

Tescan Scanning Electron Microscope

SharkSEM Remote Control

Revision 2.0.2
13th July 2010

SEM Models: Vega 3, Mira 3, Vela 3, Mira 3

TESCAN, a.s.
Brno, Czech Republic



Copyright 2010, TESCANA, a.s.
Libušina tř. 21, Brno, Czech Republic
Phone +420 547 130 411
info@tescan.cz

Table of Contents

Table of Contents	3
Preface	6
Overview	7
TCP/IP Protocol	7
Concurrent Clients	8
Functions	8
Protocol design	8
Command Processing	10
Single Command	10
Single Queue	11
Multiple Queues	12
Internal Command Processing	12
Message Structure	14
Scalar Data Types	14
Arrays	14
Compound Types	14
Storage Rules	15
Message Header	15
Message body	17
Session Management	20
Communication	22
TcpGetVersion	22
TcpRegDataPort	23
Electron Optics	25
AutoColumn	25
AutoGun	26
AutoWD	26
Degauss	27
EnumCenterings	27
EnumGeometries	28
EnumPCIndexes	29
Get3DBeam	30
GetCentering	31
GetGeometry	32
GetIAbsorbed	32
GetImageShift	33
GetPCContinual	33
GetPCFine	34
GetPCIndex	34
GetSpotSize	35
GetViewField	36
GetWD	36
Set3DBeam	37
SetCentering	37
SetGeometry	38
SetImageShift	39

SetPCContinual	39
SetPCFine	40
SetPCIndex	41
SetViewField	41
SetWD	42
Manipulators.....	43
ManipGetConfig.....	43
ManipGetCount	43
ManipGetCurr	44
ManipSetCurr	45
Stage Control	46
StgCalibrate	46
StgGetPosition.....	46
StgIsBusy	47
StgIsCalibrated	47
StgMove	48
StgMoveTo	49
StgStop	49
Input Channels and Detectors.....	51
DtAutoSignal	51
DtEnable	51
DtEnumDetectors	52
DtGetChannels	53
DtGetEnabled	54
DtGetGainBlack	54
DtGetSelected.....	55
DtSelect	55
DtSetGainBlack.....	56
Scanning	58
ScEnumSpeeds	58
ScGetBlanker	58
ScGetExternal.....	59
ScGetSpeed	60
ScScanBitmask	60
ScScanLine	61
ScScanLitho	62
ScScanXY	62
ScSetBeamPos	64
ScSetBlanker	65
ScSetExternal	65
ScSetSpeed	66
ScStopScan	66
Scanning Mode.....	68
SMEnumModes	68
SMGetMode	68
SMSetMode.....	69
Vacuum	71
VacGetPressure	71
VacGetStatus	71
VacGetVPMMode	72

VacGetVPPress	73
VacPump	73
VacSetVPMode	74
VacSetVPPress	74
VacVent	75
Airlock	77
ArlCloseValve	77
ArlGetStatus	77
ArlOpenValve	78
ArlPump	79
ArlVent	79
High Voltage	81
HVAutoHeat	81
HVBeamOff	82
HVBeamOn	82
HVEnumIndexes	83
HVGetBeam	84
HVGetEmission	85
HVGetFilTime	85
HVGetHeating	86
HVGetIndex	87
HVGetVoltage	87
HVSetIndex	88
HVSetVoltage	89
Chamber Camera	90
CameraDisable	90
CameraEnable	90
CameraGetStatus	91
SEM GUI Control	93
GUIGetScanning	93
GUISetScanning	93
Miscellaneous	95
ChamberLed	95
Delay	95

Preface

All Tescan Scanning Electron Microscopes (SEMs) are equipped with remote control capability. It can be used by remote applications like an EDX system, or lithography system or other custom application.

The remote control interface is referred as SharkSEM Remote Control.

There are two protocol generations:

- 1.x.x – SEM models produced in the years 2005 – 2010 (G2)
- 2.x.x – SEM models produced in 2010 and later (G3)

The two protocol revisions are compatible. The main difference is that some functions are not supported in both generations, some other functions are slightly different.

Overview

The microscopes are controlled locally from dedicated PC using special applications provided by Tescan as a part of standard microscope delivery. However, there are many situations when SEM should be operated remotely (across internet, from a different computer, etc.).

Typical situations when remote control can be used:

- Servicing the device
- Interfacing with EDX, WDX, EBSD systems
- Non-standard extension applications
- Training

To enable remote applications to control the SEM, each modern Tescan SEM is equipped with SharkSEM Remote Control Interface. This is a purely software interface and it is a part of Vega, Mira, Lyra software, which is the main software controlling the microscope.

There are two electron beam gun types: **field emission** and **tungsten** (thermal emission). The Vega 3 system is tungsten based, all Mira 3 systems have field emission guns. Tungsten FIB is called Vela 3, field emission FIB is called Lyra 3.

To control the SEM remotely, it is necessary to write a special application. The application should strictly follow the rules declared in this document.

Very narrow subset of remote control functions is also available via special Windows DLL. The DLL is available on request in 32-bit or 64-bit form.

The communication protocol is improved continuously. Tescan always tries to keep the protocol backward compatible, however, 100% compatibility of different releases is not guaranteed.

TCP/IP Protocol

To transport commands and image data between the remote application and the SEM, TCP/IP protocol is used. Currently IPv4 only is supported.

It is necessary to establish one or two TCP/IP connections from remote system to the SEM, there should be direct network connectivity between the two computers. The connections are always established from client to well-known server ports, SEM behaves as a **server** and remote application behaves as a **client**. In the following text, “client” means any remote application and “server” means the SEM main control application.

If direct network connection between the two systems is not available, NAT can be used.

The first connection is called **control connection** and all commands and their return values are passed through this connection.

The second connection is optional. Its purpose is to transfer the image data, so it is called **data connection**. Image data is sent through an independent connection because it helps to minimize the communication latency over the control connection.

Concurrent Clients

The SEM server can serve several clients connected at the same time. To prevent interference among the clients connected, they must synchronize write access to the microscope. If a client wants to change any microscope parameter, it should enter “exclusive mode” of operation first. If the exclusive lock is acquired by a client, the other clients are not allowed to change any microscope parameter.

It is also necessary to acquire the exclusive lock if a client wants to control the image scanning, because there is only one image scanning and acquisition unit.

Functions

Basically there are two sets of functions:

- Core functions
- Extension functions

Core functions are a standard part of each SEM. These functions are described in this document.

Sample functions:

- Set working distance
- Set brightness and contrast
- Move the stage
- Set probe current

Extension functions are available only if the microscope is equipped with an extra hardware parts. These functions are not covered by this document.

For example, FIB Lyra systems are equipped with focused ion beam column. The ion column can be controlled as well, using a set of commands described in the *SharkRemote-1-0-Fib.pdf* document.

Number of functions can be virtually unlimited. Each function is identified by a unique self-explaining string, allowing easy debugging of the client application.

Protocol design

Elementary piece of information sent between the client and the server is called **message**. Sometimes it is referred as **function** or **command**.

Each message consists of **header** and **body**. Header size is fixed, body size is variable. Header contains several mandatory fields like name of the function, body size, control flags, etc. In the body, there is encoded list of arguments passed to the function.

The server does not send any piece of information unless asked by the client. It means that currently there is no notification engine which would tell the client that some event has occurred.

Command Processing

There are several different approaches how the client can be designed in terms of the client – server interface. There is only one execution unit inside the server, but these programming models allow user to design the client application in a simple and straightforward way or a complex powerful way, depending on requirements.

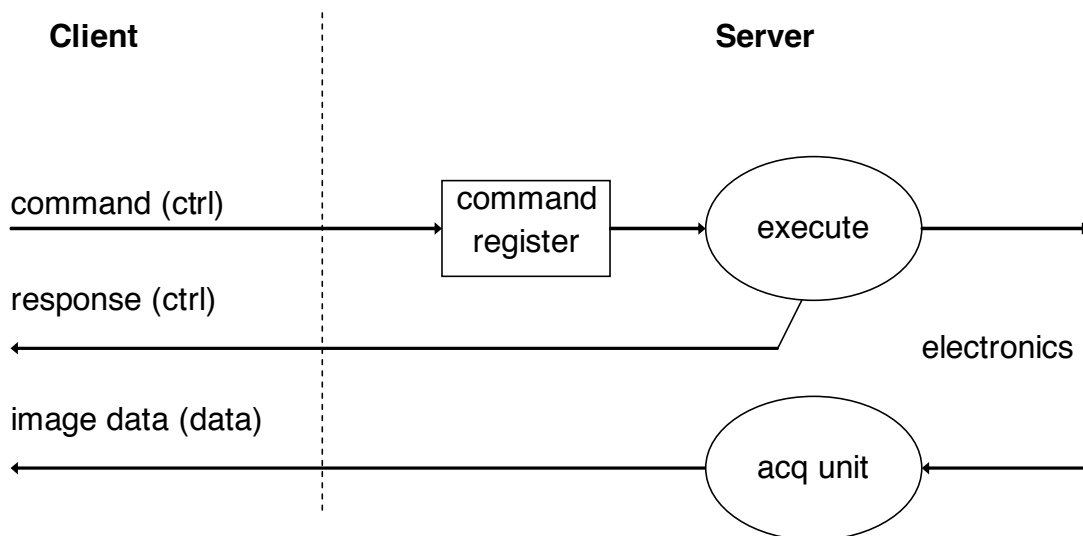
Three models are described:

- Single command
- Single queue
- Multiple queue

Note: for most applications, “single command” mode is the best choice.

Single Command

There is either none or just one command issued by the client at the same time. The server is either idle or processing the only one active command.



There are two TCP/IP connections - bidirectional control connection marked **ctrl** and unidirectional connection marked **data**.

”Send response” flag in the command header is required. See below for detailed description of the flags.

Next command must not be issued until the response from the first command is received.

It is impossible to cancel the command once sent.

The Single Command programming model is very easy to understand and it is not complicated to program the client application. Command execution is fully synchronous, that is, commands are executed one after another.

