Dynamic .NET library for TCP/IP communication with a laser system Ready to use for Visual C++ and/or .NET (C#, Visual Basic)

Revision 9: firmware 5.0.0 and higher necessary to support all commands

Last changes in revision 8:

> -MLaser_Usermessage() function no longer support . Use the MLaser_FastUsermessage() command.
>
> -the DLL calls are now thread-safe. Each function call on a valid socket requires a lock. The lock is automatically assigned and released after the call has finished.
> A timeout for assigning the lock of 10 seconds is set. If the call cannot get the lock the function call stops and return the value -2.

Last changes in revision 9:

-MLaser_SetDynamic(..), MLaser_GetDynamic() added.

Last changes in revision 10:

-MLaser_GetFastUserMessage() and MLaser_GetFastUTF8UserMessage() added

Last changes in revision 11:

-extended status structure
-datastring messages for dynamic bitmaps

Last changes in revision 12:

-adapt Testpointer() call for firmware version > 44 (5.1.0)

Last changes in revision 13:

-make the DLL completely thread-safe even for multiple socket connections.

Last changes in revision 14:

-add the MLaser_Eventhandler() function call to activate/deactivate the internal eventhandler.

Last changes in revision 15:

-add the MLaser_GetMessages() function call to retrieve the filenames.

Last changes in revision 16:

-add the MLaser_Store() function call to retrieve the filenames.
Last changes in revision 17:

-add "partial" parameter to MLaser_Asciiconfig() function call to set only specific configuration parameters

Last changes in revision 18:

-change parameter type from char to wchar_t in the following functions:

Minit()
MGetLastError()
MLaser_Start()
MLaser_Delete()
MLaser_SetDefault()
MLaser_CopyFile()
MLaser_AsciiConfig()
MLaser_Eventhandler()
MLaser_GetFilenames()

Remark:
This changes were necessary to make it easier to adapt the DLL to .Net applications. In this calls the char parameter (1 byte) was replaced by a wchar_t (2 bytes) and represents an array of Unicode characters (UTF16 coding). Note that even then you should always use valid ASCII filenames. Only the path parameter (in MLaser_CopyFile()) may contain non ASCII characters.

****

Files:

Unmanaged native C++ DLL: SocketCommDll.dll

-SocketCommDll.lib: the library, that you have to include for linking when you make an application with C++.
-.\SocketCommDllExport.h: the header file, that is necessary to import and use the functions of the dynamic library
-SocketCommDll.dll: the dynamic link library that exports the functions (must be present in the application path of your executable program that uses the functions of this DLL). This DLL is a C++ native DLL. It is available for x86 applications as well as for x64 applications.

Managed C# DLL: SocketCommNet.dll

-SocketCommNet.dll: a managed .Net Dll that just works as a wrapper to the native SocketCommDll.dll. If you develop a .Net application you can try to include by yourself the corresponding native Dll (SocketCommDll.dll for x64 or x86) . However, we recommend to use the SocketCommNet.dll, as it is a managed class that can be easily included in .Net applications.
If you use this wrapper dll , please note that you always have to use also the native SocketCommDll.dll. In this case just copy the SocketCommDll.dll into your application path. To avoid confusions we always recommend to copy the SocketCommDll.dll into your application's directory.

The accessible functions of the wrapper class are called simply CS_<native name without leading MLaser or leading M> and the parameters are already wrapped to C# accessible objects.

Last changes in revision 19:

Add the MLaser_DumpSVG() function to get a preview of the message data.

Last changes in revision 20:

Add the MLaser_ShiftRotate() function.

Last changes in revision 21:

Add the MGetDllVersion() function.
Reset IsConnected() when next command was successful (.e.g. after a timeout)

Last changes in revision 22:
Change first parameter of Minit() from (long* p) to (int& p). Replace all <long> parameters in the function calls with <int>. The Minit() parameter <p> is now just an index that refers internally to a communication handle. This change was necessary to be fully compatible with 64-bit applications.

Last changes in revision 23:
Add <len> parameter to Mlaser_GetGlobalcounter().
Add Mlaser_GetFifofield(), Mlaser_GetUTF8Fifofield(), Mlaser_FifoDump().
Add Mlaser_MultipleUsermessage(), Mlaser_MultipleUTF8Usermessage().
Add Mlaser_GetMultipleUsermessage().
Add Mlaser_GetMultipleUTF8Usermessage().

Last changes in revision 24:
correct Mlaser_GetDynamic() to retrieve correct values.

Last changes in revision 25:
increment number of sockets from 24 to 48
increment PrintStatusExt structure by 4-byte alarmmask2

Last changes in revision 26:
add Mlaser_Defocus() to change dynamically the focus
add Mlaser_Sysinfo() to get some systeminformation
add Mlaser_Coretemp() to get some core parameters

Last changes in revision 27:
correct Mlaser_Defocus(); in version 26 parameters “relative” and “format” were swapped !

Last changes in revision 28:
add Mlaser_DumpSVGExt() to add a filter to the dump information

Last changes in revision 29:

    add Mlaser_Signalstate() to select layers as printable/not printable

Last changes in revision 30:

    resolved mismatch in StatusExt structure (alarmmask2 was not set correctly)

Last changes in revision 31:

    add MLaser_GetVersionString()
    add MLaser_GetConnectionData
    add MLaser_ServerShutdown()
    add MLaser_PrintMode()

Example projects:

.\SocketCommCPPTest:

A Visual C++ example (at least Visual Studio 2005 necessary) that uses the SocketCommDll.lib, SocketCommDllExport.h and the SocketCommDll.dll.

.\SocketCommNet:

A C# project that creates the managed wrapper DLL SocketCommNet.dll, which itself needs the native SocketCommDll.dll.
Use this wrapper DLL if you want to develop any .Net application. For this purpose, copy the SocketCommNet.dll and the SocketCommDll.dll to the application's executable path.

.\NetDllTest:

A simple C# project that uses the managed wrapper DLL SocketCommNet.dll (and the native SocketCommDll.dll). Be sure to have the SocketCommNet.dll and the SocketCommDll.dll copied to your application's executable path.

.\NetDllTestVb:

A simple VisualBasic .NET project that uses the managed wrapper DLL SocketCommNet.dll (and the native SocketCommDll.dll). Be sure to have the SocketCommNet.dll and the SocketCommDll.dll copied to your application's executable path.

Headerfile 'SocketCommDllExport.h':

```
////////////////////////////////////////////////////////////
//      Status structure: can be filled with MLaser_Status() call.
//
//
//
////////////////////////////////////////////////////////////
typedef struct{
  Uint32          d_counter;//actual counter of ok-prints
  Uint32          s_counter;//actual counter of nok-prints
  Uint32          n_messageport;//actual external message selection
  unsigned char           Start;//
        Bit0:  when set, system is in printing mode (waiting for photocell/PLC)
        Bit1:  when set, system is actually printing a message
        Bit2: when set, system is waiting for a 'alarm reset' signal (DSP cards         only).
        Bit3: when set, system is waiting for an input signal (DSP cards only)
        Bit4: when set, system is waiting for an external motorized axis to reach
                a new position.
        Bit5: when set, system is in ‘printsession’ mode.
        Bit6...Bit7: reserved
  unsigned char           request;//(internal variable)
  unsigned char           option;//(internal variable)
  unsigned char           res;//reserved (0x0:default, 0x1: UMT enabled, 0x4: Batchjob
                          enabled)
  Uint32          t_counter;//total counter of prints
  Uint32          m_copies;//actual nr of copies to be printed
  Uint32 err;//alarmcode (upper WORD: codifies lastactive alarm; lower WORD: 0:
                  noalarm, else: alarm active)
  Uint32 time;//reserved D-Word (actually used as timeofprint-info)
  char  name[8];//actual active filename (max.8 chars)
  Uint32 reserved1;       //codes tha alarmmask
  Uint32 reserved2;       //codes the signalstate
} PStatus;

//extended status structure: requested with MLaser_StatusExt()-call
//
//
typedef struct{
  Uint32          d_counter;//same as above
  Uint32          s_counter;//same as above
  Uint32          n_messageport;//same as above
  unsigned char           Start;//same as above/
  unsigned char           request;//same as above
  unsigned char           option;//same as above
  unsigned char           res;//same as above
  Uint32          t_counter;//same as above
  Uint32          m_copies;//same as above
  Uint32 err;//same as above
  Uint32 time;//same as above
  Uint32 alarmmask1; //codes the alarmmask
  Uint32 signalstate;    //code the IO signalstate
  char  messagename[16];//filename (max.16 chars = 12 + 4)
  char  eventhandler[16];//filename (max.16 chars = 12 + 4)
  Uint32 alarmmask2; //codes the extended alarmmask
  Uint32 extra;  //Bit0..Bit9l dynamic usage of scanfield in permille;
                 // Bit10..Bit13 actual dynamic mode

} PStatusExt;
```

