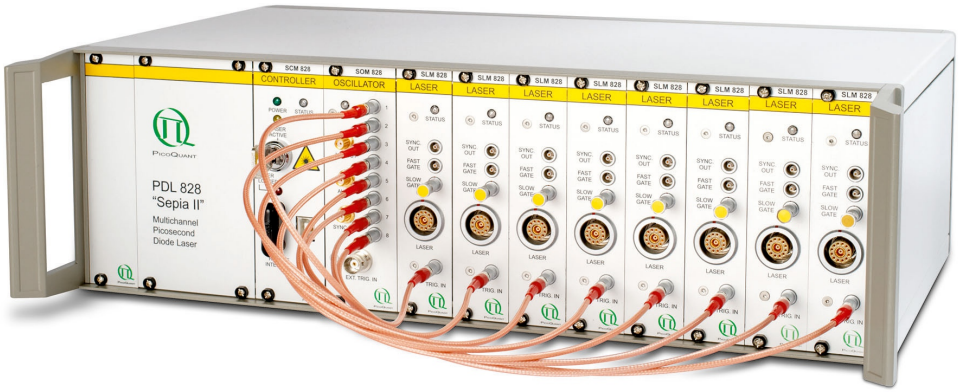
PQ Laser Device

Software Developer's API Library

PicoQuant
Unternehmen für optoelektronische Forschung und Entwicklung

Generic Programming Interface for all
Members of the Family (e.g. Sepia II, Solea…)



Software Developer's Manual

and

Programming Reference Handbook

Version 1.0.1

Table of Contents

1. Introduction

The various computer controlled laser devices from PicoQuant, including among others the PDL 828 "Sepia II", the Solea and the PPL 400, all belong to a family of laser systems based on a common modular architecture, allowing for a variety of different devices and a high degree of flexibility in the configurations possible.

For these devices, all of the dynamic working parameters (e.g. intensity, repetition rate, triggering conditions, wavelength, linewidth...) can be configured from a computer via USB connection. If no change of the working parameters is intended, the device can also run stand alone, i.e. without computer connection. The laser devices from PicoQuant are delivered with a common graphical user interface (GUI) running on Windows™, the Laser Control Software "PQLaserDrv.EXE".

In addition to this, the hereafter documented application programming interface (API) offers the possibility for the end user to build his own dedicated and tailored application. The purpose of this manual is to describe the API and explain all provided functions.

2. PQ Laser Device – Software Developer's Library

You might want to create your own control sequences or graphical user interfaces, to better adapt your PQ Laser Device to your needs and convenience. With the API provided as Windows™ dynamic link library this should be an easy task for an experienced software developer. PQ Laser Devices behave as members of a still growing family, sometimes named after its first member, which was the PDL 828 “Sepia II”. For that reason, the generic API library is called “Sepia2_Lib”, whichever member of the family you might own. Thus the aforementioned dynamic link library is named “Sepia2_Lib.dll” and all API information refer to Sepia II.

The library is provided in two different “flavours”, as x86 (32 bit) and x64 (64 bit) type as well. You can tell which version you see by referring to the file version e.g. from the properties page of the Windows™ Explorer. The major high and low word code the actual software version. In the third part of the version number (a. k. a. “minor high word”), the bit width of the target architecture is encoded. The minor low word is containing the build number. A version number like e.g. “1.1.32.393” stands for the software version 1.1, compiled for an x86 target architecture and coming as build 393, whilst “1.1.64.393” identifies the same software version, but compiled for a x64 target.

With the generic system software (GUI and DLL), we also provide “ready to use” library interfaces in C/C++ and Delphi including language specific declaration files and the import library “Sepia2_Lib.lib”. Developers who use other languages supporting access to DLLs may build their own interfaces analogue to the purchased by simply adapting the declaration files to their desired language and linking their project with the aforementioned import library. It might be necessary to encapsulate the functions-to-call for convenience.

2.1. General Notes on all PQ Laser Device API Functions

All functions exported by Sepia2_Lib.dll commonly behave according to a few conventions. The most important are listed below. Since we implemented the library in C/C++, we chose to document it in the same language. To reduce to the essential, we omitted storage classes, calling conventions and all compiler specific details on the individual function. If you use Pascal, consider that we used true booleans where ever appropriate.