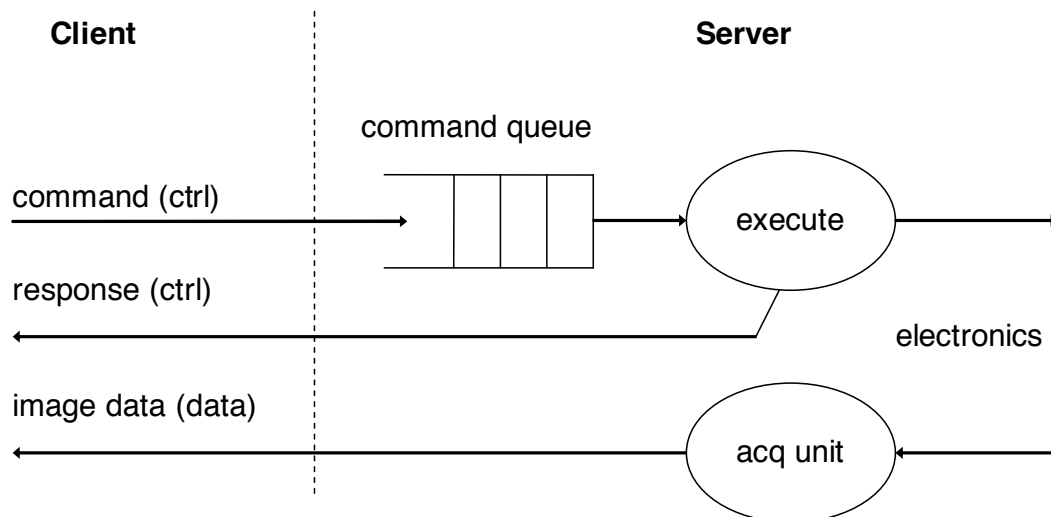
Image acquisition can be done even using this simple model. There are commands for starting and stopping the image scanning. “Start Scanning” command is executed immediately when it is received by the server. As a result, server starts to send continuous series of messages through the data channel. These messages contain image data in their body.

In the meantime (while the scanning is active), other commands may be sent to the server. This allows the client to adjust the conditions under which image is acquired during the acquisition.

Single Queue

This is more powerful solution. It allows the client application to minimize the communication latency.

There is a server-side FIFO command queue. It is possible to issue several command and store them in the server queue. The commands are processed by the server one after another. When a command is executed, response can be sent to let the client know which command has just been finished.



The client must be able to distinguish which command has been executed. To allow that, there is an optional 32-bit value which is sent back in response and which can identify the command.

It is also possible to remove the commands from the queue.

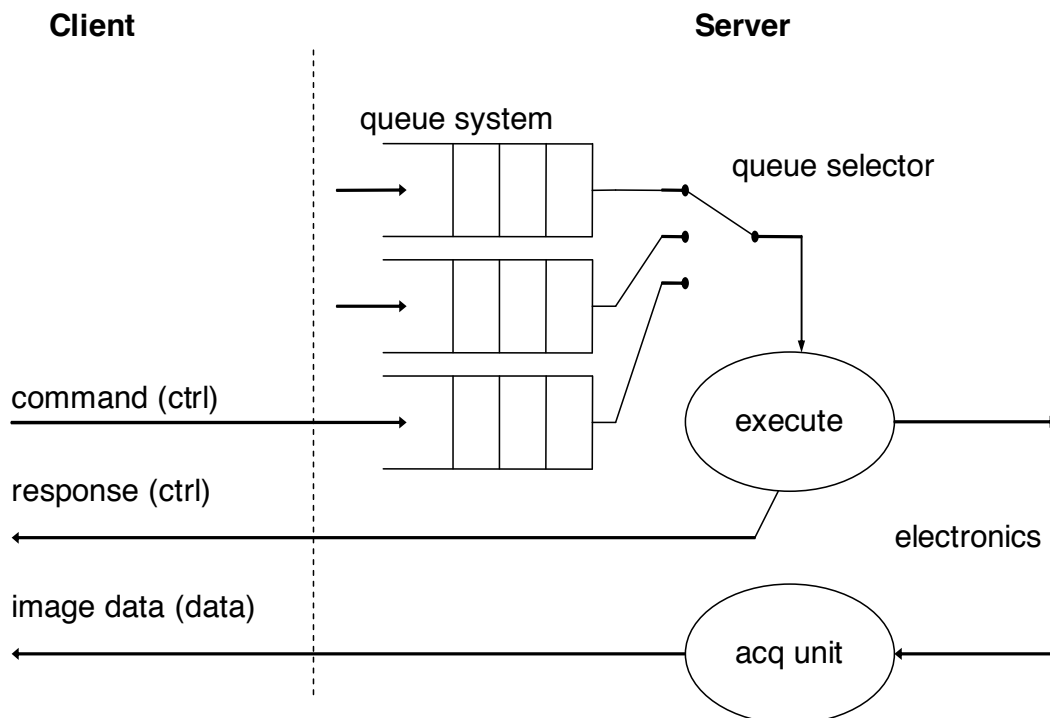
”Send response” flag in the command header is optional.

When command is issued, it is possible to either put it in the end of the queue or to put it into the head of the queue. This is useful if several commands in the queue shall be removed, in this case it is necessary to jump the queue. Another example could be polling the status of the microscope while there is a series of commands pending in the queue.

Multiple Queues

This is similar to the Single Queue model. There are several queues, each of them containing several commands.

All the Single Queue features are also valid for Multiple Queue model. Moreover, it is possible to manage the set of queues, change the active queue, etc. The active queue is selected by a special command.



To be clarified: how the active queue is switched. Next release of this documentation will address this issue.

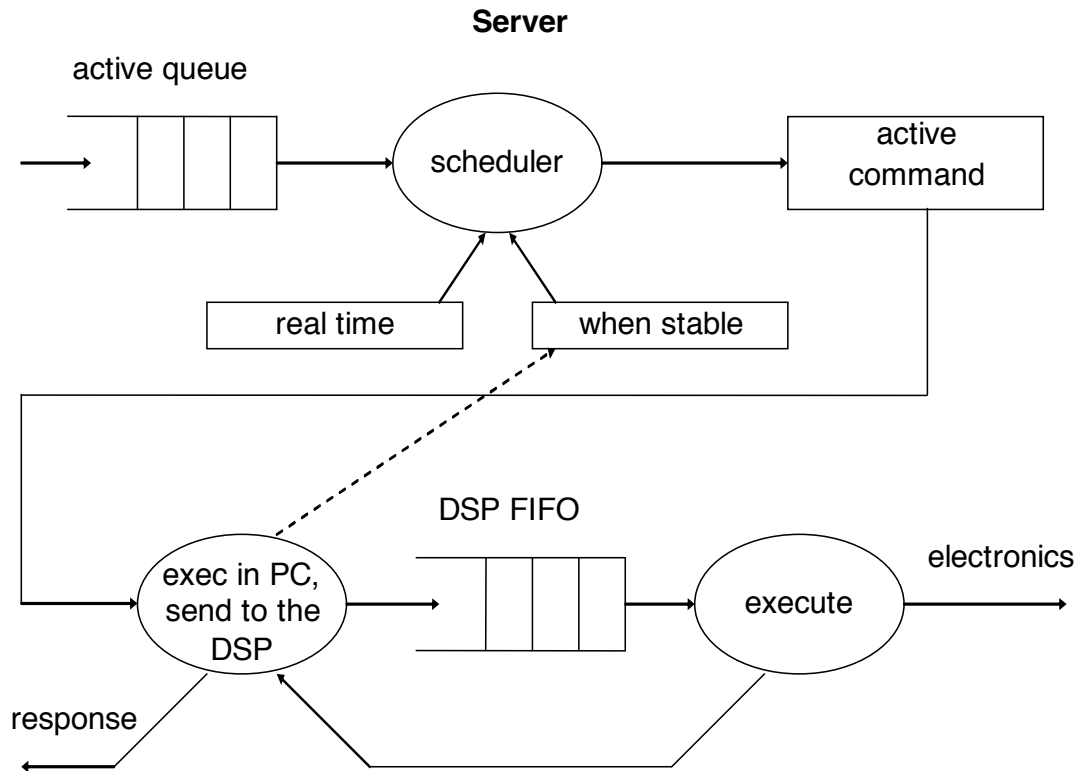
Internal Command Processing

In fact, the execution of command is not as simple as described above. There is a scheduler that controls whether the input command can be scheduled for execution or not.

The scheduler is controlled by “Wait” command flags. These tell the microscope whether it has to wait until all transient effects have disappeared. For example, if working distance is

changed, it takes an arbitrary time until the lens focus is stabilized. If image acquisition command is scheduled for execution after such a slow function, it can be marked as “Wait” command so that it is guaranteed that the system is stable at the moment of execution.

Currently there are four SEM Wait flags – Wait A, Wait B, Wait C and Wait D. FIB systems define three additional Wait flags – Wait E, Wait F, Wait G. Different waiting condition is related with each of them. See *Message Header* for details.



Since there is a DSP controller inside the microscope, the execution of the command is done in two phases. First, it is processed by the server on the PC. As a result, the command is split into several elementary commands for the DSP and then processed. For example, if working distance is set, it is necessary to compute the focus of several lenses in the optics and then send several commands to the DSP board. Finally, the DSP board will process the commands in a serial manner.

Client is not able to influence the commands already scheduled for execution.

Message Structure

In this chapter, we describe the layout of the command header and body.

Scalar Data Types

There are six fundamental data types which can appear in the message header and message body. The types are similar to C language data types.

char

This is an 8-bit wide signed integer. Its range is -128 to 127.

unsigned char

This is an 8-bit wide unsigned integer. Its range is 0 to +255.

short

This is a 16-bit wide signed integer. Its range is -32768 to 32767.

unsigned short

This is a 16-bit wide unsigned integer. Its range is 0 to 65535.

int

This is a 32-bit wide signed integer. Its range is -2147483648 to 2147483647.

unsigned int

This is a 32-bit wide unsigned integer. Its range is 0 to 4294967295.

void

This not a real type, it simply says that there is nothing.

Arrays

All six scalar types can be stored in a variable sized array. For example, **char[]** is an array of characters of variable length. The array size is unlimited, the actual size is sent together with the data. See below for details.