Description of the important variables:

This is the status structure with the most important status variables of the laser. The variable may be filled with a call to Laser_Status(&p).

d_counter: an internal counter, that counts the "good" prints (prints without errors). The variable can be resetted to zero wtih a call to Laser_CounterReset().

s_counter: an internal counter, that counts the "bad" prints (prints with errors). The variable can be resetted to zero wtih a call to Laser_CounterReset().

t_counter: an internal counter, that counts the total prints. This variable cannot be resetted.

n_messageport: this variable shows the actual BYTE value of the external message selection, when the system is in the mode "external selection".

Start: 0: a bitmask:

Bit0: when set, system is in printing mode (waiting for photocell/PLC)

Bit1: when set, system is actually printing a message

Bit2: when set, system is waiting for a 'alarm reset' signal (DSP cards only).

Bit3: when set, system is waiting for an input signal (DSP cards only)

Bit4: when set, system is waiting for an external motorized axis to reach a new position.

Bit5: when set, system is in ‘printsession’ mode.

Bit6...Bit7: reserved

err: variable that codifies the internal alarms of the system (devided in upper and lower WORD).

upper WORD (err>>16):

is used to codify the last alarm (0: no last alarm <> 0: the value codifies the last alarm.

lower WORD (err & 0xFFFF):

0:	no alarm is active
<>0:	some alarm is active

Note: you should use only the lower WORD to get access to the alarmstatus (to know if any alarm is active or not). Even a better approach for this is to use the reserved1/alarmmask1 field of the Status/Extended Status.
The upper WORD codes the last alarm even if this alarm is no longer active. The last alarm will be cleared with the Start command if no alarm is active.

name: an array of 8 BYTES that contains the name of the actual active printing file, not necessarily ending with '\0'.

reserved1/alarmmask1: the alarm-mask. This indicates the actual status of the alarms. If == 0, then no alarm is active. Each bit of the alarm-mask indicates a specific alarm according the following list:

0x00000001	**Interlock**
0x00000002	OEM-shutter
0x00000004	Overtemperature
0x00000008	Shutter
0x00000010	Laser not ready
0x00000020	X-scanner failure
0x00000040	Y-scanner failure
0x00000080	power failure(D5000 series and FIBER-laser (MO))
0x00000100	Z-scanner failure
0x00000200	Laser not armed
0x00000400	XY outofrange
0x00000800	Q-switch (D-5000 B-series)
0x00001000	triggersignal
0x00002000	file not allowed (wrong version)
0x00004000	overspeed
0x00008000	harddisk full
0x00010000	barcode creation failure
0x00020000	barcode licence failure
0x00040000	barcode library failure
0x00080000	invalid file
0x00100000	database failure
0x00200000	max.-distance alarm
0x00400000	min.-distance alarm
0x00800000	client-timeout (tcpip)
0x01000000	invalid font
0x02000000	belt stopped
0x04000000	empty message
0x08000000	initialization error
0x10000000	memory error
0x20000000	warmup in progress

0x40000000	OEM alarm active
0x80000000	extended alarm active

reserved2/signalstate: codes the state of the inputs of the system when it was triggered for printing (for internal purpose only)

messagename: a byte array of 16 BYTES that contains the name of the actual active printing file with its extension, not necessarily ending with '\0'.

eventhandler: a byte array of 16 BYTES that contains the name of the actual active eventhandler file with its extension, not necessarily ending with '\0'.

alarmmask2: extended alarm-mask. When the alarm-mask has the ‘extended alarmmask’ -bit (0x80000000) set, this DWORD codes the extended alarms.

0x00000001	Lasermeasurement failed
0x00000002	UV laser not ready
0x00000004	Pixmap out of range
0x00000008	Channelstatus error
0x00000010	PWM out of range
0x00000020	RTC alarm (real time clock)
0x00000040	CPU overtemperature
0x00000080	Board overtemperature
0x00000100	Undervoltage 5V
0x00000200	Undervoltage 3.3V
0x08000000	DSP alarmmask failure
0x10000000	Fpga-watchdog
0x20000000	Wrong lasertype selected with dipswitches
0x40000000	DSP is paused
0x80000000	Fpga failure

extra: //Bit0..Bit9l dynamic usage of scanfield in permille;
// Bit10..Bit13 actual dynamic mode

C# status structure (used in SocketCommNet.dll):

```
public struct CSStatus
    {
     public UInt32 d_counter;//actual counter of ok-prints
     public UInt32 s_counter;//actual counter of nok-prints
     public UInt32 n_messageport;//actual external message slection
     public Byte Start;//0: system is printing 1: system is stopped
     public Byte request;//(internal variable)
     public Byte option;//(internal variable)
     public Byte res;//reserved (0x0:default, 0x1: UMT enabled, 0x4:
      Batchjob enabled)
     public UInt32 t_counter;//total counter of prints
     public UInt32 m_copies;//actual nr of copies to be printed
     public UInt32 err;//alarmcode (upper WORD: codifies lastactive
      alarm; lower WORD: 0: noalarm, else: alarm active)
     public UInt32 time;//reserved D-Word (actually used as
      timeofprint-info)
     public String name;//actual active filename (max.8 chars)
     public UInt32 reserved1; //reserved
     public UInt32 reserved2; //reserved
    }
public struct CSStatusExt
   {
   public UInt32 d_counter;//actual counter of ok-prints (reseted when
      message changes)
   public UInt32 s_counter;//actual counter of nok-prints (reseted
      when message changes)
   public UInt32 n_messageport;//actual file-nr printing
   public Byte Start;//printing or stopped
      //BIT0: printing loop (prepared for printing) BIT1:   printing
      (we are actually marking)
   public Byte request;//waiting for request
   public Byte option;//options for HandleRequest()
   public Byte res;//reserved (0x0:default, 0x1: UMT enabled, 0x4:
      Batchjob enabled)
   public UInt32 t_counter;//total counter of prints
   public UInt32 m_copies;//actual nr of copies to be print
   public UInt32 err;//alarmcode
   public UInt32 time;//reserved D-Word (actually used as time-info
   public UInt32 alarmmask1;  //codes the alarmmask
   public UInt32 signalstate; //codes the IO signalstate
   public String messagename;//filename (max.16 chars = 12 + 4)
   public String eventhandler;//filename (max.16 chars = 12 + 4)
   public UInt32 alarmmask2;  //codes the extended alarmmask
   public UInt32 extra;
   }

public struct CSSysinfo
   {
    public UInt32 cputemp;//cpu temperature in 1/1000 Celsius
    public UInt32 size0;//harddisk in bytes total space in Bytes
    public UInt32 avail0;//in bytes available space in Bytes
    public UInt32 size1;//ramdisk in bytes total space in Bytes
    public UInt32 avail1;//in bytes available space in Bytes
    public UInt32 size2;//ramfont in bytes total space in Bytes
    public UInt32 avail2;//in bytes available space in Bytes
    public UInt32 size3;//logdrive in bytes total space in Bytes
    public UInt32 avail3;//in bytes available space in Bytes
    public float hours;//working hours of the system hours
    public UInt64 longcounter;//total number of prints
   }
 public struct CSCoretemp
     {
      public UInt32 cputemp;//cpu temperature in 1/1000 Celsius
      public UInt32 boardtemp;//board temperature  in 1/1000 Celsius
      public UInt32 humidity;//humidity  in 1/1000 percent
      public UInt32 voltage1;//5 V in 1/1000 Volt
      public UInt32 voltage2;//3.3V  in 1/1000 Volt
      public UInt32 fanlocaltemp;//local temperature sensor in Celsius
      public UInt32 fancurrentpwm;//current PWM of the fan in percent
      public UInt32 fantacho;//cntspersec of the fan in cps
      public UInt32 fanremotetemp;//remote temp sensor in Celsius
    }
```

The DLL functions:

int MGetDllVersion()

Returns the version number of the DLL. The version number is an increasing ordinal number that changes with each new release.