2.1.1. Naming Convention

For names of parameters we use a typed notation in this library. The names of functions commence with the library preamble "SEPIA2_", and a group identifier following. Functions are grouped by the objects, they refer to:

- the library itself ("LIB" functions),
- the communication channel ("USB" functions),
- the main controller and firmware ("FWR" functions),
- all modules on a common level ("COM" functions),
- device operational safety ("SCM" functions).

Additionally to these generic functions, which are supported by any given PQ Laser Device, there exist:

- product model specific module properties, grouped product model by model and module type by type (function names group by **type abbreviation** as given by the COM function DecodeModuleTypeAbbr).

2.1.2. Calling Convention

Consider, all functions use the stdcall calling convention. Refer to the purchased demo code and your compiler specific developer's manuals for more detailed information.

2.1.3. Transferring Arguments Convention and Memory Allocation

The transferring convention for all importing arguments (in the lists below marked with an "I") is "by value" except for strings. For importing string arguments as well as for all exporting arguments (marked with "O"), the transferring convention is "by reference". Bi-directional arguments (marked with "B") can be used for importing as well as exporting arguments, and therefore use the transferring convention "by reference" for either direction. (Use the "var" – clause in Pascal resp. a pointer to the destination variable in C/C++ to implement exports or bi-directionals.) The calling programs have to take care of sufficient memory allocation for exporting arguments. Refer to the C header files for a list of necessary maximal string or array lengths. All strings referred by this document read as strings of 8 bit (ISO-8859) characters, zero terminated, all length information for strings are given as net sizes, so don't forget for the zero termination byte in C/C++.

2.1.4. Return Values

They all return an error code (signed integer, 32 bit).

function returns:	0 :	success
	< 0 :	error

In anticipation of the detailed description, it should be mentioned already here, that the generic library function "**SEPIA2_LIB_DecodeError**" converts any value returned by any library function into a human readable error text.

2.1.5. Running Considerations

Most of the functions need a running PQ Laser Device to work properly. Since the library is already prepared to work with more than one device, you have to identify the addressed device by iDevIdx (0…7), the index of the USB channel it occupies. You could use the Windows Device Manager to find out, which respective value you have to use. You could also more generally build a loop, trying to open devices on all channels and – with respect to the returning error code – compare to the product model or even the serial number of the desired device. But notice that the open device operation establishes an **exclusive** access to the device! You may not open a device, if there is another program having already access to it. However, an application may open more than one device and communicate with them quasi simultaneous; But keep in mind, **the library is not thread-save** by design.

2.2. Common Generic API Functions

This chapter aims on functions, needed with any PQ Laser Device, independent from its product model type.

2.2.1. Library Functions (LIB)

Unlike most of the others, all functions of the LIB group also work “off-line”, without a PQ Laser Device running. They are intended to provide informations on the running conditions of the library itself.

```
/* C/C++ */  int  SEPIA2_LIB_DecodeError          (int             iErrCode,
                                                  char*           cErrorString );
```

arguments: iErrCode I : error code, integer returned from any SEPIA2_…- function call
cErrorString O : error string, pointer to a buffer for at least 64 characters

description: This function is supposed to return an error string (human–readable) associated with a given error code. If <iErrCode> is no member of the legal error codes list, the function returns an error code -9999 itself, which reads "LIB: unknown error code".

```
/* C/C++ */  int  SEPIA2_LIB_GetVersion           (char*           cLibVersion );
```

arguments: cLibVersion O : library version string, pointer to a buffer for at least 12 characters

description: This function returns the current library version string. To be aware of version changing trouble, you should call this function and check the version string in your programs, too. The format of the version string is:

```
<MajorVersion:1>.<MinorVersion:1>.<Target:2>.<Build>
```

where <Target> identifies the word width of the CPU, the library was compiled for. A legal version string could read e.g. “1.1.32.393”, which stands for the software version 1.1, compiled for an x86 target architecture and coming as build 393, whilst “1.1.64.393” identifies the same software version, but compiled for a x64 target.
Take care that at least the first three parts of the version string comply with the expected reference, thus check for compliance of the first 7 characters.

```
/* C/C++ */  int  SEPIA2_LIB_IsRunningOnWine      (unsigned char*  pbRunsOnWine );
```

arguments: pbRunsOnWine O : boolean (pointer to a byte);
true, if running in a Wine environment on a POSIX system

description: This function returns the boolean information if the library is running on Wine, relevant in a case of service. Besides this, this function is solely informative.