Sometimes **string** is mentioned in the documentation. **String** is a **char[]** array, which contains ASCII characters ≥ 1 and ≤ 255 , the last character is always zero.

Compound Types

There are currently two compound types – **float** and **map**.

float

Floating point type is implemented using zero terminated **char[]**, that is, **string**. **Float** values are sent as a string, for example “3.1415e-3” is a valid floating point number. The syntax is equal to syntax recognized by C-language *strtod()* function.

map

Map is an unordered set of (*key*, *value*) pairs. It is encoded as a **string**, each pair on a separate line. The *key* and the *value* are separated by '=' character. *Key* consists of letters, digits, underscores and dots. *Value* may consist of any ASCII characters ≥ 32 . *Key* should be considered as a unique case-sensitive identifier.

Example:

```
speed.1=1.15
speed.2=4.0
speed.3=12
```

The **map** type is very flexible. On the other hand, it is rather complex. It is used mostly for configuration purposes.

Storage Rules

If a string is used, it is always zero terminated.

Storage method is little-endian, that is, least significant byte is stored at the lowest address.

Inside the body of the message, data alignment takes place as described below. The header does not need any alignment.

Message Header

Header is the same for both request and response.

Request is a message sent by a client. *Response* is a message sent by the server.

```
struct tMessageHeader
{
    char          Cmd[16];           // :00
    unsigned int  BodySize;          // :16
    unsigned int  Id;                // :20
    unsigned short Flags;            // :24
    unsigned short Queue;            // :26
    char          Resvd[4];           // :28
};
```

Address	Type	Size in bytes	Value
00	char[]	16	Command name
16	unsigned int	4	Body size
20	unsigned int	4	Identification
24	unsigned short	2	Flags
26	unsigned short	2	Queue
28	char[]	4	Reserved

Total size of the header is 32 bytes.

Command name

Unique command name. Command name is a case sensitive, null terminated string. Maximum number of characters is 15, the last character must be followed by 0x00 null terminator. The command name is the same for both request and response.

Body size

Size of the message body in bytes, including the padding (if any).

Identification

This is an optional identification of the command. Server copies the Id from request to response. The Id is also required when message is removed from a queue, the delete call requires Id array as an argument.

Flags

There are several flags that control the message handling.

Request flags

Bit	Value	
0	Send response	
1	Jump	
2-7	Reserved, always 0	
8	Wait (SEM)	A – e-beam scanning
9		B – stage
10		C – e-beam optics
11		D – e-beam automatic procedure
12	Wait (FIB)	E – i-beam scanning
13		F – i-beam optics
14		G – i-beam automatic procedure
15	Reserved, always 0	

Response flags

Bit	Value
0-15	Reserved, always 0

Send response – if set, response is always generated when command is executed. If not set, response is generated only if there is a return value or output parameter.

Jump – if set, the command is put into the head of the FIFO queue. If not set, the command is queued normally, at the end of the FIFO queue.

Wait flags are combined using logical OR. If Wait B and Wait C are set, command is executed when stage has stopped and the electron optics is stable.

Wait A – e-beam scanning – if set, the command is executed only if electron beam scanning is inactive. Usually used with *ScStopScan()* call in the single = 1 mode.

Wait B – stage – if set, the command is executed only if stage is idle.

Wait C – e-beam optics – if set, the command is executed only if electron optics is in stable condition.

Wait D – e-beam automatic procedure – if set, the command is executed only if no electron beam automatic procedure is in progress.

Wait E – i-beam scanning – if set, the command is executed only if ion beam scanning is inactive.

Wait F – i-beam optics – if set, the command is executed only if ion optics is in stable condition.

Wait G – i-beam automatic procedure – if set, the command is executed only if no ion beam automatic procedure is in progress.

All reserved bits must be set to zero.

Queue

If no queue or single queue is used, it must be set to zero (queue 0).

If multiple queues are used, this field contains identification of the queue where the command will reside.

Client must always define the queue. If a response is sent, the queue identification is set to the same value as in the request.

Reserved

Set to zero in both request and response.

Message body

Each message is defined by its *function prototype*.

Example

void GetWD(out float wd)

This says: *GetWD* function has no input arguments, no return value and one string as an output value. That is, 32 bytes (header only) are sent in the request and $32 + 4 + N + \text{padding}$ bytes are sent as a response. 4 is the size of a field which contains size of the string, N is number of characters in the string including the terminating null character, whole message is padded to fit the argument size into multiple of four (4-padded).

Return value

Return value is handled the same way as if it was the first output argument.

Input / output type modifier

Each argument is prefixed with modifier which tells the way argument is passed.

in – sent from client to server

out – sent from server to client

inout – sent both ways

Argument order

Arguments are encoded from left to right, starting with the first parameter if request is sent and with return value if response is sent. First, header is sent, then, all arguments are encoded into the message body one after another.

Argument encoding

If argument of an integral type (fundamental type) is used, it is encoded into the body and 4-padded. That is, integral types always occupy 4 bytes in the message body.

If the argument is an array, first *unsigned int* contains number of bytes in it. This is followed by the array itself, the array is 4-padded at the end. There might be even empty array, zero is valid number of elements. Elements within the array are not padded.

Example

Let's say there is an array of three 16-bit short values. The array is encoded into 12 bytes. First, there is the 32-bit wide array size (**S**). Immediately follows the array itself (elements **A**, **B**, **C**). Since the total size is 10 bytes, 2 bytes of padding are added.

S0	S1	S2	S3	A0	A1	B0	B1	C0	C1	-	-
0				4				8			12

Example

```
void DtGetGainBlack(in int detector, out float gain, out float black);
```

Message body sent by the **client**:

Address	Type	Size in bytes	Value
00	Int	4	detector

The size of the message body is 4 bytes.

Message body sent by the **server**:

Address	Type	Size in bytes	Value
00	unsigned int	4	gain (size = 5)
04	char[]	5	gain ("3.14")
09	char[]	3	gain (padding)
12	unsigned int	4	black (size = 4)
16	char[]	4	black ("0.0")
20	char[]	0	black (padding)

The size of the message body is 20 bytes.

Session Management

First of all, control connection between the client and the server is established. Client must initiate the process by calling `TCP connect()` function. If the server is ready, connection request is accepted immediately.

The default control connection server port is 8300. This can be changed if the port is already used by another application.

Once connected, client may want to check the version of Remote Control Interface by calling `GetVersion()` function. If it is not compatible with the version reported, it must close the connection and notify the operator.

Then, client may want to establish data connection. This is an optional step, if the client does not need to read image data from the server, it may skip it. The data connection **server port** is control connection server port + 1, which is 8301 by default. The data connection **client port** number must be known at the server side prior to establishing the data connection. Server requires this value to be able to match the control and the data connection.

***Note:** this has been extended since 1.0.8. See “firewall-friendly mode” described below.*

A special function is provided to pass the client-side data connection port number to the server. Refer to ***TcpRegDataPort*** for details.

Server TCP ports (default setting)

Connection	Port
Control	8300
Data	8301

Then, authentication might be required. This is not supported yet.

Now, client is ready for microscope control. The connection may remain established as long as required. There is no limitation how the client is designed, it must use the functions according to the specification.

If the client calls unexpected function, or if the function arguments are wrong, or if the message is incorrect, it is silently ignored.

The client must read all the data sent by the server. If there is outgoing data pending and server is not able to send them to the client, the client is disconnected by the server. This does not happen immediately, there is approximately 10 s timeout. The exact time depends on the server send buffer size and amount of data produced by the server.

After the client has finished its operation, it must first close the data connection and then close the control connection. If there are some resources associated with the client connection (pending commands, locks acquired, etc.), they are automatically released when the control connection is closed.

Client must properly handle situation when connection is lost. The recommended behavior is to close the connections and tell the user that connection to the SEM was lost.

Without image transfer

Step	Client task
1	Establish control connection (port 8300)
2	Check interface version
3	Authentication
4	Control the SEM, there can be many commands
5	Close control connection

Image transfer required

Step	Client task
1	Establish control connection (port 8300)
2	Check interface version
3	Create data connection descriptor, bind to a local port
4	Call <code>TcpRegDataPort</code>
5	Establish data connection (port 8301)
6	Authentication
7	Control the SEM, there can be many commands
8	Close data connection
9	Close control connection

As described above, the data connection requires special procedure when setting up. Also note that it is important to wait until `TcpRegDataPort()` call is completed, the server needs to know the client data connection port number before the connection request is issued.

Refer to `socket()`, `connect()`, `bind()`, `getsockname()` socket functions for more information on how to the data connection.

Firewall-friendly mode

Since the above method of establishing data connection is often blocked by the firewalls, there is an alternative approach which is more reliable.

The main difference is that only single control + data connection is accepted by the server, and thus the server does not need to distinguish between the clients. In this case, `TcpRegDataPort()` function is ignored.

A good practice is still to use the `TcpRegDataPort()` function, and in case of complex network configurations to put the server into single client (firewall friendly) mode.

See `TcpRegDataPort()` for more info.

Communication

Following commands control the TCP communication channel between the client and the server.

TcpGetVersion

Read the protocol implementation version.

Arguments

char[] TcpGetVersion(void)

return value

version string

Timing

Executed immediately.

Remarks

Most basic SharkSEM functions are available in all protocol versions. However, the protocol has been growing and new functions appeared.

Beginning with version 1.0.5, it is possible to check the current protocol version by calling this function. If the client plans to call functions 1.0.5 or later, it is recommended to call *TcpGetVersion()* at the beginning of the session.

The version string consists of three decimal numbers separated by dots. Following version strings are valid examples:

“1.0.5”

“1.0.129”

“2.0.0”

Since the *TcpGetVersion()* was not present in the original version of the protocol, the servers 1.0.4 and older do not respond, because the function is not implemented. The client must properly handle the response timeout.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	yes

TcpRegDataPort

Set the originating (source) TCP port for data connection.

Arguments

int **TcpRegDataPort**(*in unsigned int port*)

port	client TCP port number of the data connection
------	-----------------------------------------------

return value	0
--------------	---

Timing

Executed immediately.

Remarks

First, control connection is established. If the client wants to transfer an image, it must also establish TCP data connection.

