# ZIMPOL Camera Communication and Control

31. 5. 2024

This document describes all the details of the ZIMPOL camera communication and control required for the software development of the ENANTIOS Brachium instrument.

The cameras for the Brachium instrument are based on new electronics developed at SUPSI in the last two years. However, the software in the camera is an adapted old version originally developed at ETH about 20 years ago. A new version is currently under development at SUPSI mainly for the ZIMPOL cameras used at IRSOL but it is likely that ENANTIOS will use the new software in future too. The software will still use the ZIMPOL communication protocol described in this document but with some optimizations of the protocol and some control functions will be different. Therefore, some hints are given to make the Brachium software already be prepared for likely changes in future camera software versions.

## Camera communication

The standard TCP/IP transport protocol is used but the application protocol is nonstandard (the ZIMPOL protocol). The camera acts as a server. For both up and down stream data only a single client socket is required for the application. The server port is usually 15000 but can be changed (in a configuration file of the camera).

## Description of the ZIMPOL protocol

The ZIMPOL protocol was originally defined at ETH for the communication of the various hardware and software components of the ZIMPOL instrument (not only the camera). A free and easy human readable ASCII format is used for most interaction except for a special binary format to transfer large data sets like the images of a camera.

In the following section a rather abstract description of the syntax is given. It was taken from the original definition but reduced to the subset of the entire syntax sufficient for hardware components. More practical explanations and examples are given in the section afterwards.

### Generic syntax

The exact syntax of the messages is defined below by production rules. Starting from the non-terminal start symbol *`$message`*, a set of repeatedly applicable replacement rules allows all syntactically correct messages to be generated. Non-terminal symbols start with a $ sign and are printed in italics. For each of these symbols, a vertical list of alternatives is given. These can contain terminal symbols (printed in bold), further non-terminal symbols, or a regular expression (printed in italics). Graphically invisible terminal symbols such as line feed and control characters are represented in the notation `<ASCIIname>`.

```
$message
  $mbody $eol
  <EOT>
```

The **<EOT>** byte is foreseen to indicate to terminate the TCP/IP communication.

```
$mbody
  $status
  $cmdgroup

$eol
  <LF>
  <CR>
  <CR><LF>
```

The camera accepts all three alternatives of end of line bytes. For the client the recommendation is to use a line feed only. The camera server currently sends **<CR><LF>** but this may change in the future and therefore for the client it is recommended to implement the acceptance of all alternatives.

```
$status
  $cmdIdentifier:$umbody
  $cmdIdentifier;

$cmdgroup
  $cmd
  $cmd $cmdgroup
```

It is recommended to send groups of commands separated with end of line bytes instead of a white space.

```
$cmd
  $cmdIdentifier
  $assignment
  $inquiry

$assignment
  $reference=$constant

$inquiry
  $reference?

$reference
  $parIdentifier
  $parIdentifier[$subscripts]


$subscripts
  $subscript
  $subscript,$subscript


$subscript
  [0-9]*



$cmdIdentifier
$parIdentifier
  [a-zA-Z][a-zA-Z0-9_]*
```

```
$umbody
  $umword
  $umbody $umword

$umword
  $word
  {$word}

$word
  [!"#$%'()*+,-./0-9:;<=>?@A''-Z[\]^_`a-z|~]+

$constant
  $integer
  $real
  $string
  $binarydata

$integer
  $sdecimal
  $binary
  $octal
  $hex

$sdecimal
  $udecimal
  -$udecimal
  +$udecimal

$udecimal
  0
  [1-9][0-9]*

$binary
  0b[0-1]+

$octal
  0o[0-7]+

$hex
  0x[0-9a-fA-F]+
```

The camera should send integer values as decimal constants only. Therefore, the implementation of the other formats (binary, octal and hexadecimal) is most likely not required.

```
$real
  $sdecimal.$udecimal$exponent
  $sdecimal.$udecimal
  $sdecimal.

$exponent
    e$sdecimal
    E$sdecimal

$string
  ["][!#$%'()*+,-./0-9:;<=>?@A''-Z[\]^_`a-z{|}~]*["]
  ["][\040\041\043-\176]["]
  {$umbody}
```

A (simple) character string delimited with `"` may contain all printable ASCII characters including the single space, but no `"`. The alternative variant, the block string delimited with `{}` can contain all printable ASCII characters and control characters such as line feeds etc. *($umbody)*. Character strings can be empty.