2.2.2. Device Communication Functions (USB)

The functions of the USB group handle the PQ Laser Device as an USB device. Besides opening and closing, they provide information on the device and help to identify the desired instance if there is more than one PQ Laser Device connected to the PC.

```
/* C/C++ */  int  SEPIA2_USB_OpenDevice          (int             iDevIdx,
                                                 char*           cProductModel
                                                 char*           cSerialNumber );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
cProductModel	B	:	product model, pointer to a buffer for at least 32 characters
cSerialNumber	B	:	serial number, pointer to a buffer for at least 12 characters

description: On success, this function grants exclusive access to the PQ Laser Device on USB channel <iDevIdx>. It returns the product model and serial number of the device, even if the device is blocked or busy (error code -9004 or -9005; refer to appendix 4.2). If called with non-empty string arguments, the respective string works as condition. If you pass a product model string, e.g. “Sepia II” or “Solea”, all devices other than the specified model are ignored. The analogue goes, if you pass a serial number; Specifying both will work out as a logical AND (“&&” in C-terms) performed on the respective conditions. Thus an error code is returned, if none of the connected devices fit the condition

```
/* C/C++ */  int  SEPIA2_USB_OpenGetSerNumAndClose (int             iDevIdx,
                                                 char*           cProductModel
                                                 char*           cSerialNumber );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
cProductModel	B	:	product model, pointer to a buffer for at least 32 characters
cSerialNumber	B	:	serial number, pointer to a buffer for at least 12 characters

description: When called with empty string parameters given, this function is used to iteratively get a complete list of all currently present PQ Laser Devices. It returns the product model and serial number of the device, even if the device is blocked or busy (error code -9004 or -9005; refer to appendix 4.2). The function opens the PQ Laser Device on USB channel <iDevIdx> non-exclusively, reads the product model and serial number and immediately closes the device again. Don't forget to clear the returned parameter strings if called in a loop. When called with non-empty string parameters, with respect to the conditions, the function behaves as specified for the OpenDevice function.

```
/* C/C++ */  int  SEPIA2_USB_GetStrDescriptor    (int             iDevIdx,
                                                 char*           cDescriptor );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
cSerialNumber	O	:	USB string descriptors, pointer to a buffer for at least 255 characters

description: Returns the concatenated string descriptors of the USB device. For a PQ Laser Device, you could find e.g. the product model string and the firmware build number there, relevant in a case of service. Besides this, this function is solely informative.

```
/* C/C++ */  int  SEPIA2_USB_CloseDevice         (int             iDevIdx );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)

description: Terminates the exclusive access to the PQ Laser Device identified by <iDevIdx>.

2.2.3. Firmware Functions (FWR)

The functions of this group directly access low level structures from the firmware of the PQ Laser Device to initialize the dynamic data layer of the library. Right after opening a PQ Laser Device, any program utilizing this API has to perform a call to the GetModuleMap function, before it can access any module of the laser device.

```
/* C/C++ */  int  SEPIA2_FWR_DecodeErrPhaseName   (int              iErrPhase,
                                                  char*            cErrorPhase );
```

arguments:
- iErrPhase I : error phase, integer returned by firmware function GetLastError
- cErrorPhase O : error phase string, pointer to a buffer for at least 24 characters

description: This function also works “off-line”, without a PQ Laser Device running. It decodes the phase in which an error occurred during the latest firmware start up. Refer to the GetLastError function from the same group below.

```
/* C/C++ */  int  SEPIA2_FWR_GetVersion          (int              iDevIdx,
                                                  char*            cFWVersion );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- cLibVersion O : firmware version string, pointer to a buffer for at least 8 characters

description: This function, in opposite to other GetVersion functions only works “on line”, with the need for a PQ Laser Device running. It returns the actual firmware version string. To be aware of version changing trouble, you should call this function and check the version in your programs, too.