C# function of SocketCommNet.dll

public **int CS_GetDllVersion**()

void MInit(int &p,const wchar_t *name,const wchar_t *ip,const wchar_t *path)

p: reference to an int variable. The Minit() function fills this variable with a int number that must be used as a reference for all other function calls with this connection.

name: name of the system to be connected to; actually without meaning and reserved for future application.

ip: string of the IP of the system to connect to (e.g. "192.168.0.180")

path: string of the default path to be used when copying local files (e.g. ".\\")
Note: the path parameter is no longer used. The path must be set directly in the Corresponding MLaser_PrintCopyFile() function call !

NOTE: In all following function calls the value of p, that was assigned by the Minit() function call, must be passed to the function calls. It is an index that references to the internally used communication handle. Its value ranges from 1 – 24.
Up to 24 connections can be opened simultaneously. If the return value (p) is zero, you probably may have opened already 24 connections. You must then call the MFinish() function for one of your open connections.
C# function of SocketCommNet.dll

public **void CS_Init**(ref **int** *p*, **string** *name*, **string** *ip*, **string** *path*)

Return values of the function calls:

Calling any function of the library should return 0 if no error has occurred. In case of an errorstring is set which can be requested with a call to ***MGetLastError()*** .

The most common non-zero return values are due to:

Return value:

0:	no error
-1:	no socket client open for communication
1:	sending of TCP/IP packet failed or timeout (no answer from laser)
2:	answer command does not match with sent command
3:	timeout during establishing the connection. The default timeout is 2000 ms and can be increased with ***MSetTimeout.***
32:	command not supported or accepted by the laser
256:	too less data in answer
512:	unknown protocol error

Some specific return values are described in the function calls.

int MStartClient(int p)

Opens a communication socket to the system. Returns 0 if the communication i established; returns <>0 if an error occured. This function must be called once after constructing a socketcomm object. Once the function was called succesfully, you can send control commands to the laser system.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_StartClient**(**int** *p*)

void MFinish(int p)

Shuts down the communication socket defined by <p>. This function should be the last function call before leaving the program or if you do not want to use anymore the assigned socket defined by <p>. It cuts the socket communication and frees some internally reserved memory.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll
public **void** **CS_Finish**(**int** *p*)

int MLaser_ServerShutdown(int p, Int32 bexit)

Forces the connected server to shutdown and restart in case that bexit=0. When bexit=1, then the connected server shuts down and forces the laser's firmware to shutdown, after which the laser will not be responsive any more. In case of a restart (bexit=0) the actual socket will not be responsive. You would need to use MLaser_StartClient(p) again to reconnect to the laser or Mfinish(p) if you do not want to use the assigned socket defined by <p>.

p: the value of the communication handle assigned by Minit()

bexit; argument to determine a restart (0) or a complete shutdown (1)

C# function of SocketCommNet.dll

public **int** **CS_ServerShutdown**(**int** *p, Int32 bexit*)

int   MLaser_Knockout(int p)

Indicates to the connected laser system, that the connection will be shut down. This function call is necessary to close carefully the socket connection of the laser system and should be called before the ShutdownClient() function.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Knockout**(**int** *p*)

int   ***MGetLastError(int p ,wchar_t &txt)***

Fills the txt-array of max. 256-wchar_t. The provided string contains a description of the error that has occurred (only communication errors). Usually, you should not get any communication error, except a timeout error when the laser system is blocked.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_GetLastError**(**int** *p*, ref **string** *txt*)

int MIsConnected(int p)

Returns 0 if the system is not connected and 1 if the system is connected.

p: the value of the communication handle assigned by Minit()

C# fun ction of SocketCommNet.dll

unsigned short MGetVersion(int p)

Returns the internal version number of remote serverprogram.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **ushort** **CS_GetVersion**(**int** *p*)

int MLaser_GetVersionString(int p, char *out,int len,int option)

Returns the internal version string of remote serverprogram.

p: the value of the communication handle assigned by Minit()

out: a reserved buffer for the receiving string

len: the length of the reserved buffered

option: 0 , gets the first version string
            1   gets the second version string

The version strings are informative only.

C# function of SocketCommNet.dll

public **int** **CS_GetVersionString**(**int** *p,ref string txt, int option*)

int MLaser_GetConnectionData(int p, unsigned char &b1, unsigned char &b2, unsigned short foundmask)

Gets some internal connection data information. The data are only valid when the system is connected.

p: the value of the communication handle assigned by Minit()

b1: a byte indicating the following

0xF1: 64-bit version with internal barcode library
0xF0: 32-bit version with internal barcode library
0xFF: version without internal barcode library

b2: a byte indicating the following

0xFF: firmware aborted (rescue program is active)

In any other case the following Bits are cleared according to the properties of the running firmware:

Bit0: internal use
Bit1: running firmware for a SM108 controlcard
Bit2: running firmware for a SM117 controlcard
Bit3: running firmware for a GAMEART controlcard
Bit4: running firmware for a SM120 controlcard
Bit5: running firmware for a SM121 controlcard
Bit6: running firmware for a SM140 controlcard
Bit7: running firmware for a DSP controlcard

foundmask: a WORD indicating the following

Upper byte: internal use
Lower byte:
0x00: no scannercard found
0x02: SM108 found
0x04: SM117 found
0x08: GAMEART found
0x10: SM120 found
0x20: SM121 found
0x40: SM140 found
0x80: DSP card found

C# function of SocketCommNet.dll

```
public int CS_GetConnectionData(int p,ref byte b1, ref byte b2, ref UInt16 foundmask)
```

int MSetTimeout(int p ,int timeout)

Sets communication timeout variable. The value is passed in units of milliseconds. The default value (if this function is not called) is 2000 ms.
If the time that passes since start of a command sent to the laser and until receiving an answer from the laser is greater than the timeout, a timeout error occures and the corresponding function call returns with an error.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_SetTimeout**(**int** *p*, **int** *timeout*)

int MLaser_Start(int p ,const wchar_t *filename,int nr)

Indicates the system to enter into the printing mode with the actual file set to <filename>. The <filename> should be passed without extension in case of msf-files and must include the extension in case of xml-files!
int nr: number of prints to be done. If nr is equal to 1, a testprint will be done immediately; if nr is equal to 0, the system enters into an eternal printing mode until the stop signal is sent, or until some internal error occures. If set to 0xffffffff the system prints 1 copy but waits for the photocell/PLC signal.

p: the value of the communication handle assigned by Minit()

specific return values:

0: no error
4: selected file not found
8: some alarms active

Note: this start command should not be used in case of an enabled external message table or an enabled batch job (use MLaser_StartExtended() for this purpose). If a NULL or empty filename is passed, the laser activates the standard printing mode (batch mode and external table mode will be deactivated) and tries to load "0.msf" as the printing file. This command can thus be used to switch back from batch o external table mode to the standard mode.