When the client tries to establish the data connection, the server must be able to match the data connection with the control connection. There are two ways how to achieve that.

Classic approach - registering the source TCP port

Prior to establishing the data connection, server must know the data connection originating (client) port. The port number is transferred from client to server using this function.

The procedure is:

- Establish control connection (server port 8300)
- Bind data connection to local port
- Read back the local port number
- Call *TcpRegDataPort()*
- Establish data connection

The return value is provided to the client to make sure that the server has processed the request and the client may try to establish the data connection.

Firewall-friendly approach

The classic approach has one significant disadvantage. It can hardly pass the firewalls, because most firewalls alter the connection source port.

Since version 1.0.8, the server can be configured the way that it accepts just single connection (or better to say, single pair of control connection and data connection). If the server is configured this way, it is not necessary to call the *TcpRegDataPort()* anymore, because the server matches the data connection and the control connection automatically (because there is only one control connection).

The advantage is that such method easily traverses firewalls, but, on the other hand, it is not possible to use more than one client at the same time.

See also *Session Management* article.

Call Context

Must be called before establishing the data connection.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Electron Optics

This set of functions controls how the electron optics is set up. The most important parameters are scanning mode, magnification, probe current and working distance.

AutoColumn

Start automatic column alignment procedure. Depending on the scanning mode, OBJ Centering and IML Centering are adjusted.

Arguments

void AutoColumn(in int channel)

channel input video channel

Timing

Execution time may vary between 10 and 60 seconds (wait D).

Remarks

Channel index is a zero based index of the input channel. Appropriate detector must be selected as an active.

There are several important preconditions. If they are not fulfilled, the column centering may fail.

- Good signal quality
- Good structure on the specimen surface
- Well focused image

As a result, objective centering value is changed. Unfortunately, it is not possible to automatically verify the quality of the automatic alignment result.

See also *DtSelect()*, *AutoWD()*, *AutoSignal()*.

Call Context

Scanning must be inactive.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

AutoGun

Start automatic gun alignment procedure.

Arguments

*void **AutoGun**(in int channel)*

channel

input video channel

Timing

Execution time may vary between 10 and 30 seconds (wait D).

Remarks

Channel index is a zero based index of the input channel. Appropriate detector must be selected as an active.

Especially the tungsten filament is worn during the SEM operation. It may be useful to re-center the gun if the probe current decreases.

As a result, the electronic gun shift and the electronic gun tilt values are changed.

See also *DtSelect()*, *AutoSignal()*.

Call Context

Scanning must be inactive.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

AutoWD

Start automatic WD adjustment procedure.

Arguments

*void **AutoWD**(in int channel)*

channel

input video channel

Timing

Execution time may vary between 10 and 60 seconds (wait D).

Remarks

Channel index is a zero based index of the input channel. Appropriate detector must be selected as an active first, the automatic procedure requires good signal quality and suitable specimen to achieve good result.

As a result, WD value is changed.

See also *DtSelect()*.

Call Context

Scanning must be inactive.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Degauss

Degauss the microscope column.

Arguments

void Degauss(void)

Timing

Synchronous execution. No command is executed until *Degauss()* is completed. Takes several seconds to finish.

Remarks

During the degauss procedure, all electromagnetic coils in the microscope column are fed with alternating current with decreasing amplitude. The aim is to minimize the magnetic remanence.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

EnumCenterings

Get list of all column centering parameters.

Arguments

map EnumCenterings(void)

return value

list of centering parameters

Timing

Executed immediately.

Remarks

The *map* has the following form:

```
cen.0.name  
cen.0.count  
cen.0.unit  
cen.1.name  
...  
cen.X.name  
cen.X.count  
cen.X.unit
```

The key is separated by two dots. The middle number is an index of the centering parameter. Application should not expect that the indexes form a consecutive sequence. The index can be any integer number.

name	human-readable name.
count	1 or 2 (one if only the first value is used, two otherwise)
unit	%, mm, or other physical unit

The exact set of centering parameters depends on the SEM configuration.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

See also *GetCentering()*, *SetCentering()*

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

EnumGeometries

Get list of all available scanning transformations.

Arguments

map EnumGeometries(void)

return value

list of geometric transformations

Timing

Executed immediately.

Remarks

The *map* has the following form:

```
geom.0.name  
geom.0.count  
geom.0.unit  
geom.1.name  
...  
geom.X.name  
geom.X.count  
geom.X.unit
```

The key is separated by two dots. The middle number is an index of the parameter. Application should not expect that the indexes form a consecutive sequence. The index can be any integer number.

name	human-readable name.
count	1 or 2 (one if only the first value is used, two otherwise)
unit	%, mm, deg, or other physical unit

The exact set of centering parameters depends on the SEM configuration.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

See also *GetGeometry()*, *SetGeometry()*

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

EnumPCIndexes

Get list of all PC indexes.

Note: *PC* (*Probe Current*) is an old name for *BI* (*Beam Index*). For backward compatibility, remote control still uses the *PC* parameter. The *PC* and *BI* range is usually 1 to 20. The higher is *BI*, the higher is the electron beam current. *PC* uses inverse logic.

Arguments

map **EnumPCIndexes**(void)

return value

list of PC indexes

Timing

Executed immediately.

Remarks

The *map* has the following form:

```
pc.1.current  
pc.2.current  
...  
pc.N.current
```

1 is the minimum index, **N** is the maximum index. The list of indexes is a continuous sequence and the corresponding values are increasing or decreasing as well. Currently there are 20 indexes (1 to 20), but this may change in the future.

current is the estimated probe current in pA (picoamps). Application should not expect that the value is accurate, because it depends on the acceleration voltage, gun centering, and other parameters.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Get3DBeam

Get electronic beam tilt.

Arguments

void Get3DBeam(out float alpha, out float beta)

alpha

tilt angle along the X scanning axis [deg]

beta

tilt angle along the Y scanning axis [deg]

Timing

Executed immediately.

Remarks

3D Beam is the angle between the optical axis of the column and the beam scanning on the specimen surface.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

GetCentering

Get column centering parameter.

Arguments

void GetCentering(in int index, out float x, out float y)

index	index of the parameter – see <i>EnumCenterings()</i>
x, y	centering values in X and Y axis

Timing

Executed immediately.

Remarks

Note: it is usually not necessary to read/write the centering from the remote application.

This function returns one or two values (the **y** parameter may not be used in some cases).

The SEM column centering depends on:

- 1) accelerating voltage
- 2) scanning mode
- 3) active detector

Each accelerating voltage range (index) keeps its own centering set. Each scanning mode keeps its centering set as well. Finally, there are some detectors (In-beam) which have an effect on centering as well and must keep an extra set.

See also *EnumCenterings()*, *SetCentering()*

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

GetGeometry

Get image (scanning) geometry parameter.

Arguments

void GetGeometry(in int index, out float x, out float y)

index index of the parameter – see *EnumGeometries()*
x, y one or two values

Timing

Executed immediately.

Remarks

Note: it is usually not necessary to read or adjust the image geometry from the remote application.

This function returns one or two values (the **y** parameter may not be used in some cases).

Unlike the centering, the image geometry does not depend on the accelerating voltage or the scanning mode. There are several limitations, for example it is not possible to use dynamic focus in the FIELD mode.

See also *EnumGeometries()*, *SetGeometry()*

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

GetIAbsorbed

Read specimen current.

Arguments

float GetIAbsorbed(void)

return value

absorbed current in [pA]

Timing

Executed immediately.

Remarks

-

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

GetImageShift

Get electronic image shift.

Arguments

void GetImageShift(out float x, out float y)

x	x shift [mm]
y	y shift [mm]

Timing

Executed immediately.

Remarks

Image Shift is done in the scanning coils of the SEM.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

GetPCContinual

Get probe current (beam current).

Arguments

int **GetPCContinual**(void)

return value

PC value (usually 1.0 to 20.0)

Timing

Executed immediately.

Remarks

Refer to *SetPCContinual*().

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
no	2.0.2 and later	yes

GetPCFine

Get probe current fine adjustment. **This function is obsolete.**

Arguments

float **GetPCFine**(void)

return value

PC fine value [%]

Timing

Executed immediately.

Remarks

See *SetPCFine*().

Call Context

Anytime

Compatibility

Provided for backward compatibility only. Please use *GetPCContinual*() instead.

1.x.x	2.x.x	DLL
yes	yes - obsolete	no

GetPCIndex

Get probe current index.

Arguments

int **GetPCIndex**(void)

return value

PC index

Timing

Executed immediately.

Remarks

Refer to *SetPCIndex*(.).

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

GetSpotSize

Get electron beam spot size.

Arguments

float **GetSpotSize**(void)

return value

spot size in [nm]

Timing

Executed immediately.

Remarks

The electron beam spot size is a calculated value. It depends on the scanning mode, working distance, probe current and other parameters. The spot size is twice the microscope resolution (under given conditions).

If the spot size is exactly zero (0.0), it is not defined. This will happen when the spot size cannot be calculated.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
1.0.8 and later	yes	no

GetViewField

Get current view field.

Arguments

float **GetViewField**(void)

return value

view field in [mm]

Timing

Executed immediately.

Remarks

View field is the width of scanning window. This is especially important if scanning window shape is different than square.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

GetWD

Get current working distance.

Arguments

float **GetWD**(void)

return value

working distance in [mm]

Timing

Executed immediately.

Remarks

Working Distance is a distance between the end of the objective and the surface of the observed specimen. If *Working Distance* is set properly, image should be sharp.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

Set3DBeam

Set electronic beam tilt.

Arguments

void Set3DBeam(in float alpha, in float beta)

alpha	tilt angle along the X scanning axis [deg]
beta	tilt angle along the Y scanning axis [deg]

Timing

Executed immediately.

Remarks

3D Beam is the angle between the optical axis of the column and the beam scanning on the specimen surface. It is usually only a small value (several degrees). The shorter the working distance is, the higher the beam tilt range is. If the required value is too high, it is silently limited.

Some detectors (eg. In-beam) may reset the beam tilt value when selected. Make sure that the *DtSelect()* is called before *Set3DBeam()* function.

Call Context

Available in the *Resolution* or *Depth* scanning modes only.