```
/* C/C++ */  int  SEPIA2_FWR_GetLastError        (int              iDevIdx,
                                                  int*             piErrCode,
                                                  int*             piPhase,
                                                  int*             piLocation,
                                                  int*             piSlot,
                                                  char*            cCondition );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- piErrCode O : error code, pointer to an integer
- piPhase O : error phase, pointer to an integer
- piLocation O : error location, pointer to an integer
- piSlot O : error slot, pointer to an integer
- cCondition O : error condition string, pointer to a buffer for at least 55 characters

description: This function returns the error description data from the last start up of the PQ Laser Device's firmware. Decode the error code transferred on <piErrCode> using the function DecodeError from the LIB group. Analogous, use the function DecodeErrPhaseName from the FWR group on <piPhase>. Location and condition can't be decoded and are introduced only for a few phases, but if given, they identify the circumstances of error more detailed.

```
/* C/C++ */  int  SEPIA2_FWR_GetModuleMap        (int              iDevIdx,
                                                  int              iPerformRestart,
                                                  int*             pwModuleCount );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iPerformRestart I : boolean (integer), defines, if a soft restart should precede fetching the map
- pwModuleCount O : current number of PQ Laser Device configurational elements, (pointer to an integer)

description: The map is a firmware and library internal data structure, which is essential to the work with PQ Laser Devices. It will be created by the firmware during start up. The library needs to have a copy of an actual map before you may access any module. You don't need to prepare memory, the function autonomously manages the memory acquirements for this task.
Since the firmware doesn't actualise the map once it is running, you might wish to restart the firmware to assure up to date mapping. You could switch the power off and on again to reach the same goal, but you also could more simply call this function with iPerformRestart set to 1. The PQ Laser Device will perform the whole booting cycle with the tiny difference of not needing to load the firmware again…

```
/* C/C++ */  int  SEPIA2_FWR_GetModuleInfoByMapIdx (int             iDevIdx,
                                                    int             iMapIdx,
                                                    int*            piSlotId,
                                                    unsigned char*  pbIsPrimary,
                                                    unsigned char*  pbIsBackPlane,
                                                    unsigned char*  pbHasUTC );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iMapIdx	I :	index into the map; defines, which module's info is requested
piSlotId	O :	slot number (pointer to an integer) of the module identified by iMapIdx
pbIsPrimary	O :	boolean (pointer to a byte); true, if the index given points to a primary module
pbIsBackPlane	O :	boolean (pointer to a byte); true, if the map index given points to a backplane
pbHasUTC	O :	boolean (pointer to a byte); true, if the map index given points to a module with uptime counter

description: Once the map is created and populated by the function GetModuleMap, you can scan it module by module, using this function. It returns the slot number, which is needed for all module-related functions later on, and three additional boolean information, namely if the module in question is a primary (e. g. laser driver) or a secondary module (e. g. laser head), if it identifies a backplane and furthermore, if the module supports uptime counters.

```
/* C/C++ */  int  SEPIA2_FWR_GetUptimeInfoByMapIdx (int             iDevIdx,
                                                    int             iMapIdx,
                                                    unsigned long*  pulMainPwrUp,
                                                    unsigned long*  pulActivePwrUp,
                                                    unsigned long*  pulScaledPwrUp );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iMapIdx	I :	index into the map; defines, which module's info is requested
pulMainPwrUp	O :	main power up counter value (pointer to an unsigned long) of the module identified by iMapIdx; Divide by 51 to get an approximation of the power up time in minutes
pulActivePwrUp	O :	active power up counter value (pointer to an unsigned long) of the module identified by iMapIdx; Divide by 51 to get an approximation of the active power up time (i.e. laser unlocked) in minutes
pulScaledPwrUp	O :	scaled power up counter value (pointer to an unsigned long) of the module identified by iMapIdx; If it is > 255, divide this value by the active power up counter to get an approximation of the power factor.

description: If the function GetModuleInfoByMapIdx returned true for HasUTC, you can get three counter values using this function. They can be used to roughly calculate the power up times.

```
/* C/C++ */  int  SEPIA2_FWR_FreeModuleMap          (int             iDevIdx );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)

description: Since the library had to allocate memory for the map during the GetModuleMap function, this function is to restitute the memory just before your program terminates. You don't need to call this function between two calls of GetModuleMap for the same device index but you should call it for each device you ever inquired a map during the runtime of your program.