The camera sends string values delimited with `{}` although they are all simple strings. This will most likely be changed in the new software version. Therefore, the client software should be able to handle both variants of string delimiter.

```
$binarydata
  <SOH>$binDescriptor<STX>$bytes<ETX>

$binDescriptor
  $btype$bsize
  $btype$bsize $binAttributes

$btype
  s8
  s16
  s32
  s64
  u8
  u16
  u32
  u64
  f32
  f64
```

The small letter stands for signed (s), unsigned (u) and floating-point values (f). The number is the bit size.

```
$bsize
  [$numelements]
  [$numrows,$numcolumns]
  [$numframes,$numrows,$numcolumns]

$binAttributes
  $binAttributes $binAttribute
  $binAttribute

$binAttribute
  $attrIdentifier=$constant

$bytes
  $bytes $byte
  $byte
```

The camera used the binary format to send images. A more detailed explanation is given in the section camera operation.

# Explanatory notes and examples

## Parameter and commands

The camera is mainly controlled by sending scalar parameters and commands. Parameters are transmitted with an assignment (*$parIdentifier*=*$constant*), e.g.:

```
it=1.5001
isl=20
seq_phase[1]=145
```

The last example is an array parameter where the element with the index number 1 is sent.

For a command the name of the command (*$cmdIdentifier*) is transmitted, e.g.:

```
gi
gis
abort
```

Commands but also many parameters immediately trigger a function in the camera. If the function could be executed successfully the camera responds with a status message. For parameters with *$parIdentifier=$constant* where *$constant* is the actual value that could possibly be slightly different from the transmitted value. E.g.

```
it=1.5
```

For commands the response is *$cmdIdentifier;* e.g.

```
gis;
```

The ZIMPOL protocol also has the alternative variant *$cmdIdentifier*:**Done** that is currently not used by the camera but may be implemented in the client program too.

In case the function cannot be executed or fails during execution for parameter and commands the response is `$Identifier:$umbody` as soon as it fails. Where *$umbody* contains an error message in the general form:

```
ERROR: some optional description of the failure
```

E.g. setting a negative exposure time is not possible and after sending `it=-0.1` the camera response is:

```
it: ERROR: set error in "it=-0.1"
```

The keyword **ERROR:** (including the : character) at the beginning of the message indicates the error. It is not a clean implementation of the protocol because the correct keyword would be **Error:**. A non-case-sensitive check of the message keywords may be a good solution for possible future changes. The example does also include a space after `it:` that is not mandatory. The protocol does allow to include space characters between tokes although it is not recommended to do.

## Parameter inquiry

Parameter inquiries are done by sending the name of the parameter with a ? at the end. Array parameters can only be inquired element by element. Examples:

```
it?
isl?
Name?
seq_phase[0]?
seq_phase[1]?
```

If successful, the response comes again in the assignment format:

```
it=1.20
isl=10
Name={cam1}
seq_phase[0]=37
seq_phase[1]=560
```

or an error status message is sent in case something went wrong.

## Asynchronous interactions

Interactions can run in an asynchronous way if the camera supports this. E.g. several parameter inquiries or assignments can be sent at once before waiting for the first response.

Further the camera can send status messages and parameters at any time. Especially when a function in the camera is running that takes a longer time like image acquisitions where various informative status messages and image data is sent.

# Camera operation

## Parameters and commands

The following table lists relevant parameters and commands of the camera that are used for normal camera operation. At the camera startup the parameters are set to the most useful value for operation. Some of them may never be changed during operation e.g. the image size.