C# function of SocketCommNet.dll

public **int** **CS_Start**(**int** *p*, **string** *filename*, **int** *nr*)

int MLaser_StartExtended (int p ,int nr, int msg, int batch)

Indicates the system to enter into the printing mode either with the external message table activated or using the batch job table.

p: the value of the communication handle assigned by Minit()

int nr: number of prints to be done (if nr is equal to 1, a testprint will be done immediately; if nr is equal to 0, the system enters into an eternal printing mode until the stop signal is sent, or until some internal error occures.)

int msg: specifies the table-entry where the printing should be started in case of a batch job table. This value is ignored if you are using the external message table. Valid values are 0 – 255.

int batch:

> if batch==0, then the system activates the external message table selection and the actual file will be determined depending on the hardware input-signals at the customer connector.
>
> If batch != 0, then the system activates the batch table execution.
> To start the batch table execution at a arbitrary position you must specify the position with the parameter <msg> and you must use the MLaser_Reload() command to force the laser to activate the specified entry.

specific return values:

0: no error
4: selected file not found
8: some alarms active

C# function of SocketCommNet.dll

public **int** **CS_StartExtended**(**int** *p*, **int** *nr*, **int** *msg*, **int** *batch*)

int ***MLaser_Stop(int p ,int timeout)***

Send the stop signal to the system. The laser stops immediately printing and leaves the printing mode. An optional timeout value can be passed to the function call (if timeout is equal to zero, the default timeout is assumed).

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Stop**(**int** *p*, **int** *timeout*)

int ***MLaser_Reload(int p)***

Commands the system to reload the actual printing file. This call is usually used after having sent an updated printing file to the laser.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Reload**(**int** *p*)

int MLaser_TriggerPrint(int p)

Simulates a photocell/PLC signal for printing. It can be used as a software trigger instead of the photocell/PLC inputs. The system will only print, if the system is in printing mode and no alarm is active.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_TriggerPrint**(**int** *p*)

int MLaser_CounterReset(int p)

Resets the internal d_counter and s_counter of the laser's internal status structure. Recommended to use each time before calling the Laser_Start() function, so you can easiely control the number of prints since having entered into the printing mode.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_CounterReset**(**int** *p*)

int MLaser_Settime(int p)

Sets the actual system time in the remote lasersystem. The local operation system time is used as the reference time.

p: the value of the communication handle assigned by Minit()

specific return values:

0: no error
3: error in date format
4: error in time format
5: error in date+time format

C# function of SocketCommNet.dll

public **int** **CS_Settime**(**int** *p*)

int MLaser_Delete(int p ,const wchar_t *name)

The remote file <name> will be deleted inside the laser. Use carefully !

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Delete**(**int** *p*, **string** *name*)

int MLaser_SetDefault(int p ,const wchar_t *name)

Changes the actual printing file to <name>. No extension must be set in <name>. The system will load the selected file <name> for printing.

p: the value of the communication handle assigned by Minit()

specific return values:

0: no error
32: command not accepted due to file not found.

C# function of SocketCommNet.dll

public **int** **CS_SetDefault**(**int** *p*, **string** *name*)

int MLaser_CopyFile(int p ,const wchar_t * sourcefile, const wchar_t * path, unsigned char option)

Function to copy files to/from the laser system. Usually is reserved for further and more complex applications.

sourcefile: local filename to be sent to the laser (with extension!)
path: directory ending with "\\" of location of the source file.
option: copy to/from laser
0: copy to RAMDisk of laser
1: copy to Harddisk of Laser
2: copy from Harddisk of Laser
4: copy from RAMdisk of Laser

p: the value of the communication handle assigned by Minit()

specific return values:

0: no error
2: timeout in receiving data
3: local file could not be opened for reading
4: requested file is empty or does not exist
5: local file cannot be opened for writing
8: file could not be copied to RAMdisk
16: file could not be copied to harddisk
32: file could not be renamed in harddisk

C# function of SocketCommNet.dll

public **int** **CS_CopyFile**(**int** *p*, **string** *filename*, **string** *path*, **byte** *option*)

int MLaser_Status(int p ,PStatus &status)

Fills the status variable <status> with the actual values of the system. Function is used to check the actual status of the laser system.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Status**(**int** *p*, ref **SocketCommNet.SocketComm.CSStatus** *status*)

int MLaser_StatusExt(int p ,PStatusExt &statusext)

Fills the status variable <statusext> with the actual values of the system. Function is used to check the actual status of the laser system.

p: the value of the communication handle assigned by Minit()

Note: this call should only be used when the version number of the firmware is > 39 (returned by MGetVersion()).

C# function of SocketCommNet.dll

public int **CS_StatusExt**(int *p*, ref **SocketCommNet.SocketComm.CSStatusExt** *statusext*)

int MLaser_PrintMode(int p ,Uint32 &mode)

Sets/gets the actual printmode of the laser. The laser can work in 3 different printmodes according the <mode> parameter:

0 : default, single file
1: UMT, user message table mode (file selected according bit selection)
4: Batchjob, files selected according the batchjob table
2: just gets the actual active printmode

The <mode> parameter will be changed after this call according to the actual printmode.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public int **CS_PrintMode**(int *p*, Uint32 *mode*)

int MLaser_Mode(int p ,unsigned char &mode)

Sets the actual mode of the laser. The laser can work in 4 different modes:

0 = static mode:
1= dynamic mode
2= dynamic distance mode
3= dynamic-static mode

If you want to request the actual mode, you have to set the <mode> variable to
8. If the function returns 0, the <mode> variable contains the actual mode.

Note: this function is used only for some special applications where the mode must be switched between static and dynamic.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Mode**(**int** *p*, ref **int** *mode*)

int MLaser_FastUsermessage(int p ,unsigned char field,const char *text)

Sets the string of an internal usermessage field to <text>.

unsigned char field: the field of the internal usermessage, that will be overwritten.

const char *text: a pointer to a string array of max. 127 chars.

p: the value of the communication handle assigned by Minit()

Note: For firmware 5.0.0 and higher the char-array can be as long as 2040 characters.

specific return value:

0: correctly set.
8: buffered usermessage is enabled and buffer is full.
other values: the errorstring is accordingly set.

C# function of SocketCommNet.dll

public **int** **CS_FastASCIIUsermessage**(**int** *p*, **int** *field*, **string** *text*)

The string text should contain only characters with a hex value < 256. No conversion to UTF8 will be done.

int MLaser_GetFastUsermessage(int p ,unsigned char field, char *buf, int &len)

Gets the string of an internal usermessage field and copies up to <len> characters to <buf>, including the terminating '\0' character. <len> is then set to the number of copied characters (including the terminating '\0').

unsigned char field: the field of the internal usermessage,
char *buf: a pointer to a char array provided by the caller.
int &len: the length of the provided buffer.
p: the value of the communication handle assigned by Minit()

specific return value:

0: correctly set.
4: field not correct.
other values: the errorstring is accordingly set.

Note: For firmware 5.0.0 and higher the char-array can be as long as 2040 characters.

C# function of SocketCommNet.dll

public **int** **CS_GetFastASCIIUsermessage**(**int** *p*, **int** *field*, ref **string** *txt*)

Gets the actual content of the field as a sequence of ASCII characters (characters with hexvalue < 256).

int MLaser_FastUTF8Usermessage(int p ,unsigned char field,const char *text)

Sets the UTF8 string of an internal usermessage field to <text>.

In contrast to the MLaser_FastUsermessage() function which codes the string to be sent in ASCII, this function takes an UTF8 string as input. If your message's object has an extended font selected (*.mfx,*.dmx) , the object's input string can be sent as an UTF8 string.

unsigned char field: the field of the internal usermessage, that will be overwritten.

const char *text: a pointer to a string array of max. 2000 chars.

p: the value of the communication handle assigned by Minit()

specific return value:

0: correctly set.
8: buffered usermessage is enabled and buffer is full.
other values: the errorstring is accordingly set.