2.2.4. Common Module Functions (COM)

The functions of the COM group are strictly generic and will work on any module you might find plugged to a PQ Laser Device. Except for the functions on presets and updates, they are mainly informative.

```
/* C/C++ */  int  SEPIA2_COM_DecodeModuleType      (int             iModuleType,
                                                   char*           cModuleType );
```

arguments:
- iModuleType I : module type, integer returned by common function GetModuleType
- cModuleType O : module type string, pointer to a buffer for at least 55 characters

description: This function works “off line”, without a PQ Laser Device running. It decodes the module type code returned by the common function GetModuleType and returns the appropriate module type string (ASCII–readable).

```
/* C/C++ */  int  SEPIA2_COM_DecodeModuleTypeAbbr  (int             iModuleType,
                                                   char*           cModTypeAbbr );
```

arguments:
- iModuleType I : module type, integer returned by common function GetModuleType
- cModTypeAbbr O : module type abbr. string, pointer to a buffer for at least 4 characters

description: This function works “off line”, without a PQ Laser Device running, too. It decodes the module type code returned by the common function GetModuleType and returns the appropriate module type abbreviation string (ASCII–readable).

```
/* C/C++ */  int  SEPIA2_COM_GetModuleType         (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iGetPrimary,
                                                   int*            piModuleType );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number, integer (000…989; refer to manual on slot numbers)
- iGetPrimary I : boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
- piModuleType O : module type, pointer to an integer

description: Returns the module type code for a primary or secondary module respectively, located in a given slot.

```
/* C/C++ */  int  SEPIA2_COM_GetSerialNumber       (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iGetPrimary,
                                                   char*           cSerialNumber );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number, integer (000…989; refer to manual on slot numbers)
- iGetPrimary I : boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
- cSerialNumber O : serial number string, pointer to a buffer for at least 12 characters

description: Returns the serial number for a given module.

```
/* C/C++ */  int  SEPIA2_COM_GetPresetInfo          (int             iDevIdx,
                                                    int             iSlotId,
                                                    int             iGetPrimary,
                                                    int             iPresetNr,
                                                    unsigned char*  pbIsSet,
                                                    char*           cPresetMemo );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number, integer (000…989; refer to manual on slot numbers)
iGetPrimary	I	:	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
iPresetNr	I	:	preset number, integer (-1 = factory defaults, 0 = current settings, 1 = preset 1, 2 = preset 2)
pbIsSet	O	:	boolean (pointer to a byte), true, if preset block was already assigned
cPresetMemo	O	:	preset memo, pointer to a buffer for at least 64 characters

description: Returns the preset info identified by iPresetNr for a given module. Initially, the content of preset 1 and preset 2 is not assigned; In this case, the content of pbIsSet will be false (i. e. 0). Additionally, the text stored with the presets when the function “SaveAsPreset” was last invoked for the preset block, is returned in cPresetMemo.

```
/* C/C++ */  int  SEPIA2_COM_RecallPreset           (int             iDevIdx,
                                                    int             iSlotId,
                                                    int             iGetPrimary,
                                                    int             iPresetNr );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number, integer (000…989; refer to manual on slot numbers)
iGetPrimary	I	:	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
iPresetNr	I	:	preset number, integer (-1 = factory defaults, 1 = preset 1, 2 = preset 2)

description: Recalls the preset data as stored in the preset block identified by iPresetNr. Recalling a preset means to overwrite all current settings by the desired ones.
The settings previously active are lost!

```
/* C/C++ */  int  SEPIA2_COM_SaveAsPreset           (int             iDevIdx,
                                                    int             iSlotId,
                                                    int             iGetPrimary,
                                                    int             iPresetNr,
                                                    char*           cPresetMemo );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number, integer (000…989; refer to manual on slot numbers)
iGetPrimary	I	:	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
iPresetNr	I	:	preset number, integer (-1 = factory defaults, 0 = current settings, 1 = preset 1, 2 = preset 2)
cPresetMemo	I	:	preset memo, pointer to a buffer for at least 64 characters

description: Stores the currently active settings into the preset block identified by iPresetNr for a given module. Consider, if presets were already stored in the desired presets block, they will be overwritten without any further request. Don't forget to pass a meaningful text over with the cPresetMemo; It might be working as a remainder to prevent you from an unintentional loss of preset data. Use the GetPresetInfo function to get informed on potential presets already stored in the destination block.

