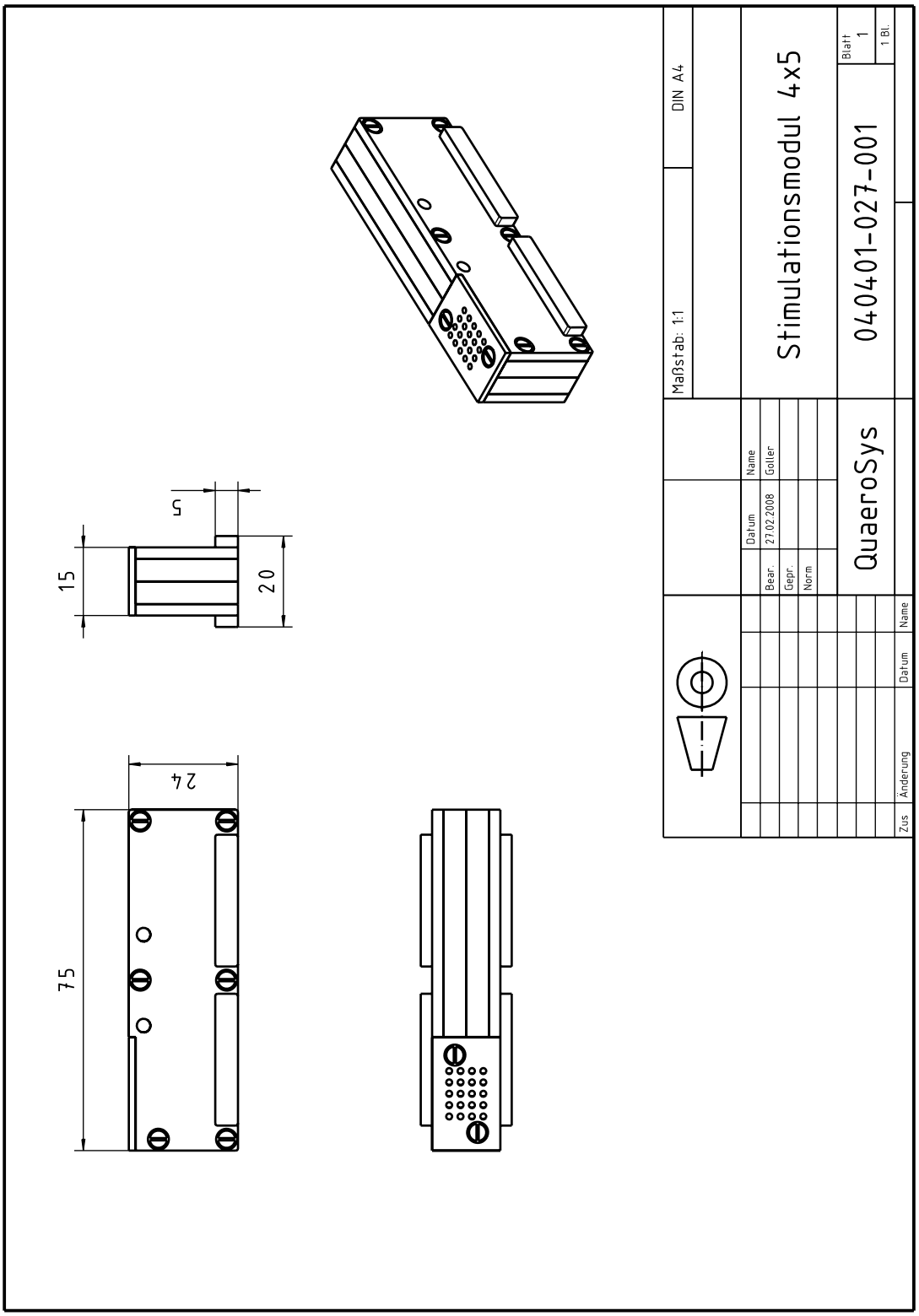

Figure A.7: Module variant 4 rows of 5 pins.