The first column is the name (identifier). The second column describes the data type with a capital letter for integers (I), real (R), and string (S) values. The letter C is used for commands. In the case of array parameters, the dimensions are given in square bracket [d] behind the type. The ZIMPOL binary data format description is used if the parameter transmits the value in this format. If the parameter has a special mode it is indicated in round brackets (mode). The (ro) indicates a read only parameter. The (so) is used for parameters sent from the camera only, that cannot be written and inquired by the client.

| Name | Type[d] (mode) | Unit | Description |
|---|---|---|---|
| **Image acquisition, measurements** | | | |
| it | R | seconds | Integration time (acquisition time) |
| isl | I | > 0 | Image sequence length (number of images) |
| ifn | I (ro) | | Input frame number |
| seq_mode | I | | Image sequence mode |
| seq_state | I (so) | [0-3] | Image sequence state (send in img data header) |
| seq_phase | I[4] | | Image sequence phase |
| img | u16[r,c] (so) | | Image data (binary data format) |
| gi | C | | Get image (single image acquisition) |
| gis | C | | Get image sequence (starts a measurement) |
| abort | C | | Aborts image acquisition |
| **Image size parameters** | | | |
| img_h | I | | Image height (number of rows) |
| img_w | I | | Image width (number of columns) |
| img_t | I | | Image top |
| img_l | I | | Image left |
| **Temperature control** | | | |
| tr | R | °C | Temperature requested (set point) |
| tc | R (ro) | °C | Temperature cold side, image sensor temperature |
| tw | R (ro) | °C | Temperature warm side (Peltier elements secondary cooling) |
| pp | R (ro) | [0.0 - 1.0] | Peltier power |
| tr_limited | R (ro) | °C | Send while temperature changes |
| pphl | R | [0.0 - 1.0] | Peltier power high limit |
| twhl | R | °C | Temperature warm side high limit |

|  |  |  |  |
|---|---|---|---|
| tc_mode | I | [0-4] | Temperature control mode<br>0: off<br>1: read temperature<br>2: control temperature<br>3: set Peltier power manually<br>4: control stopped |
| humidity | R (ro) | % | Humidity electronic side |
| temp_humid | R (ro) | °C | Temperature electronic side |
| temp_dew | R (ro) | °C | Dew point |
| vacuum | R (ro) | bar | Pressure vacuum side |
| temp_vacuum | R (ro) | °C | Temperature vacuum side |
| **Special camera parameters and commands** |  |  |  |
| date | S |  | Date and time in the string format.<br>`YYYY-MM-DDThh:mm:ss` |
| Name | S |  | Camera name |
| CCDName | S |  | Name (type) of CCD sensor |
| CCDSerieNr | S |  | Serial number of CCD sensor |
| halt | C |  | Camera shutdown |
| reboot | C |  | Camera reboot |

## Temperature control of the ZIMPOL sensor

A thermometric cooling (Peltier elements) is used for the ZIMPOL image sensor and in addition a secondary cooler is needed to cool the Peltier elements.

At startup of the camera the cooling is off and no temperature information is actively sent by the camera. The two main parameters to control the temperature are `tr` for the set point and `tc_mode` to switch on and off the temperature control loop in the camera.

The parameter `tc_mode` accepts the integer values 0 to 4. The default value is 0 (off). Setting to 1 causes the camera to send the parameters `tc, tw` and `pp`, every second, e.g.:

```
tc=25.13
tw=19.49
pp=0.00
```

The most important information for the user is `tc` the current temperature of the image sensor. The values of `tw` and `pp` are normally not needed. The camera also sends these parameters every second with the `tc_mode` values of 1 to 4. For normal operation `tr` is set to the target temperature and `tc_mode` to 2, e.g.

```
tr=-15
tc_mode=2
```

As long as the temperature has not yet reached the set point the parameter `it_limited` is send in addition, a typical output looks like:

```
tc=25.02
tw=20.32
pp=0.07
tr_limited=24.56
```

If the set point (`tr`) is changed at any time later, it immediately starts to change to the new target temperature.

In case the secondary cooler fails the thermometric cooling would get out of control and put maximum power to the Peltier element. This could destroy the Peltier elements and in the worst case even the image sensor. Therefore, the camera automatically executes a safety control. The parameter `tw` is compared with the temperature warm side high limit (parameter `twhl`). If `tw` exceeds `twhl` the temperature control will immediately stop and the `tc_mode` parameter is set to the exception state 4. The client software should be able to react to this exception in case it receives `tc_mode=4` from the camera.