```
/* C/C++ */  int  SEPIA2_COM_GetSupplementaryInfos (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iGetPrimary,
                                                   char*           cLabel,
                                                   char*           cReleaseDate,
                                                   char*           cRevision,
                                                   char*           cMemo );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number, integer (000…989; refer to manual on slot numbers)
iGetPrimary	I :	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
cLabel	O :	internal label string, pointer to a buffer for at least 8 characters
cReleaseDate	O :	release date string, pointer to a buffer for at least 8 characters, format is “YY/MM/DD”
cRevision	O :	revision string, pointer to a buffer for at least 8 characters
cMemo	O :	serial number string, pointer to a buffer for at least 128 characters

description: Returns supplementary string information for a given module. Mainly needed for support...

```
/* C/C++ */  int  SEPIA2_COM_HasSecondaryModule    (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piHasSecondary);
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number, integer (000…989; refer to manual on slot numbers)
piHasSecondary	O :	boolean, (pointer to an integer)

description: Returns if the module in the named slot has attached a secondary one (laser head).

```
/* C/C++ */  int  SEPIA2_COM_IsWritableModule      (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iGetPrimary,
                                                   unsigned char*  pbIsWritable );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number, integer (000…989; refer to manual on slot numbers)
iGetPrimary	I :	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
pbIsWritable	O :	boolean, (pointer to a byte); false, if the memory block is write protected

description: Returns the write protection state of the module's definition, calibration and set-up memory.

```
/* C/C++ */  int  SEPIA2_COM_UpdateModuleData      (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iSetPrimary,
                                                   char*           cDCLFileName );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number, integer (000…989; refer to manual on slot numbers)
iSetPrimary	I :	boolean (integer), defines, if this call concerns a primary (e. g. laser driver) or a secondary module (e. g. laser head) in the given slot
cDCLFileName	I :	file name (coming as windows path), of the binary image of the update data; pointer to a zero-terminated ANSI character buffer

description: Returns the write protection state of the module's definition, calibration and set-up memory.

2.2.5. Device Operational Safety Controller Functions (SCM)

This module implements the safety features of the PQ Laser Device, as there are the thermal and voltage monitoring, the interlock (hard locking) and soft locking capabilities.

```
/* C/C++ */  int  SEPIA2_SCM_GetPowerAndLaserLEDS  (int             iDevIdx,
                                                    int             iSlotId,
                                                    unsigned char*  pbPowerLED,
                                                    unsigned char*  pbLaserActLED);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SCM module
pbPowerLED	O	:	boolean (pointer to a byte), state of the power LED; true : LED is on
pbLaserActLED	O	:	boolean (pointer to a byte), state of the laser active LED; true : LED is on

description: Returns the state of the power LED and the laser active LED.

```
/* C/C++ */  int  SEPIA2_SCM_GetLaserLocked        (int             iDevIdx,
                                                    int             iSlotId,
                                                    unsigned char*  pbLocked );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SCM module
pbLocked	O	:	boolean (pointer to a byte), laser lock state

description: Returns the state of the laser power line. If the line is down either by hardlock (key), power failure or softlock (firmware, GUI or custom program) it returns locked (i. e. true or 1), otherwise unlocked (i. e. false or 0).
Note, that you can't decide for what reason the line is down…

```
/* C/C++ */  int  SEPIA2_SCM_GetLaserSoftLock      (int             iDevIdx,
                                                    int             iSlotId,
                                                    unsigned char*  pbSoftLocked );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SCM module
pbSoftLocked	O	:	boolean (pointer to a byte), contents of the soft lock register

description: Returns the contents of the soft lock register.
Note, that this information will not stand for the real state of the laser power line. A hard lock overrides a soft unlock…

```
/* C/C++ */  int  SEPIA2_SCM_SetLaserSoftLock      (int             iDevIdx,
                                                    int             iSlotId,
                                                    unsigned char   bSoftLocked );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SCM module
bSoftLocked	I	:	boolean (byte), desired value for the soft lock register

description: Sets the contents of the soft lock register.
Note, that this information will not stand for the real state of the laser power line. A hard lock overrides a soft unlock…

2.3. API Functions for “Sepia II” Specific Modules

PQ Laser Devices of the product model “Sepia II” are usually equipped with a SOM 828 type oscillator module in slot 100 and a variable number of SLM 828 laser driver modules in the higher numbered slots. But as the Sepia II is the forebear to the whole family, it could house literally all slot-mountable modules (i.e. all except for the “Solea” modules) designed for the family.

2.3.1. Oscillator Functions (SOM)

As you could already learn from the main manual chapters on the SOM 828 module, this module is a very powerful means for the design of synchronized controlling signals. It incorporates the complex functionalities of a pre-divider, a burst generator, a sequencer and a signal splitter for multiple synchronous outputs in one entity. With the announced but yet to come version SOM 828 D, these features will be enhanced beyond that by an independent delay or a synchronous burst combiner selectively for each individual burst signal output.