C# function of SocketCommNet.dll

public **int** **CS_FastUsermessage**(**int** *p*, **int** *field*, **string** *text*)

The string text can be any Unicode string. It will be internally converted to an UTF8 string and sent to the laser as UTF8. Use this function in conjunction with an internal Unicode font (e.g. unicode.unx) if you want to pass international character strings.

int MLaser_GetFastUTF8Usermessage(int p ,unsigned char field, char *buf, int &len)

Gets the string of an internal usermessage field as an utf8-sequence and copies up to <len> characters to <buf>, including the terminating '\0' character. <len> is then set to the number of copied characters (including the terminating '\0').

unsigned char field: the field of the internal usermessage,

char *buf: a pointer to a char array provided by the caller.
int &len: the length of the provided buffer.
p: the value of the communication handle assigned by Minit()

specific return value:

0: correctly set.
    4: field not correct.
    other values: the errorstring is accordingly set.

C# function of SocketCommNet.dll

public **int** **CS_GetFastUsermessage**(**int** *p*, **int** *field*, ref **string** *txt*)

int MLaser_EnableBufferedUM(int p ,int get,int &actsize, int defsize)

Creates/resets an internal buffer for usermessage fields 0 - 35 with size= <defsize>.

int get: if == 0 the buffer is created/resetted
if == 1 the buffer is not changed (is just for receiving the actual size)

int &actsize: receives the actual buffersize

int defsize: defines the buffersize (values 0 – 1000)
when set to 0, the buffered usermessage mode is disabled.

p: the value of the communication handle assigned by Minit()

This command is available for firmware 3.6.2 and higher. The buffer is created/resetted for each field (0 – 35) and works as a FIFO. The laser prints one by one of the entries. The FIFO can be filled with the FastUsermessage command. When the FIFO is full the FastUsermessage command returns the value 8 and the errorstring is set to "Field not set (<fieldnumber>!!!)".
Do not use the Usermessage command when you work with buffered usermessages !!!! Use instead always the Fastusermessage command !!!

C# function of SocketCommNet.dll

```
public int CS_EnableBufferedUM(int p, int get, ref int actsize, int defsize)
```

int MLaser_EnableBufferedUMExt(int p ,int get,int &actsize, int &field, int &fillstatus,int defsize)
From firmware 4.2.9 on available.
Same command as MLaser_EnableBufferedUM() but adds two new parameters for requesting more information when the "get" parameter is set to 1:

int &field: defines the usermessage field in case of "get=1" and "get=2".

From firmware 5.5.0 and in case of "get=0" (enable the buffer):

This variable defines the number of consecutive fields used for buffering. If set to '0' the actual number of fields are used (default: 36 ; for older firmwares 16). This variable will be filled with the actual number of fields used for buffering.

int &fillstatus: receives the number of elements in the FIFO defined by the field.

Note: the parameter "get" has different meanings:

int get: if == 0 the buffer is created and reset.
if == 1   the buffer is not changed (is just for receiving the actual size of the buffer and the fillstatus of a field)
if==2 the buffersize is not changed but the <fillstatus> of <field> is received and AFTERWARDS the FIFO of this field is emptied.

C# function of SocketCommNet.dll

public **int** **CS_EnableBufferedUMExt**(**int** *p*, **int** *get*, ref **int** *actsize*, ref **int** *field*, ref **int** *fillstatus*, **int** *defsize*)

int MLaser_EnableBufferedDataString(int p ,int get,int &actsize, int &field, int &fillstatus,int defsize)

From firmware 5.0.9 on available.

Same command as MLaser_EnableBufferedUMExt(), but instead of a buffered usermessage a buffered datastring is used. A datastring is used to fill the content of customized bitmaps. See the GUI Help for more information.

int &field: defines the datastring field (0,1,2,3)

int &fillstatus: receives the number of elements in the FIFO defined by the field.

Note: additionally, the parameter "get" can have more meanings now:

int get: if == 0 the buffer is created/resetted
if == 1   the buffer is not changed (is just for receiving the actual size of the buffer and the fillstatus of a field)
if==2 the buffersize is not changed but the <fillstatus> of <field> is received and AFTERWARDS the FIFO of this field is emptied.

C# function of SocketCommNet.dll

public **int** **CS_EnableBufferedDataString**(**int** *p*, **int** *get*, ref **int** *actsize*, ref **int** *field*, ref **int** *fillstatus*, **int** *defsize*)

int MLaser_FastDataString(int p ,unsigned char field, const char *in, int len)

Sets the data of an internal datastring field to <in>.

unsigned char field: the field of the internal datastring, that will be overwritten. (values 0,1,2,3)

const char *in: a pointer to a char array of <len> chars.

int len: the length of then char array.

p: the value of the communication handle assigned by Minit()

specific return value:

0: correctly set.
4: field not valid
8: buffered datastring is enabled and buffer is full.
64: buffer overflow (memory access error)
other values: the errorstring is accordingly set.

C# function of SocketCommNet.dll

public **int** **CS_FastDataString**(**int** *p*, **int** *field*, **byte[]** *buf*, **int** *len*)

int MLaser_GetFastDataString(int p ,unsigned char field, char *out, int &len)

Gets the data of an internal datastring field and copies up to <len> characters to <out>. <len> is then set to the number of copied characters.

unsigned char field: the field of the internal datastring,
char *out: a pointer to a char array provided by the caller.
int &len: the length of the provided buffer.
p: the value of the communication handle assigned by Minit()

specific return value:

0: correctly set.
4: field not valid
32: provided buffer too small
64: buffer overflow (memory access error)
other values: the errorstring is accordingly set.

C# function of SocketCommNet.dll

public **int** **CS_GetFastDataString**(**int** *p*, **int** *field*, ref **byte[]** *buf*, ref **int** *len*)

int MLaser_MultipleUsermessage(int p , char *in, int inlen, char *out, int &outlen, int &fields)

Sends multiple usermessages within a single command.

p: the value of the communication handle assigned by Minit()

char *in: a byte array that stores the information of the fields and usermessages. The format has to be:

|field0-byte|ASCII-string|null-byte|field1-byte|ASCII-string\null-byte|…….

int inlen: the size of the in-array in bytes.

char *out: a byte array where the answer of the operation is stored. Its size should be at least the number of fields that are sent within this command.
The out format is as follows:

|field0-return||field1-return||field2-return|..   where the field-return indicates if the corresponding string has been stored. Possible values are
0: string could not be added to the FIFO buffer
1: string set correctly (FIFO buffer is not enabled)
2: string added to the FIFO buffer

int &outlen: an integer reference. <outlen> must be set to the size of the out-arrays. The function fills this parameter with the number of bytes written to the out-array.

int &fields: The function fills this variable with the number of accepted strings within this command.

Note: The total size of the in-array must be smaller than 2040 !

C# function of SocketCommNet.dll

public int **CS_MultipleUsermessage**(Int32 *p*,**Byte[] inbuf.Int32 inlen,Byte[] outbuf, ref Int32 outlen, ref Int32 fields**)

The string text is interpreted as an ASCII byte sequence. No conversion to UTF8 will be done.

int MLaser_MultipleUTF8Usermessage(int p , char *in, int inlen, char *out, int &outlen, int &fields)

Same as the ***Mlaser_MultipleUsermessage*** function, except that all strings are interpreted as UTF8 strings.

C# function of SocketCommNet.dll

public int **CS_MultipleUsermessage**(Int32 *p*,**Byte[] inbuf.Int32 inlen,Byte[] outbuf, ref Int32 outlen, ref Int32 fields**)

int MLaser_GetMultipleUsermessage(int p , char *in, int inlen, char *out, int &outlen)

Gets multiple usermessages within a single command.

p: the value of the communication handle assigned by Minit()

char *in: a byte array that stores the information of the fields to be retrieved. The format has to be:

|field0-byte|field1-byte|…….

int inlen: the size of the in-array in bytes.

char *out: a byte array where the answer of the operation is stored. Its size should be large enough to receive the answer (max. 2040)
The out format is as follows:

|field0|ASCII-string|null-byte|field2|ASCII string|nul-byte....

int &outlen: an integer reference. <outlen> must be set to the size of the out-arrays. The function fills this parameter with the number of bytes written to the out-array.