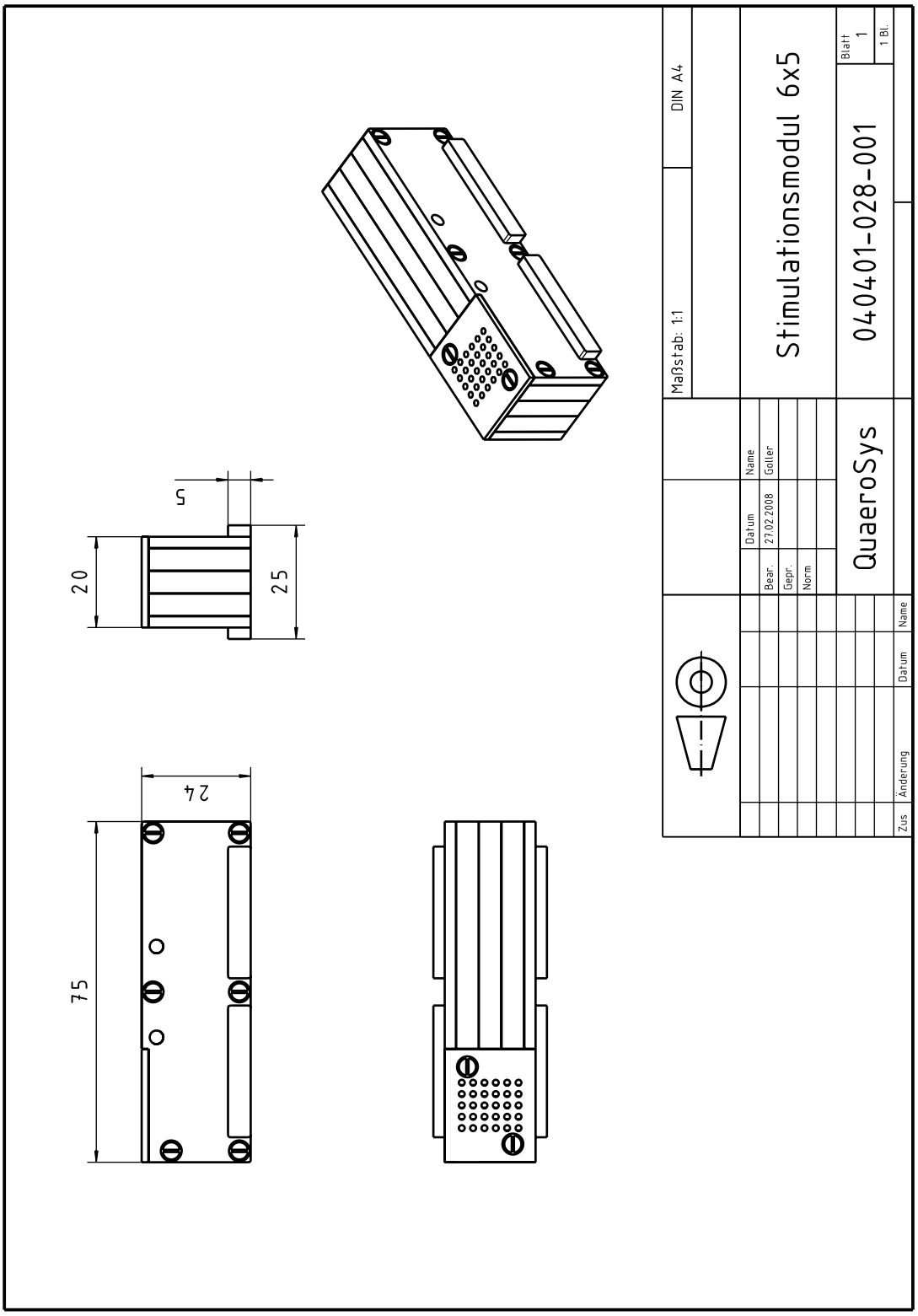

Figure A.8: Module variant 6 rows of 5 pins.