## Single image acquisition

In a first step the required value of the acquisition time in seconds may be sent with the `it` parameter, e.g.

```
it=1.5
```

A single image can then be taken by sending the command:

```
gi
```

A typical output of the camera will then be:

```
gi: busy ifn=0
ifn=0
img=<binary data value of the image>
gi: read image; result = 0000; ifn = 0
gi;
```

Important is the `gi;` at the end for a successful execution of the command or `gi: ERROR` in case of a failure. The other `gi:` status messages and the `ifn` parameter are for information only and can be ignored.

The `img` parameter contains the image data sent in the ZIMPOL binary format. A typical example looks like:

```
<SOH>u16[560,1280] it=1.50 date={2024-05-28T14:31:37} tc=-15.00 tw=18.12
seq_mode=0 seq_state=0 img_l=0 img_t=0<STX><data bytes><ETX>
```

The first part `u16` is always the same since the camera currently sends pixel data as unsigned 16-bit integer numbers. The next part contains the dimensions of the image in the example 560 rows and 1280 columns. These numbers depend on the settings of the image size which can also be inquired by the parameters `img_h` and `img_w`. A user change of the image size is currently not foreseen for the Brachium instrument and therefore will be the same too (but maybe different from the example). The next part contains some image meta data in the form of 8 parameters. These are all standard camera parameters (see command and parameter table for details).

## Performing measurements

For performing a measurement, it usually requires taking a series of images. The `gis` command should be used (the `gi` command could also be used with a loop on the client side but this would be less efficient because of a larger overhead.

The parameter `isl` is set to the number of images to be taken and the `gis` command is sent, e.g.

```
isl=3
gis
```

This example will create the following output from the camera:

```
gis: busy ifn=0
ifn=0
img=<binary data value of the image>
gis: read image; result = 0000; ifn = 0
gis: busy ifn=1
ifn=1
img=<binary data value of the image>
gis: read image; result = 0000; ifn = 1
gis: busy ifn=2
ifn=2
img=<binary data value of the image>
gis: read image; result = 0000; ifn = 2
gis;
```

It is very similar to the `gi` command as it is explained above.

For measurements the parameter `seq_mode` is very important. It sets the camera to a certain image sequence mode. For the Brachium instrument only modes 0 and 32 are relevant. In `seq_mode=0` all images are taken with the same camera settings. It is useful for single image acquisition or dark frame measurements. More important is `seq_mode=32` which is required for all polarimetric measurements. In this mode the camera uses two different camera modulator phase values defined in the first two elements of the `seq_phase` parameter. The two values are alternating changes between images. It has several consequences to be considered.

A complete measurement always requires an even number of images but the camera does not take care of this and will execute odd numbers too. The client software needs to handle user inputs in a correct way to ensure even numbers are taken.

The parameter `seq_state` included in the image meta data will have alternating values of 0 and 1 that indicates which value of the phase parameter has been used to take the image. This information is very important for data processing later and therefore must not get lost when the image is saved. The camera will take images with alternating phase values but it does not always start with the same value. It can be either the first or the second value depending on how the previous image series has ended (also using the `gi` command in sequence mode 32 will alternating change the state).

As mentioned above the image sequence mode 0 is normally used to take single images or dark frame measurements. This would mean to set the `seq_mode` parameter to the correct value before sending the `gi` or `gis` command. But on the other hand, these two tasks could also be performed in mode 32 without a significant side effect. It could simplify the camera control process. Therefore, the recommendation is to set `seq_mode=32` as default in the camera and the client software is not changing it at any time.

## Aborting image acquisitions and measurements

To abort a single image acquisition the `abort` command needs to be sent. This can also be used to abort a measurement started with the `gis` command.

For measurements an alternative method is to change the `isl` value to the number that is equal to the current frame number (or smaller but do not set it to 0 because this may cause an endless series). This is somehow a cleaner way to abort a measurement.

Changing the `isl` value during a running measurement can also be used to shorten or extend the measurement length. This is currently not foreseen for the Brachium instrument.

## Shutdown procedure

For a clean camera shutdown and power-off a controlled increase of the image sensor temperature is preferable because fast changes of the sensor temperature may decrease the lifetime of the sensor. Such a procedure could be the following.

1. Set the sensor to an ambient temperature, e.g. `tr=20`
2. Monitor the `tc` value and in addition the Peltier power `pp` value.
3. If both `tc` and `pp` are in the save range send the `shutdown` command
4. After a certain delay power-off the camera