Also Affected

Setting the 3D Tilt resets the image shift to zero.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

SetCentering

Set column centering parameter.

Arguments

void SetCentering(in int index, in float x, in float y)

index	index of the parameter – see <i>EnumCenterings()</i>
x, y	centering values in X and Y axis

Timing

Typically 5 ms to finish (wait C).

Remarks

Note: it is usually not necessary to read or adjust the centering from the remote application.

If the values are out of range, they are limited.

See also *EnumCenterings()*, *GetCentering()*

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

SetGeometry

Set image (scanning) geometry parameter.

Arguments

void SetGeometry(in int index, in float x, in float y)

index index of the parameter – see *EnumGeometries()*
x, y geometry values

Timing

Typically 5 ms to finish (wait C).

Remarks

Note: it is usually not necessary to read or adjust the image geometry from the remote application.

If the values are out of range, they are limited.

See also *EnumGeometries()*, *GetGeometry()*

Call Context

Anytime

Also Affected

Some parameters are related, for example Image Shift will change the 3D Beam value.

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

SetImageShift

Set electronic image shift.

Arguments

void SetImageShift(in float x, in float y)

x	x shift [mm]
y	y shift [mm]

Timing

Executed immediately.

Remarks

Image Shift is done in the scanning coils of the SEM. The maximum range depends on the scanning mode and other conditions, and thus it is not guaranteed. If the shift value is too high, it is silently limited. It is recommended to read back the image shift value after the change and check whether the required value was accepted.

Call Context

Anytime

Also Affected

Setting the image shift resets the 3D Tilt value to zero.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

SetPCContinual

Set probe current (beam current).

Arguments

void SetPCContinual(in float pc)

pc	PC value (usually 1.0 to 20.0)
----	--------------------------------

Timing

Several hundreds of milliseconds until the system becomes stable (wait C).

Remarks

PC Continual is an extension of PC Index concept. Instead of integer value, floating point value capable of continuous beam current control is accessible. The value is typically between 1.0 and 20.0, 1.0 is the highest current and 20.0 is the lowest one. More information can be received by calling *EnumPCIndexes()* function.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
no	2.0.2 and later	yes

SetPCFine

Set probe current fine adjustment. **This function is obsolete.**

Arguments

void SetPCFine(in float pcfine)

pcfine PC fine [%]

Timing

Several hundreds of milliseconds until the system becomes stable (wait C).

Remarks

Probe Current Fine is a value in the range -5 % to + 5.0 %. If the value is equal to zero percent, the beam current is defined by the current PC index. If the value is greater than zero, the beam current grows higher, if it is less than zero, the probe current is lower. The function may not be linear in the whole +/- 5.0 range.

The *Probe Current Fine* value is associated with each PC index. The Get / Set functions control the *PC Fine* for the currently selected index only.

PC Fine should be used only for special applications (eg. EDS/WDS probe current tuning), for general probe current control *SetPCIndex()* should be used.

Call Context

Anytime

Compatibility

Removed in version 2.0.2 and later. Please use *SetPCContinual()* instead.

1.x.x	2.x.x	DLL
yes	2.0.1 and older	no

SetPCIndex

Set probe current index.

Arguments

void SetPCIndex(in int index)

index

PC index

Timing

Several hundreds of milliseconds until the system becomes stable (wait C).

Remarks

Probe Current Index is an index into the microscope PC table. Typically the value is between 1 and 20, 1 is the highest current and 20 is the lowest one. More information can be received by calling *EnumPCIndexes()* function.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

SetViewField

Set view field.

Arguments

void SetViewField(in float vf)

vf

view field in [mm]

Timing

Typically 5 ms to finish (wait C).

Remarks

View Field is subject to limitation. If the required value is out of the limits, it is automatically adjusted.

View Field is a replacement of traditional *Magnification* function. Since *Magnification* is related to the computer display, remote control works rather with *View Field*.

Call Context

Not available in *Fish Eye* scanning mode.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

SetWD

Set working distance.

Arguments

void SetWD(in float wd)

wd working distance in [mm]

Timing

Several hundreds of milliseconds until the system becomes stable (wait C).

Remarks

Working Distance is subject to limitation. If the required value is out of the limits, it is automatically adjusted.

Call Context

Not available in *Fish Eye* scanning mode.

Also Affected

View Field may slightly change when *Working Distance* is modified.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

Manipulators

There can be one or more manipulators installed in the SEM. In most cases, just single stage for specimen manipulation is available.

ManipGetConfig

Get manipulator configuration.

Arguments

map **ManipGetConfig**(*in int index*)

index	index of the manipulator – see <i>ManipGetCount()</i>
return value	manipulator description

Timing

Executed immediately.

Remarks

The *index* is zero based.

The *map* contains following items:
 `manipulator.index`
 `manipulator.name`

The **name** is human-readable string.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.10 and later	2.0.1 and later	no

ManipGetCount

Get number of installed manipulators.

Arguments

int ManipGetCount(void)

return value 1, 2, ... number of manipulators

Timing

Executed immediately.

Remarks

Number of available manipulators is returned. See *ManipGetConfig()* for more info about specific manipulator. On most devices, the result is 1.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.10 and later	2.0.1 and later	no

ManipGetCurr

Get currently selected manipulator.

Arguments

int ManipGetCurr(void)

return value 0, 1, ... currently selected manipulator

Timing

Executed immediately.

Remarks

Returns index of the currently selected manipulator.

See *ManipSetCurr()* for details.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.10 and later	2.0.1 and later	no

ManipSetCurr

Select manipulator.

Arguments

void ManipSetCurr(in int index)

index index of the manipulator

Timing

Executed immediately.

Remarks

Defines which manipulator becomes selected. All subsequent stage *StgXxx()* commands refer to this manipulator.

After microscope initialization, manipulator 0 is automatically selected.

See also *ManipGetCurr()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.10 and later	2.0.1 and later	no

Stage Control

Most microscopes are equipped with motorized stage. Stage can be fully controlled via remote control interface, there are only limitations related to the stage model. For example, some microscopes do not support motorized movement in Z axis.

StgCalibrate

Calibrate the stage.

Arguments

void StgCalibrate(void)

Timing

The calibration takes up to one minute, depending on the stage configuration. Wait B flag is set while the calibration is in progress.

Remarks

If the calibration is successful, *StgIsCalibrated()* returns 1 after Wait B reset.

Call Context

Stage must not be busy.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

StgGetPosition

Get current stage position.

Arguments

void StgGetPosition(out float x, out float y, out float z, out float rotation, out float tilt)

x	absolute x coordinate [mm]
y	absolute y coordinate [mm]
z	absolute z coordinate [mm]
rotation	absolute stage rotation [degrees]
tilt	absolute stage tilt [degrees]

Timing

Executed immediately.

Remarks

If the stage is busy (movement is in progress), current position is updated approximately every 500 ms.

If an axis is not motorized, zero is returned.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

StgIsBusy

Determine if the stage is moving.

Arguments

int StgIsBusy(void)

return value

0 – stage idle, 1 – stage movement in progress

Timing

Executed immediately.

Remarks

-

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

StgIsCalibrated

Determine if the stage has been successfully calibrated.

Arguments

int StgIsCalibrated(void)

return value

0 – not calibrated
1 - calibrated

Timing

Executed immediately.

Remarks

-

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

StgMove

Start stage movement, specify velocity.

Arguments

void StgMove(in float vx[, in float vy[, in float vz[, in float vrotation[, in float vtilt]]]])

vx	x velocity [mm/s]
vy	y velocity [mm/s]
vz	z velocity [mm/s]
vrotation	rotation velocity [degrees/s]
vtilt	tilt velocity [degrees/s]

Timing

Executed immediately, Wait B flag is set during the movement.

Remarks

Infinite movement is started. Application must stop it later using *StgStop()* call.

Important: Stages without position indicators will run infinitely. If the stage is forced to move for a long time after it hits one of the mechanical limits, it can be damaged. When possible, use *StgMoveTo()* instead of this function.

If less than 5 arguments are passed, the other velocities are zero.

This function is supported by all types of stages.

Call Context

Stage must not be busy when *StgMove()* function is executed. If the stage is busy, the command is ignored.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.10 and later	yes	no

StgMoveTo

Move the stage to a position.

Arguments

void StgMoveTo(in float x[, in float y[, in float z[, in float rotation[, in float tilt]]]])

x	absolute x position [mm]
y	absolute y position [mm]
z	absolute z position [mm]
rotation	absolute rotation [degrees]
tilt	absolute tilt [degrees]

Timing

The execution time may vary. The time depends on the distance of the movement.
Wait B flag is set while running.

Remarks

The stage position is restricted in several ways, if an invalid position is required, it is silently limited into the allowed range.

If less than 5 arguments are passed, other positions are kept unchanged.

This function is supported only by stages equipped with position indicators.

Call Context

Stage must not be busy when *StgMoveTo()* function is executed. Stage must be calibrated. If the stage is busy or not calibrated, the command is ignored.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

StgStop

Stop the stage movement.

Arguments

void StgStop(void)

Timing

Executed immediately.

Remarks

If the stage is in motion, it is stopped immediately.

Call Context

Anytime. Make sure that this call does not wait for Wait B flag.

Compatibility

1.x.x	2.x.x	DLL
1.0.8 and later	yes	no

Input Channels and Detectors

Each microscope is equipped with at least one image input channel. Each input is assigned one of the detectors. This section describes how detectors are controlled and assigned to input channels.

DtAutoSignal

Invoke automatic brightness/contrast adjustment procedure.

Arguments

void DtAutoSignal(in int channel)

channel input video channel

Timing

Execution time may vary between 1 and 10 seconds (wait D).

Remarks

Channel index is a zero based index of the input channel. Brightness and contrast of the active detector is adjusted to get optimal signal.

If *DtAutoSignal()* function is called, appropriate detector must be selected as an active one first.

See also *DtSelect()*, *DtSetGainBlack()*, *DtGetGainBlack()*.

Call Context

Scanning must be inactive.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

DtEnable

Enable/disable input channel.