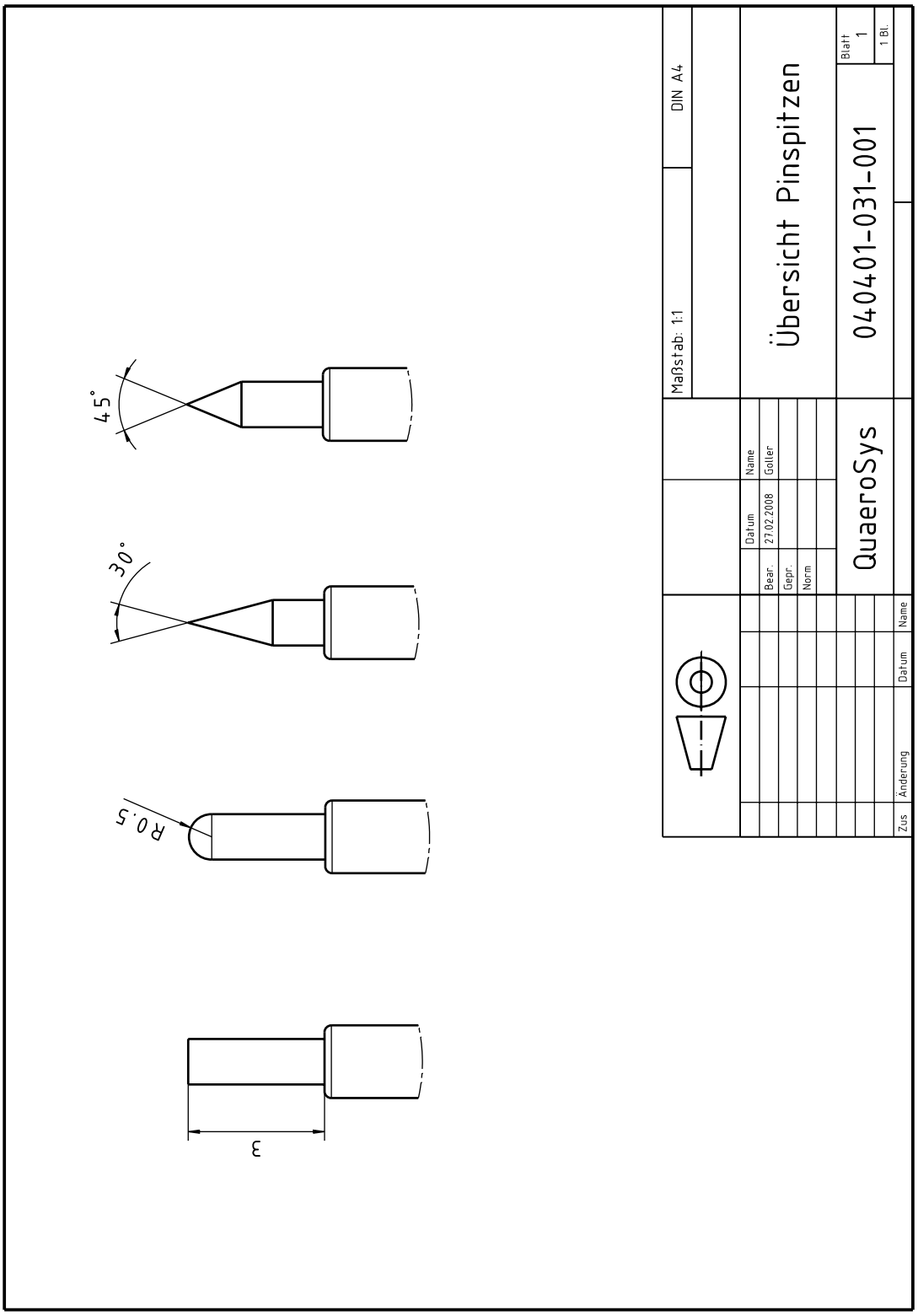

Figure A.9: Different variants of pin heads.