If the answer would not fit into the out-array then at least terminating null-byte of the field that does not fit within the answer would not be present.

C# function of SocketCommNet.dll

public int **CS_GetMultipleUsermessage**(Int32 *p*,**Byte[] inbuf.Int32 inlen,Byte[] outbuf, ref Int32 outlen**)

The string text is an ASCII byte sequence. No conversion to UTF8 will be done.

int MLaser_GetMultipleUTF8Usermessage(int p , char *in, int inlen, char *out, int &outlen, int &fields)

Same as the ***Mlaser_GetMultipleUsermessage*** function, except that all strings have to be UTF8 strings.

C# function of SocketCommNet.dll

public int **CS_GetMultipleUTF8Usermessage**(Int32 *p*,**Byte[] inbuf.Int32 inlen,Byte[] outbuf, ref Int32 outlen, ref Int32 fields**)

int MLaser_GetFifofield(int p , int field, int index, char *out, int &outlen, int &elements)

Retrieves the string of an element of a FIFO field.

p: the value of the communication handle assigned by Minit()

int field: the field number to be requested.

int index: the index of the FIFO field to be requested. The index is counted from the ‘upper’ side of the FIFO. Index ‘0’ means the last entry that was added to the FIFO.

char *out: a byte array that receives the string of the (field, index) to be retrieved.

int &outlen: the size of the out-array in bytes. The function fills this variable with the written bytes.

int &elements: the function fills this variable with the actual number of elements in the FIFO field.

C# function of SocketCommNet.dll

public **int** **CS_GetFifofield**(**Int32** *p*, **Int32 field, Int32 index, ref string txt, ref Int32 elements**)

int MLaser_GetFifofield(int p , int field, int index, char *out, int &outlen, int &elements)

Same as the ***MLaser_GetFifofield*** function, except that all strings are UTF8 strings.

C# function of SocketCommNet.dll

public **int** **CS_GetUTF8Fifofield**(**Int32** *p*, **Int32 field, Int32 index, ref string txt, ref Int32 elements**)

int MLaser_FifoDump(int p)

The laser dumps all FIFO buffers into a single file (‘umdump.cnf’) in the RAM-disk of the laser. The file can then be retrieved with the MLaser_CopyFile() function.

A dump file can only be created when the system is not in the printing mode or when the system is in alarm-state. If the system is in printing mode and no alarm is active the dump file will not be created and the function returns (32).

Dump file content ('umdump.cnf'):

```
FIFODUMP
(field0,index0): string
(field0,index1): string
(field0,index2): string
(field0,index3): string
(field0,index4): string
....
.....
```

Only the actual elements of all fields are stored inside the file.

int MLaser_SetGlobalCounter (int p ,unsigned char field,const char *counter)

Sets the value of a global internal counter

unsigned char field: the field of the internal counter

const char *counter: a pointer to a '\0' terminated string array containing the counter value.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_SetGlobalCounter**(**int** *p*, **int** *field*, **string** *counter*)

int MLaser_GetGlobalCounter (int p ,unsigned char field, char *counter, int len)

Gets the value of a global internal counter

unsigned char field: the field of the internal counter

char *counter: a pointer to a '\0' terminated string array to where the countervalue should be copied. Be sure to reserve an array that will be big enough (e.g. 48 bytes)

int len: the length of the provided counter array in bytes.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_GetGlobalCounter**(**int** *p*, **int** *field*, ref **string** *counter*)

int MLaser_SetPrivateCounter (int p ,unsigned char field, int repeats, int prints)

Sets the "repeats" and the "prints" of a global internal counter defined by its field number.

```
A global internal counter is a counter identified by a field
number. The "repeats" is the number of prints of an internal
counter without increasing it by the defined "steps". Usually,
"repeats" and "step" is set in the message for each counter
independently. With this command you can change the number of
"repeats" for the internal counters of the actual loaded
message. The counters, whose "repeats" are to be changed are
identified by the "fieldnumber".
Not only the "repeats" are set with this command, but also the
number of prints of this counter. Usually, the number of prints
is zero, and the counter will perform its next increment after
"repeats" prints. If the number of prints are set to a value >
0, the counter will perform its next increment after ("repeats"
- "number of prints") prints and from then on it will increment
after "repeats" prints.
```

unsigned char field: the field of the internal counter

int repeats: the number of prints of an internal counter without increasing the counter.
int prints: the number of prints since the last increment of the internal counter.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public int **CS_SetPrivateCounter**(int *p*, int *field*, int *repeats*, int *prints*)

int MLaser_Offset (int p, int &dx, int &dy, int relative,int format /*=0*/,int reset /*=0*/)

Sets a global offset (dx,dy) for all messages in static (and dynamic) printing.

dx: Offset in x

dy: Offset in y

relative: if set to 0: offset is absolute
if set to 1: offset is relative (relative to any previous offset)

format: (optional for firmware3.6.0 and higher)

0: units are ideal units (100 000 units = Scanfield)

1: units are microns
2: units are 0,1 mm

reset: (optional for firmware3.6.0 and higher)

0: offset valid for all prints
< > 0: offset will be used only for the next print. Any subsequent prints will be printed with zero offset if no new offset is sent to the laser.

p: the value of the communication handle assigned by Minit()

firmware3.5.0 (and higher):

Note: The offset is valid only for static printing. If the offset is absolute, all messages will be shifted by the (dx,dy)-value sent to the laser. To reset the offset, you have to send (0,0) to the laser.

If you send a relative offset, the value you have sent will be added to the actual internal offset.

In both cases, absolute or relative, the function-call fills both variables (dx, dy) with the actual internal global offset, so you can monitor the actual offset.

Units: (dx,dy) have integer values. A value of +50 000 would shift the message by half of the total scanfield. So, the units are scanfield dependent.
No control check will be done when the shift is applied, so the user must assure to stay within the total scanfield.

firmware3.6.0 (and higher):

The offset is valid for static and dynamic printing.

C# function of SocketCommNet.dll

public int **CS_Offset**(int *p*, ref int *dx*, ref int *dy*, int *relative*, int *format*, int *reset*)

int   MLaser_ShiftRotate (int p, float dx, float dy, float angle,float x0, float y0, wchar_t *layername,wchar_t *objectname)

Allows to shift and rotate a message, a layer or an object.

p: the value of the communication handle assigned by Minit()

dx: Offset in x in [mm]

dy: Offset in y in [mm]
x0: x-coordinate of the rotation center in [mm] (invariant point)

y0: x-coordinate of the rotation center in [mm] (invariant point) angle: rotation angle in degrees

layername: if not empty the offset and rotation is only applied to objects within the layer with this name. Note that layernames should not contain any SPACES.

objectname: if not empty the offset and rotation is only applied to the object with this name. Note that objectnames should not contain any SPACES.

Note: this function works only for firmware version 5.3.2 from 15/10/2014 on.