Arguments

void DtEnable(in int channel, in int enable[, in int bpp])

channel	input video channel index
enable	0 – disable, 1 – enable
bpp	bits per pixel (optional argument) 8 or 16, default is 8

Timing

Executed immediately.

Remarks

This function is called before scanning is started. If input is enabled, image data is sent to the client. If input is disabled, image data is ignored.

Example:

```

DtSelect(0, 1);           // select detector #1 into channel #0
DtEnable(0, 1);          // enable channel #0, 8 bpp
DtAutoSignal(0);         // adjust detector brightness/contrast
ScScanXY(...);           // start scanning

// process image data

ScStop(...);             // stop scanning

```

If 16 bits per pixel is selected, the *ScData()* callback has twice as much bytes in the data buffer than in 8-bit mode.

See also *DtGetEnabled()*.

Call Context

Scanning must be inactive when calling this function.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Protocol versions <= 1.0.5 do not have the bpp argument and use 8 bpp. Protocol versions >= 1.0.6 have optional bpp argument.

DtEnumDetectors

Get list of available detectors.

Arguments

map DtEnumDetectors(void)

return value

list of detectors

Timing

Executed immediately.

Remarks

Map of detectors is returned.

The *map* has the following form:

```
det.0.name
det.0.detector
det.1.name
det.1.detector
...
det.N-1.name
det.N-1.detector
```

N is number of installed detectors.

name is human-readable detector name.

detector is a numeric detector index provided in *Dt()* functions, eg. *DtSelect()*.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

DtGetChannels

Get number input video channels.

Arguments

int DtGetChannels(void)

return value

number of available input video channels

Timing

Executed immediately.

Remarks

When channel index is referred, the channel index must be less than the number of available channels.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

DtGetEnabled

Get input channel status (enabled / disabled, bpp).

Arguments

void DtGetEnabled(in int channel, out int enable, out int bpp)

channel	input video channel index
enable	0 – disabled, 1 – enabled
bpp	bits per pixel (8 or 16)

Timing

Executed immediately.

Remarks

See *DtEnable()* for details.

Call Context

-

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.11 and later	2.0.1 and later	no

DtGetGainBlack

Get detector gain and black level (contrast/brightness).

Arguments

void DtGetGainBlack(in int detector, out float gain, out float black)

detector	detector index
gain	gain (contrast) [percent]
black	black level (brightness) [percent]

Timing

Executed immediately.

Remarks

Detector index is a zero based detector index returned by *DtEnumDetectors()*.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

DtGetSelected

Get detector selected into a channel.

Arguments

void DtGetSelected(in int channel, out int detector)

channel
detector

input video channel index
detector index

Timing

Executed immediately.

Remarks

Returns index of the detector selected into the specified channel.

See also *DtEnumDetectors()*, *DtGetChannels()*, *DtSelect()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.11 and later	2.0.1 and later	yes

DtSelect

Select active detector.

Arguments

void DtSelect(in int channel, in int detector)

channel	input video channel index
detector	detector index

Timing

Executed immediately.

Remarks

There is an arbitrary detector assigned to each input channel. Since there is an analog cross switch, any detector can be assigned to any input channel.

See also *DtEnumDetectors()*, *DtGetChannels()*, *DtGetSelected()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

DtSetGainBlack

Set detector gain and black level.

Arguments

void DtSetGainBlack(in int detector, in float gain, in float black)

detector	detector index
gain	gain (contrast) [percent]
black	black level (brightness) [percent]

Timing

Execution time depends on the type of the detector. For standard SE and BSE detectors the typical time is 100 ms. Wait C is set while not ready.

Remarks

Detector index is a zero based detector index returned by *DtEnumDetectors()*.

Call Context

Anytime

Also Affected

Objective centering and 3D Beam can be changed if some detectors are selected. If 3D Beam is used, its value should be set up later, after *DtSelect()* is finished.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

Scanning

Scanning functions control how the electron beam scans over the specimen surface. The most important parameters are scanning rate, window size and scanning method. There are several scanning methods available – most frequently used XY Scan, then Line Scan, Mask Scan and others.

ScEnumSpeeds

Get list of available scanning speeds.

Arguments

map ScEnumSpeeds(void)

return value

scanning speed list

Timing

Executed immediately.

Remarks

Map of speeds is returned.

The map *key* has the following form:

`speed.1.dwell`

`speed.2.dwell`

...

`speed.N.dwell`

Where **N** is the total number of speed indexes.

The map *value* contains **float** value describing pixel dwell time in microseconds for given speed index.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScGetBlanker

Get current beam blanker mode.

Arguments

int ScGetBlanker(void)

return value

blanking mode

Timing

Executed immediately.

Remarks

There are three possible blanking modes:

0 – beam is always ON, that is, blanker is switched off.

1 – beam is always OFF.

2 – beam is ON only if the image acquisition unit requires it. In the XY scanning mode, beam is OFF during the line and the frame retrace. In the Litho mode, there is a separate flag for controlling the blanker. In the Bitmask mode, beam is ON only if corresponding bit in the bit mask is set to 1.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScGetExternal

Get external scanning status.

Arguments

int ScGetExternal(void)

return value

external scanning status

Timing

Executed immediately.

Remarks

External scanning can either be ON or OFF:

0 – external scanning is OFF.

1 – external scanning is ON.

If external scanning is ON, scanning amplifiers switch their inputs to external scanning ramp. External scanning is typically used by EDX systems.

Call Context
Anytime

Also Affected
-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScGetSpeed

Get current scanning speed index.

Arguments

int ScGetSpeed(void)

return value

scanning speed index

Timing

Executed immediately.

Remarks

Scanning speed is always valid, no matter if the scanning is active or stopped. Refer to *ScEnumSpeeds()* for details.

Call Context
Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScScanBitmask

This function has been removed due to the redesign of the scanning ramp generator.

Compatibility

1.x.x	2.x.x	DLL
yes	no	no

ScScanLine

Start scanning along a line. Beginning of the line, end of the line, pixel dwell time and number of pixels are provided.

In this mode, electron beam can be deflected to any visible position. This feature is sometimes referred as *Spot Mode*.

Arguments

```
int ScScanLine(  
    in unsigned int frameid,  
    in unsigned int width,  
    in unsigned int height,  
    in unsigned int x0,  
    in unsigned int y0,  
    in unsigned int x1,  
    in unsigned int y1,  
    in unsigned int dwell_time,  
    in unsigned int pixel_count,  
    in int single  
);
```

return value	0 – ok, < 0 – failed, invalid parameters
frameid	unique frame id sent back in the data callback
width, height	dimensions of the whole scanning window
x0, y0	beginning of the line
x1, y1	end of the line
dwell_time	pixel acquisition time in [ns]
pixel_count	number of pixels on the line
single	0 – continual scanning, 1 – single frame

```
void ScData(  
    in unsigned int frameid,  
    in int channel,  
    in unsigned int index,  
    in int bpp,  
    in char[] data  
);
```

See *ScScanXY()* for details.

Timing

ScScanLine() is executed immediately. Wait A flag is set during scanning. The flag is cleared when single scan has finished or when *ScStopScan* is called.

ScData() is called periodically when image data is ready, the period is about 50 ms.

Remarks

ScScanLine()

The microscope detectors and input image channels must be configured before calling *ScScanLine()*.

Width x *height* is the window size related to *View Field* microscope parameter. The line is drawn from (x0, y0) to (x1, y1). X grows from left to right, Y grows from top to bottom. Top left corner has coordinates (0, 0).

Examples:

ScScanLine(0, 1000, 1000, 0, 0, 999, 999, 1000, 2048, 1)

Scan single image, frame id 0, from top-left corner to bottom-right corner, the line consists of 2048 pixels, each pixel acquisition takes 1000 ns.

ScData()

is called to pass the image data to the client.

Call Context

Scanning must be inactive when calling *ScScanLine()*. No automatic procedure should be running. Make sure the wait flags are set properly.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScScanLitho

This function has been removed due to the redesign of the scanning ramp generator.

Compatibility

1.x.x	2.x.x	DLL
yes	no	no

ScScanXY

Start scanning over rectangular area.

Arguments

int ScScanXY(
 in unsigned int frameid,
 in unsigned int width,
 in unsigned int height,
 in unsigned int left,

```

        in unsigned int top,
        in unsigned int right,
        in unsigned int bottom,
        in int single
    );

    return value          0 – ok, < 0 – failed, invalid parameters
    frameid               unique frame id sent in the data callback
    width, height         dimensions of the whole scanning window
    left, top, right, bottom definition of the visible region
    single                0 – continual scanning, 1 – single frame

```

```

void ScData(
    in unsigned int frameid,
    in int channel,
    in unsigned int index,
    in int bpp,
    in char[] data
);

```

frameid	frame id
channel	channel index
index	index of the first pixel in the data buffer
bpp	bits per pixel (8 / 16)
data	image data buffer

Timing

ScScanXY() is executed immediately. Wait A flag is set during scanning. The flag is cleared when single scan has finished or when *ScStopScan()* is called.

ScData() is called periodically when image data is ready, the period is about 50 ms.

Remarks

ScScanXY()

Scanning according to the supplied parameters is started, other parameters like scanning speed, input signal configuration, etc., should have already been specified.

Width x *height* is the window size related to *View Field* microscope parameter. *Left*, *top*, *right* and *bottom* describe the visible region over which the beam scans.

Examples:

```

ScScanXY(13500, 1000, 1000, 0, 0, 999, 999, 1)
Scan single image, frame id 13500, 1000x1000 pixels.

```

```

ScScanXY(13501, 1000, 1000, 450, 450, 549, 549, 1)
Scan single image, frame id 13501, reduced raster 100x100 pixels from the center.

```

```

ScScanXY(13502, 10000, 10000, 4500, 4500, 5499, 5499, 1)

```

Scan single image, frame id 13502, reduced raster 1000x1000 pixels from the center, digital zoom 10x.

ScData()

is called when image data is sent to the client. If 16-bit scanning is active, one pixel consists of two bytes, the bytes are stored in little-endian order.

Call Context

Scanning must be inactive when calling *ScScanXY()*. No automatic procedure should be running. Make sure the wait flags are set properly.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

ScSetBeamPos

Deflect the electron beam, in other words, set beam position. This can be called only if scanning is inactive.