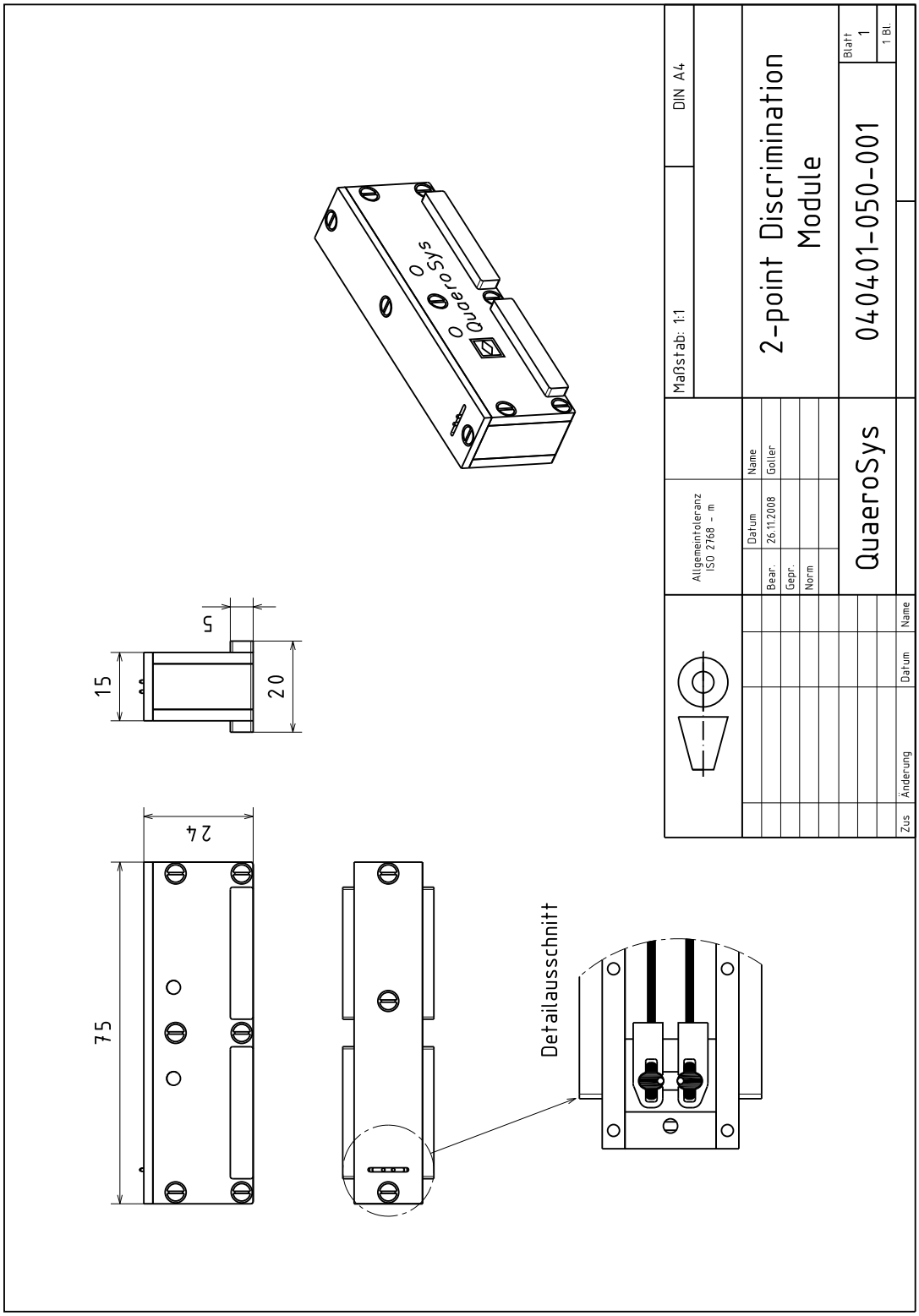

Figure A.10: Two point discrimination module.

B Revision History

V 0.1	12.02.2005	
V 0.2	16.06.2005	
V 0.3	01.09.2005	
V 0.4	09.05.2008	Complete rewrite because of Matlab
V 0.5	12.06.2008	Added drawings, restructuring
V 0.5.1	29.07.2008	Updated installation information
V 0.6	19.08.2008	Changes regarding new driver versions
V 0.7	24.04.2009	Added two-point discrimination
		Changes in `setPinBlock` commands
		Added hyperlinks and bookmarks
		Fixes in description of trigger commands