C# function of SocketCommNet.dll

public **int** **CS_ShiftRotate**(**int** *p*, **float** *dx*, **float** *dy*, **float** *angle*, ***float*** *x0*, ***float*** *y0*, ***String*** *layername* , ***String*** *objectame*)

int MLaser_Defocus (int p, int &dz, int relative,int format)

Sets a z-defocus value. The z-defocus value is added to any defocus value of the actual message/layer. After return of this function the value dz will contain the actual absolute z-defocus value in the units according to the 'format' parameter. If you just want to receive the actual value, set a relative offset of dz = 0. For firmware versions > 5.6.4 (version number > 88) you can also set/get the global z-position value. The global z-position value is added to any z-position value of the message/layer.

dz: Offset in Z

relative:

if set to 0: offset value is absolute z-defocus
if set to 1: offset value is relative z-defocus (relative to any previous offset)
if set to 2: offset value is absolute z-position
if set to 3: offset value is relative z-position (relative to any previous offset)

format:

0: units are ideal units (100 000 units = Scanfield)
1: units are microns
2: units are 0,1 mm

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Defocus**(**int** *p*, ref **int** *dz*, **int** *relative*, **int** *format*)

int MLaser_Powerscale (int p, int set, int member,, int &value)
(from firmware version 4.1.0 on)

With this command you can apply a scaling-factor to the power or speed properties of the layers, or to the bitmap properties “pixeltime” and “pixelpower”. The scaling factors are valid for all messages until a reboot of the machine or until a new scaling factor is sent to the laser. The scaling factors are applied always to the original values of the messages !

p: the value of the communication handle assigned by Minit()

set: 0: sets a member’s scaling factor to <value>
1: gets a member’s actual scaling factor and writes it to <value>
member: defines, which scaling factor should be set/get
0: pixel time for bitmap or 2D-printing
1: pixel power for bitmap or 2D-printing (YAG and fiber lasers only)
2: layer speed
3: layer power

value: the scaling factor to be applied in permille. That means, a value of 1000 corresponds to a scaling factor of 1.000, a value of 500 corresponds to a scaling factor of 0.500.

Note that a scaling factor of 500 (= 0.5) applied to the pixel time (member=0) will decrease the printing time, while applied to the layer speed (member=2) it will increase the printing time !!!

C# function of SocketCommNet.dll

```
public int CS_Powerscale(int p, int set, int member, ref int value)
```

int MLaser_SetDynamic (int p ,int var, int &value)

Sets the some dynamic parameters of the laser.

p: the value of the communication handle assigned by Minit()
var: 0: sets the printdistance in micrometer for dynamic-distance mode
1: sets the velocity in mm/min for internal encoder printing
value: the value of the dynamic parameter to be set (printdistance or velocity)

The sent parameters are valid as long the laser's configuration is not actualized or overwritten with any other external application.

C# function of SocketCommNet.dll

public int **CS_SetDynamic**(int *p*, int *var*, ref int *value*)

int MLaser_GetDynamic (int p ,int var, int &value)

Gets the some dynamic parameters of the laser.

p: the value of the communication handle assigned by Minit()
var: 0: gets the printdistance in micrometer for dynamic-distance mode
1: gets the velocity in mm/min for internal encoder printing
value: this variable will be filled with the requested information (printdistance or velocity).

C# function of SocketCommNet.dll

public int **CS_GetDynamic**(int *p*, int *var*, ref int *value*)

int MLaser_AsciiConfig (int p ,const wchar_t *name,int partial/*=0*/)

Loads the configuration parameters stored in the file <name>.
If <name> is an empty string the system will try to load the "sysvars.cnf" from the RAMDISK of the laser. If <name> is not empty and <name> does not begin with a '.' then the system will try to load the <name> file from the HARDDISK of the laser. If you want that the file is loaded from the RAMDISK and the <name> is not "sysvars.cnf" you would have to preceed the filename with ".\ram\" (e.g. <name>. = ".\ram\myconfig.cnf").

p: the value of the communication handle assigned by Minit()
name: the filename of the configuration file
partial: if "0" (default) then the configuration is reset to a defaul configuration before it is loaded from the file.
If "1" then only the parameters that are inside the file will be changed.
If the configuration file contains all parameters of the configuration you might choose a value "0" for the 'partial' parameter. If you send only some parameters within the configuration file you should choose a value "1" for the 'partial' parameter.

specific return value:

0: correctly set.
4: file not found

C# function of SocketCommNet.dll

public **int** **CS_AsciiConfig**(**int** *p*, **string** *name*, **int** *partial*)

int MLaser_StartPrintSession (int p, int ignorealarms)

Prepares the laser for the printing mode. It usually enables the laser for printing, opens an optional shutter, moves the scanner to the optional "keepwarm" position and sets the diode current for YAG systems to the minimum powerlevel. If the laser has the autopointer option activated, this call activates the red pointer in the center of the scanfield. Any subsequent call that results in a loading of a message (MLaser_Start(), MLaser_SetDefault) changes the red pointer to the enclosing square of the loaded message.
This call should usually be used before a MLaser_Start() command is issued.

p: the value of the communication handle assigned by Minit()

int ignorealarms:	value <> 0, then any non-critical alarm will not kick the laser out of the printing mode, once entered into the printing mode with the MLaser_Start() command.
	value = 0, then it depends on internal commandline settings if an alarm kicks the laser out of the printing mode or not.

C# function of SocketCommNet.dll

public **int** **CS_StartPrintSession**(**int** *p*, **int** *ignorealarms*)

int MLaser_EndPrintSession (int p)

Ends a printsession and terminates the printing mode. Sets the outputpower to zero and deactivates the red pointer in case it was activated.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_EndPrintSession**(**int** *p*)

int MLaser_TestPointer (int p, int on)

Activates/deactivates the optional red diode pointer.

p: the value of the communication handle assigned by Minit()
on: 0 deactivates the red pointer
1 activates the red pointer

firmware versions < 5.0.9:

The pointer is activated in the center of the scanfield. If the "REALPOINTER" parameter in the systemvariables is activated, the pointer switches between the center position and the actual loaded marking data.
When <on> is set to "0" this command maps to the MLaser_Stop() command, which stops the redpointer and stops the printing mode.

firmware version >=5.0.9:

The realpointer is activated with the enclosing rectangle of the actual loaded marking data. If "REALPOINTER" is activated, it switches between the enclosing square and the loaded marking data.
When <on> is set to "0" the red pointer is deactivated but the printing mode is not deactivated !

C# function of SocketCommNet.dll

```
public int CS_TestPointer(int p, int on)
```

int MLaser_Eventhandler(int p ,const wchar_t *name)

The remote eventhandler file <name> will be loaded by the laser and the eventhandler will be activated.

p: the value of the communication handle assigned by Minit()

firmware >=5.1.0

If the <name> does not specifiy a valid eventhandler file or a NULL string is passed to this function, the eventhandler will be deactivated.

The function call does not give you feedback about a correct loading of the eventhandler file (you may watch the extended status to see if the eventhandler file is correctly assigned).
The laser first tries to load the file from RAMdisk , then, if not successful, from the harddisk. The eventhandler file must therefore reside either in RAM- or harddisk of the laser.

C# function of SocketCommNet.dll

```
public int CS_Eventhandler(int p, string name)
```

int MLaser_GetFilenames (int p ,const wchar_t *extension, int frame,wchar_t *buf,int &bufsize)

Requests a frame of filenames in the ramdisk with the supplied extension.

p: the value of the communication handle assigned by Minit()

extension: the extension of the files to be requested (e.g. "msf" for binary laser files)

frame: the requested frame number (0 - 255).
For each call a maximum of 25 filenames are sent. To request the first frame set frame to 0, then increment frame until no more data are passed to the provided buffer (bufsize will be set to 0 after the call)

buf: a supplied buffer whose size is given in bufsize. The library call fills this buffer with the requested filenames (maximum 25 per frame). Each filename will be separated by 0x0a (LF).

bufsize: set this value to the size of the supplied buffer. The value represents the number of wchar_t of the buf array reserved by the caller.
After the call the library will set this value to the size of the wchar_t that were copied to the supplied buffer.

firmware >=5.1.6

C# function of SocketCommNet.dll

public **int** **CS_GetFilenames**(**int** *p*, **string** *extension*, **int** *frame*, ref **string** *filenames*)

int MLaser_Store (int p ,int &flags)

Stores the actual internal usermessage and/or counter values in the laser's harddisk.

p: the value of the communication handle assigned by Minit()
flags: determines what is to be stored and is set by the DLL after the call to indicate the caller what was stored.
value: 0x00000001 store the internal usermessages
value: 0x00000002 store the internal global counters
value: 0x00000003 store usermessages and counters

firmware >=5.1.7

C# function of SocketCommNet.dll

public **int** **CS_Store**(**int** *p*, ref **int** *flags*)

int MLaser_MTable (int p)