Arguments

void ScSetBeamPos(in float x, int float y)

x	x position, 0.0 – 1.0
y	y position, 0.0 – 1.0

Timing

Executed immediately.

Remarks

The virtual image size is 1.0 x 1.0. Top left corner has (0, 0) coordinates. The *ScSetBeamPos()* deflects the beam to the given position. The beam remains in this position till either next *ScSetBeamPos()* call is issued or till scanning is started.

Call Context

Scanning must be inactive.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.6 and later	yes	yes

ScSetBlanker

Set beam blanker operating mode.

Arguments

void ScSetBlanker(in int mode)

mode blanking mode

Timing

Executed immediately.

Remarks

Refer to *ScGetBlanker()* for description of the blanking modes.

If the scanning is inactive, blanker is switched on/off immediately. If the scanning is active, blanker mode is changed after the scanning is stopped.

If the beam blanker is not installed, calling this function is legal but has no effect.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

ScSetExternal

Enable/disable external scanning.

Arguments

void ScSetExternal(in int enable)

enable enable flag

Timing

Executed immediately.

Remarks

If *enable* is set to one, external scanning is enabled. See *ScGetExternal()* for details.

External scanning does not affect internal scanning. This function controls simple switching circuit (multiplexer), it does not start or stop the internal scanning.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	yes

ScSetSpeed

Set current scanning speed index.

Arguments

void ScSetSpeed(in int speed)

speed

required scanning speed index

Timing

Typically 5 ms to execute (wait C).

Remarks

If incorrect index is provided, closest valid index is selected automatically.

Call Context

Anytime.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

There is a difference between 1.x.x and 2.x.x protocol versions. In the 2.x.x version, the scanning speed is not changed if scanning is in progress. Scanning must be restarted to apply the new scanning speed value.

ScStopScan

Stop scanning.

Arguments

void ScStopScan(void)

Timing

1 to 20 ms to execute (wait C).

Remarks

If scanning was initialized using one of the scanning functions (*ScScanXY()* or similar), it must be stopped later using *ScStopScan()*. It does not matter if single scan or continual scanning was requested.

It is not guaranteed that the image data has already been sent at the moment this call is completed. Application must use the frame id identifier to distinguish which frame the image data belongs to.

In particular, following command order can be useful.

```
ScScanXY(101, -- first region --, 1);      // single scan of region 101
void ScStopScan(void);                     // wait A set to 1
                                           // we get here when the region 101 is finished
ScScanXY(102, -- second region --, 1);     // single scan of region 102
void ScStopScan(void);                     // wait A set to 1
                                           // we get here when the region 102 is finished
...
```

Note that the *ScStopScan()* call is required after every command which starts scanning. If wait A was not specified, single scan would be terminated prematurely.

Call Context

Scanning must be active when calling this function.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Scanning Mode

SMEnumModes

Enumerate all available scanning modes.

Arguments

map **SMEnumModes**(void)

return value HV index map

Timing

Executed immediately.

Remarks

The map *key=value* has the following form (example):

```
mode.0.name=RESOLUTION
mode.1.name=DEPTH
mode.5.name=WIDE FIELD
```

The total number of scanning modes depends on the SEM model and configuration. Each scanning mode is assigned a number. Some scanning mode numbers may be omitted. That is, the sequence may not be consecutive.

Each map *value* contains **string** value describing name of the scanning mode.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

SMGetMode

Get current scanning mode.

Arguments

int SMGetMode(void)

return value

scanning mode number

Timing

Executed immediately.

Remarks

Refer to *SMEnumModes()* for details.

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

SMSetMode

Set current scanning mode.

Arguments

void SMSetMode(in int mode)

mode

scanning mode number

Timing

Executed immediately. Function following the *SMSetMode()* may be delayed automatically, because automatic degauss of the column takes place and the degauss procedure cannot be interrupted.

Remarks

Only valid scanning modes should be selected. If the scanning mode number is invalid, the behavior is undefined.

Call Context

Scanning must be inactive.

Also Affected

Column centering values are coupled with each scanning mode. It may also happen that view field is changed, because each scanning mode has different magnification limits.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Vacuum

VacGetPressure

Read microscope chamber/column/gun pressure.

Arguments

float VacGetPressure(in int gauge)

return value	pressure in [Pa]
gauge	0 – chamber, 1 – column, 2 – gun

Timing

Executed immediately.

Remarks

Use together with *VacGetStatus()* to obtain complete information about the vacuum conditions in the SEM.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacGetStatus

Read microscope vacuum status.

Arguments

int VacGetStatus(void)

return value	vacuum status
--------------	---------------

Timing

Executed immediately.

Remarks

The vacuum status is a single integer value. It tells the current vacuum conditions of the microscope.

- 1 error – call *VacGetPressure()* to check the pressure
- 0 ready for operation
- 1 pumping in progress
- 2 venting in progress
- 3 vacuum off (pumps are switched off, valves are closed)
- 4 chamber open

It takes several minutes to evacuate the microscope. If sufficient vacuum level is not reached in an arbitrary time, the system reports error (-1).

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacGetVPMode

Get on/off status of the variable pressure mode.

Arguments

int VacGetVPMode(void)

return value 0 – high vacuum operation
 1 – variable pressure operation

Timing

Executed immediately.

Remarks

This function returns the vacuum operation mode. To find out whether the vacuum conditions allow imaging, call *VacGetStatus()* function.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacGetVPPress

Get variable pressure setpoint.

Arguments

float **VacGetVPPress**(*void*)

return value

pressure in [Pa]

Timing

Executed immediately.

Remarks

Does not measure anything, just returns the required chamber pressure setpoint. The value is valid in both high vacuum and variable pressure mode.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacPump

Evacuate the microscope chamber.

Arguments

void **VacPump**(*void*)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

After the pumping procedure start, remote application should check the vacuum status periodically for success/error.

The call has no effect if the microscope is already evacuated.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacSetVPMode

Change vacuum mode from variable pressure to high vacuum and vice versa.

Arguments

void VacSetVPMode(vpmode)

vpmode 0 – high vacuum mode
 1 – variable pressure mode

Timing

Executed immediately. May take several seconds to complete the process, may even require user intervention.

Remarks

The vacuum mode change is a complex process. Some microscopes are not capable of doing the mode change automatically and user intervention is required, it may be necessary to change the aperture configuration or to open/close a valve.

Use *VacGetStatus()* to find out if the microscope is ready for operation after the change of the vacuum mode.

Call Context

Anytime

Also Affected

Detectors are enabled / disabled depending on the vacuum mode. HV is switched off (thermal emission) or beam is switched off (field emission) automatically.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacSetVPPress

Set variable pressure setpoint.

Arguments

void VacSetVPPress(pressure)

pressure

pressure in [Pa]

Timing

Executed immediately.

Remarks

Set variable pressure setpoint. Depending on the SEM model and configuration, the value is silently limited to a certain range. It may take an arbitrary time before the required pressure in the SEM chamber is reached.

Call Context

Anytime

Also Affected

Some detectors may not function if the pressure is out of their range.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

VacVent

Vent the microscope chamber.

Arguments

void VacVent(void)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

Call *VacVent()* when chamber has to be opened. For field emission microscopes, this only allows air (nitrogen) into the chamber. On the other hand, tungsten gun microscopes have no valve between the chamber and the gun. It means that the main high voltage is automatically switched off during the venting procedure.

The call has no effect if the microscope is already vented.

Call Context

Anytime

Also Affected

HV is switched off automatically (Vega\\ only).

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Airlock

Airlock (load lock) is a device for fast specimen exchange. The airlock chamber is attached to the main SEM chamber, there is a separating valve between them through which a sample is pushed in and pulled out.

ArlCloseValve

Close the separation valve between the SEM chamber and the airlock chamber.

Arguments

void ArlCloseValve(void)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

The separation valve is closed and the SEM chamber and the airlock chamber are separated.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.7 and later	yes	no

ArlGetStatus

Read airlock status.

Arguments

int ArlGetStatus(void)

return value

airlock status

Timing

Executed immediately.

Remarks

The airlock status is one of:

- 2 airlock is not installed
- 1 error
- 0 ready – separation valve can be opened
- 1 open – separation valve is opened
- 2 pumping in progress
- 3 venting in progress
- 4 airlock vented, door closed
- 5 airlock vented, door open
- 6 vacuum system is switched off

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.7 and later	yes	no

ArlOpenValve

Open the separation valve between the SEM chamber and the airlock chamber.

Arguments

void ArlOpenValve(void)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

ArlOpenValve() can be called only if both SEM chamber and airlock chamber are evacuated. The effect is that the separation valve is opened and a sample can be inserted to or removed from the SEM chamber.

Call Context

Both SEM and airlock chambers evacuated.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.7 and later	yes	no

ArlPump

Pump (evacuate) the airlock chamber.

Arguments

void ArlPump(void)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

ArlPump() evacuates the airlock chamber. The pumping is slower than venting, it takes tens of seconds. Check *ArlGetStatus()* to find out when the pumping is finished.

The call has no effect if the airlock is already evacuated.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.7 and later	yes	no

ArlVent

Vent the airlock chamber.

Arguments

void ArlVent(void)

Timing

This is a non-blocking call, it is executed immediately.

Remarks

Call *ArlVent()* if the separating valve is closed and the airlock chamber shall be open to air. The venting is fast (several seconds).

The call has no effect if the airlock is already vented.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.7 and later	yes	no

High Voltage

In this chapter, High Voltage stands for the main accelerating voltage of the microscope. The tungsten gun and the field emission microscopes use different accelerating voltage power supply unit. They share the same remote control function set and implement commands in a bit different way if necessary.

The accelerating voltage is set up in two steps:

- 1) select one of the HV indexes
- 2) set the actual HV value

The reason for using HV indexes is that it is not possible to use a single set of column configuration values for the whole HV range. There are rather several indexes and each of them has its own column configuration. The per-index ranges do not overlap.

HVAutoHeat

Start automatic filament heating procedure. Only supported by tungsten gun SEMs.

Arguments

void HVAutoHeat(in int channel)

channel input video channel

Timing

Execution time between 20 and 60 seconds (wait D).