Force the laser to read the internal "table.cnf" file sued for message table and/or batchjob mode. The "table.cnf" must be existent in the laser's harddisk, is an ASCII file and has the following format:

```
<pos>: <repeat prints> <name>
..
..
..

where <pos> stands for the position (0 - 255) , <repeat prints> stands
for the number of prints of the file in case of the batch job mode,
and <name> stands for the file to be printed (filename with extension
(msf or xml)).
The fields must be separated by a single space !

Example with 3 positions:

0: 1 moert.xml
5: 1 test.msf
7: 1 testfile.xml
```

p: the value of the communication handle assigned by Minit()

firmware >=5.2.2

C# function of SocketCommNet.dll

public **int** **CS_MTable**(**int** *p*)

int MLaser_DumpSVG(int p ,const wchar_t *name)

The file <name> will be created by the laser in the Ramdisk containing the actual message data in svg-format (scalable vector graphics).

p: the value of the communication handle assigned by Minit()

firmware >=5.2.7 (07/02/2013)

The <name> should include the desired file extension and can be a string of up to 16 characters. If the <name> is an empty string, the file “dump.svg” will be created. This command is used to get a preview of the actual loaded message.

C# function of SocketCommNet.dll

public **int** **CS_DumpSVG**(**int** *p*, **string** *name*)

int MLaser_DumpSVGExt(int p ,const wchar_t *name,int filter, const wchar_t *layername)

The file <name> will be created by the laser in the Ramdisk containing the actual message data in svg-format (scalable vector graphics).

p: the value of the communication handle assigned by Minit()

firmware >=5.6.3 (20/11/2020)

The <name> should include the desired file extension and can be a string of up to 16 characters. If the <name> is an empty string, the file "dump.svg" will be created. This command is used to get a preview of the actual loaded message.

filter: allows to filter the object output into the svg filename

0:	all objects are output
1:	all objects in the layer defined by the <layername> are output
2:	objects are output according the actual signalstate and the signalmask settings of the layers

layername: A string up to 28 characters (when the string contains characters other than ASCII characters, the 28 character limit refers to the number of characters when this string is converted to a utf8-string).
It defines the layer's name whose objects should be output to the svg file in case that <filter> is set to 1.

C# function of SocketCommNet.dll

public **int** **CS_DumpSVGExt**(**int** *p*, **string** *name*, ***int*** *filter*, ***string*** *layername*)

int MLaser_Sysinfo(int p ,PSysinfo &info)

Fills the variable <info> with the actual values of the system according to the description of the PSysinfo structure. Values of -1 or INT_MIN typically indicates that the corresponding variable is not available inside the system.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public **int** **CS_Sysinfo**(**int** *p*, ref **SocketCommNet.SocketComm.CSSysinfo** *info*)

int MLaser_Coretemp(int p ,PCoretemp &info)

Fills the variable <info> with the actual values of the system according to the description of the PCoretemp structure. Values of -1 or INT_MIN typically indicates that the corresponding sensor is not available in the system.

p: the value of the communication handle assigned by Minit()

C# function of SocketCommNet.dll

public int **CS_Coretemp**(int *p*, ref SocketCommNet.SocketComm.CSCoretemp *info*)

int MLaser_Signalstate(int p ,int32 get,Uint32 &signalstate,Uint32 &enabled)

Command used to set the signalstate via software instead of using the hardware IO signalstate. Each time the laser starts to print the signalstate is latched and can be used to select specific layers to be printable or not printable. The layers has to be configured correspondingly for this purpose (see GUI help).
With this command you can force the laser to use the sent 'signalstate' as its internal signalstate to determine which layers are printable/not printable.
The hardware signalstate is a 32 bit register with the following meaning:

bit31,bit30,.....photocell2(bit18), photocell(bit17), PLC input(bit16),bit15, bit,14,........bit9,bit8,external selection7,..external selection0.

Thus, with some external IO signals one can control which layer is to be printed, when the layer's settings have been adjusted accordingly.
When you send a 'software-signalstate' the 32-bit value that is sent will act as the new signalstate and the hardware signalstate will be disabled.

p: the value of the communication handle assigned by Minit()

firmware >=5.6.4

get: 0 for retrieving the actual software signalstate and if it is enabled
1 for setting and enabling the software signalstate.
2 for disabling the software signalstate (enabling the hardware signalstate)

signalstate:
The signalstate to be sent. Set it to '0' when you disable or get it. The laser fills this variable with the actual software signalstate.

enabled:
The laser fills this variable with 0 (software signalstate disabled) or 1 (software signalstate disabled).

C# function of SocketCommNet.dll

public **int** **CS_Signalstate**(**int** *p*, **Int32** get, **ref UInt32** signalstate, **ref UInt32** enabled)

How to use the DLL functions ???

Before using any command to communicate with the laser system you have to call the MInit(...) function.
This will open a communication socket with the parameters that you passed to the function. The MInit() call will initialize a value for an internal communication handle and assigns and internal index (int &p) to it. This value must be passed to all following function calls.
This is necessary to identify a specific connection to a lasersystem, e.g. when you work with multiple systems.

After this you must connect to the laser with the function call:

err = MStartClient();

If the connection is established correctly, the function returns 0 else it return <> 0 and you can get a description of the error by a call to MGetLastError().

Remember, that you can set the timeout any time with MSetTimeout(value).

If the MStartClient() function call returns 0, you will have a connection to the laser system and from now on you can communicate with the system. That means, you can make any function call that starts with MLaser_..... (e.g. MLaser_Status(), MLaser_Start(), ...).

When you want to finish a connection to the laser you should first call the MLaser_Knockout() function to tell the laser system that the communciation will be shut down. Then you must call the MFinish() function to close correctly the sockets..

<u>Note:</u> see the SocketCommCPPTest.dsw project.

<u>Notes regarding</u> "Printsession" and "printing mode":

Printing mode:

The laser is in "printing mode", when a valid Start command was sent (MLaser_Start(). The Bit0 of the "Start" byte in the PStatus structure will then be set to "1".

Only when the laser is in printing mode and no alarm is occurring it will start a print as soon as the trigger signal is applied according the configuration settings.

The COMPUTER_READY contact at the customer is closed only when the laser is in printing mode AND no alarm is actually occurring.

As a default setting any alarm will stop the printing mode.

You can overwrite this default behaviour if you set the “-x” command line parameter for the firmware (see GUI Help for more information).

You can also overwrite this behaviour if you issue a MLaser_StartPrintSession() command with “ignorealarms” set to “1”.

In both cases, the laser will not be kicked out of the printing loop (once activated with a start command) in case that a non-critical alarm happens. Critical alarms are usually overtemperature-, initialization-, and memory alarms. Most other alarms (e.g. interlock, shutter, empty message, etc…) are alarms that can automatically be cleared (e.g. closing the interlock, providing correct data, etc…). See the GUI Help for more information.

ANY MLaser_Stop() command will stop the printing mode. Thus, a MLaser_Start() command must be issued again to recover the printing mode.

PrintSession:

The laser starts a “print session” as soon as the MLaser_StartPrintSession() is sent. It basically just prepares the system for printing but does NOT ACTIVATE the printing mode ! If the “autopointer” is activated this function call will activate the red pointer automatically. Bit5 of the Start variable in the status structure indicates the PrintSession status.

A print session is started always with a MLaser_StartPrintSession() command or with a valid MLaser_Start() command (where MLaser_Start() also activates the printing mode).

A print session is terminated only with the MLaser_EndPrintSession() command.

1. <u>ALARM codes of upper WORD of status.err:</u>

Each alarm has assigned a unique number (the alarm code). The meaning of the alarm and its code may differ from system to system.

An alarm can be a hardware alarm or a software alarm. A hardware alarm is an alarm the requires one or more digital inputs on the scanner card. A software alarm does not require a digital input.

For a complete list of the alarm codes see 'alarmcodes.pdf'.