Remarks

This function is supported by tungsten gun SEMs only.

Channel index is a zero based index of the input channel. Appropriate detector must be selected as an active one.

During the automatic heating run, *AutoGun()* and *DtAutoSignal()* procedures are invoked.

The heating level is adjusted only for the current HV index. The other HV indexes remain untouched.

The automatic procedure relies on the specimen, it won't succeed if conductive specimen is not installed or if the detector does not give reasonable image signal.

See also *DtSelect()*, *AutoGun()*, *DtAutoSignal()*, *HVGetHeating()*.

Call Context

Scanning must be inactive, no automatic procedure running.
HV must be switched on.

Beam must be switched on.
Valid detector must be selected.
Non-conductive specimen must be installed.

Also Affected

Gun centering, gain and black level of the active detector.

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

HVBeamOff

Turn off the beam.

Arguments

void HVBeamOff(void)

Timing

Executed immediately.

Remarks

This function switches off the beam. The way how this is done depends on the SEM type.

Tungsten gun: turn off the filament heating.
Field emission gun: close the gun valve.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVBeamOn

Turn on the beam.

Arguments

void HVBeamOn(void)

Timing

Executed immediately.

Remarks

This function switches on the beam. See *HVBeamOff()* for details.

Tungsten gun microscopes require several seconds to stabilize the beam after it has been switched on.

Call Context

If the vacuum is not *ready*, this command is silently ignored. See *VacGetStatus()* how to determine the vacuum conditions.

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVEnumIndexes

Enumerate HV indexes.

Arguments

map *HVEnumIndexes(void)*

return value HV index map

Timing

Executed immediately.

Remarks

It is not possible to set any accelerating voltage immediately. Instead of that, it is necessary to select one of predefined HV indexes first. Each index several microscope parameters associated:

- actual HV value
- set of column centering values
- gain, offset of detectors
- gun heating level and HV bias (tungsten gun only)

If an index is selected by calling *HVSetIndex()*, all above parameters are loaded automatically.

The map *key=value* has the following form:

hv.0.min=200
hv.0.max=5000

```

hv.0.value=1000
hv.1.min=5000
...
hv.N.min=20000
hv.N.max=30000
hv.N.value=30000

```

N+1 is the total number of HV indexes. The typical number of indexes is four, but application must be able to handle different count.

The map *value* element contains accelerating voltage in [V] for given HV index. The *min* and the *max* elements contain minimum allowed [V] and maximum allowed voltage [V] for this index, respectively. All numbers are floating point.

For future compatibility, application must silently ignore any *key* which does not have exactly the above form.

See also *HVGet()*, *HVSet()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Min, max introduced in version 1.0.5

HVGetBeam

Get beam status.

Arguments

int HVGetBeam(void)

return value	-1	filament is blown
	0	beam is off
	1	beam is on

Timing

Executed immediately.

Remarks

Determine the current beam status. It is possible to switch the beam on and off by *HVBeamOn()* and *HVBeamOff()* functions.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVGetEmission

Read measured emission current. The result is an approximate current emitted from the SEM filament tip.

Arguments

float **HVGetEmission**(void)

return value

emission current [μ A]

Timing

Executed immediately.

Remarks

First, it is necessary to determine the gun status by *HVGetBeam()*. The emission current value is valid only if the beam is on.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVGetFilTime

Get the filament live time since the last exchange.

Arguments

int **HVGetFilTime**(void)

return value

filament live time [s]

Timing

Executed immediately.

Remarks

The result is the time elapsed since the last filament exchange. Only the live time (operation time) is counted.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

HVGetHeating

Read filament heating current.

Arguments

float ***HVGetHeating***(void)

return value

requested heating current [A]

Timing

Executed immediately.

Remarks

Returns the requested heating current. This is an indicative value. Depending on the SEM type, the proper heating current can be between 1.0 A and 3.0 A.

The heating level has separate value for each HV index. *HVGetHeating()* returns the value associated with the current index.

See also *HVAutoHeat()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
-------	-------	-----

1.0.5 and later	yes	no
-----------------	-----	----

HVGetIndex

Get current HV index.

Arguments

int **HVGetIndex**(void)

return value	0, 1, 2, ...	HV index
	1000	HV is switched off
	1001	HV change is in progress

Timing

Executed immediately.

Remarks

Current HV status is determined and returned. There can be several HV indexes (typically four). Each index is linked with one predefined accelerating voltage value.

See *HVEnumIndexes()* for general discussion what HV index is.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVGetVoltage

Read current accelerating voltage.

Arguments

float **HVGetVoltage**(void)

return value	accelerating voltage [V]
--------------	--------------------------

Timing

Executed immediately.

Remarks

See also *HVGetIndex()*, *HVSetVoltage()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	yes

HVSetIndex

Set HV index and load all associated parameters.

Arguments

void HVSetIndex(in int index)

index

valid HV index

Timing

Several seconds (maximum about 100 seconds if the HV change is significant) until the system becomes stable (check Wait C flag).

Remarks

HV index is set and all associated parameters are loaded.

See *HVEnumIndexes()* for general discussion what HV index is.

Since many microscope parameters are adjusted by the HV index change, it is recommended to check the operating conditions and the image quality after the function is finished.

Call Context

Anytime

Also Affected

Accelerating voltage, column centering values and detector settings are changed. It is also not guaranteed that view field size, scanning speed and PC settings will persist.

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

HVSetVoltage

Set current accelerating voltage.

Arguments

void HVSetVoltage(in float voltage)

voltage

accelerating voltage [V]

Timing

Several seconds (maximum about 50 seconds if the HV change is significant) until the system becomes stable (check Wait C flag).

Remarks

The requested HV value must be inside the range of the currently selected HV index. If the value is outside the range, it will be limited. HV index is not switched by this function.

See also *HVEnumIndexes()*, *HVSetIndex()*, *HVGetVoltage()*.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

Chamber Camera

Most SEMs are equipped with chamber camera – chamberscope. Remote client can ask for video from the camera. The video consists of static images sent in regular intervals.

CameraDisable

Stop the chamberscope image stream.

Arguments

void CameraDisable(void)

Timing

Executed immediately.

Remarks

The SEM stops sending the stream of camera images. See *CameraEnable()* for details.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.6 and later	yes	no

CameraEnable

Open camera window in the SEM software, enable the chamberscope image stream.

The camera functions have been introduced in protocol version 1.0.6.

Arguments

*void CameraEnable(
 in int channel,
 in float zoom,
 in float fps,
 in int compression
)*

channel	reserved, must be 0
zoom	zoom (0.05 – 1.0)
fps	frames per second (0.1 – 5.0)

compression reserved, must be 0

```
void CameraData(  
    in int channel,  
    in int bpp,  
    in int width,  
    in int height,  
    in char[] data  
);
```

channel	channel index (reserved, should be 0)
bpp	bits per pixel (always 8)
width	image width
height	image height
data	image data, width * height bytes

Timing

If the Chamber View module in the SEM software is closed, it may take several seconds till the client receives valid image. If the Chamber View module is running, the first image is sent immediately.

Remarks

CameraEnable()

The camera image size is not fixed, the actual size is sent in the image callback. Client can ask for image with reduced size (if zoom < 1.0).

CameraData()

Image is uncompressed, grayscale, sent in a byte array. The data starts with top-left corner. The camera image is resampled according the client's request. Each image is sent at once, in a single message.

The *CameraData()* messages are sent through the data channel (same as the scanning data).

Call Context

Call *CameraEnable()* only if camera is disabled.

Also Affected

The Chamber View module in the SEM software is opened if necessary.

Compatibility

1.x.x	2.x.x	DLL
1.0.6 and later	yes	no

CameraGetStatus

Get chamber camera status.

Arguments

*void **CameraGetStatus**(in int channel, out int is_enabled,
out float zoom, out float fps, out int compression)*

channel	channel index (reserved, should be 0)
is_enabled	0 – off, 1 – on
zoom	zoom
fps	frames per second
compression	compression mode, always 0

Timing

Executed immediately.

Remarks

-

Call Context

Anytime

Compatibility

1.x.x	2.x.x	DLL
1.0.6 and later	yes	no

SEM GUI Control

There are a few functions which control the Vega / Mira / Lyra SEM graphical user interface. Generally speaking, the remote application should not modify or use the SEM user interface, this is undesirable. On the other hand, there are situations when some interaction between the remote client and the server UI is eligible and suitable.

GUIGetScanning

Get scanning status of the SEM GUI.

Arguments

int GUIGetScanning(void)

return value 0/1 flag - SEM live window is scanning

Timing

Executed immediately.

Remarks

Return value:

0 – no SEM live window is currently scanning

1 – there is a live SEM window currently scanning

When combined with *GUISetScanning()*, it is possible to stop the SEM scanning, execute some procedure, and then restore the scanning back to its original state.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

GUISetScanning

Enable / disable the SEM GUI scanning.

Arguments

void GUISetScanning(in int enable)

enable

enable flag

Timing

Executed immediately.

Remarks

If *enable* is set to one, scanning in the topmost live window is started. If it is zero, scanning is stopped. See *GUIGetScanning()* for details.

Call Context

Anytime

Also Affected

The result of *GUISetScanning(1)* call is that continual scanning is started. The remote client must not issue any scanning requests until the scanning is stopped (either by invoking *GUISetScanning(0)* or *ScStopScan()*).

Compatibility

1.x.x	2.x.x	DLL
1.0.5 and later	yes	no

Miscellaneous

ChamberLed

Switch the chamber illumination LED on and off.

Arguments

void ChamberLed(in int onoff)

onoff 0 – off, 1 – on

Timing

Typically 10 ms to execute (wait C).

Remarks

If there is a detector sensitive to the LED light, detector recovery time must be considered carefully.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no

Delay

Delay execution.

Arguments

void Delay(in int delay)

time time in [ms]

Timing

Wait C is set for given period.

Remarks

This function does not delay execution. It only sets Wait C flag. Thus, following commands are delayed only if they have Wait C flag set to one.

When the specified time has elapsed, the Wait C flag is reset.

Call Context

Anytime

Also Affected

-

Compatibility

1.x.x	2.x.x	DLL
yes	yes	no