```
/* C/C++ */  int  SEPIA2_SOM_DecodeFreqTrigMode    (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iFreqTrigMode,
                                                   char*           cFreqTrigMode );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SOM module
	iFreqTrigMode	I :	index into the list of reference sources, integer (0…4)
	cFreqTrigMode	O :	frequency resp. trigger mode string, pointer to a buffer for at least 32 characters

description: Returns the frequency resp. trigger mode string at list position <iFreqTrigMode> for a given SOM module. This function only works “on line”, with a PQ Laser Device running, because each SOM may carry its individual list of reference sources. Only the list positions 0 and 1 are identical for all SOM modules: They always carry the external trigger option on respectively raising and falling edges. To get the whole table, loop over the list position index starting with 0 until the function terminates with an error.

```
/* C/C++ */  int  SEPIA2_SOM_GetFreqTrigMode       (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piFreqTrigMode);
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SOM module
	piFreqTrigMode	O :	index (pointer to an integer) into the list of reference sources

description: This function inquires the current setting for the reference source in a given SOM. In the integer variable, pointed to by <piFreqTrigMode> it returns an index into the list of possible sources.

```
/* C/C++ */  int  SEPIA2_SOM_SetFreqTrigMode       (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iFreqTrigMode );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SOM module
	iFreqTrigMode	I :	index (integer) into the list of reference sources,

description: This function sets the new reference source for a given SOM. It is passed over as a new value for the index into the list of possible sources.

```
/* C/C++ */  int  SEPIA2_SOM_GetTriggerRange      (int             iDevIdx,
                                                  int             iSlotId,
                                                  int*            piMilliVoltLow,
                                                  int*            piMilliVoltHigh);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
piMilliVoltLow	O	:	pointer to an integer, containing the lower limit of the trigger range
piMilliVoltHigh	O	:	pointer to an integer, containing the upper limit of the trigger range

description: This function gets the adjustable range of the trigger level. The limits are specified in mV.

```
/* C/C++ */  int  SEPIA2_SOM_GetTriggerLevel      (int             iDevIdx,
                                                  int             iSlotId,
                                                  int*            piMilliVolt );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
piMilliVolt	O	:	pointer to an integer, returning the actual value of the trigger level

description: This function gets the current value of the trigger level specified in mV.

```
/* C/C++ */  int  SEPIA2_SOM_SetTriggerLevel      (int             iDevIdx,
                                                  int             iSlotId,
                                                  int             iMilliVolt );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
iMilliVolt	I	:	integer, containing the desired value of the trigger level

description: This function sets the new value of the trigger level specified in mV. To learn about the individual valid range for the trigger level, call GetTriggerRange.
Notice: Since the scale of the trigger level has its individual step width, the value you specified will be rounded off to the nearest valid value. It is recommended to call the GetTriggerLevel function to check the “level in fact”.

```
/* C/C++ */  int  SEPIA2_SOM_GetBurstValues      (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned char*  pbDivider,
                                                  unsigned char*  pbPreSync,
                                                  unsigned char*  pbMaskSync );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
pbDivider	O	:	pointer to a byte, returning the current divider for the pre scaler
pbPreSync	O	:	pointer to a byte, returning the current pre sync value
pbMaskSync	O	:	pointer to a byte, returning the current mask sync value

description: This function returns the current settings of the determining values for the timing of the pre scaler. Refer to the main manual chapter on SOM 828 modules to learn about these values.

```
/* C/C++ */  int  SEPIA2_SOM_SetBurstValues      (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned char   bDivider,
                                                  unsigned char   bPreSync,
                                                  unsigned char   bMaskSync );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
bDivider	I	:	byte (1…255), containing the desired divider for the pre scaler
bPreSync	I	:	byte (0…<bDivider>-1), containing the desired pre sync value
bMaskSync	I	:	byte (0…255), containing the desired mask sync value

description: This function sets the new determining values for the timing of the pre scaler. Refer to the main manual chapter on SOM 828 modules to learn about these values.

```
/* C/C++ */  int  SEPIA2_SOM_GetBurstLengthArray   (int             iDevIdx,
                                                   int             iSlotId,
                                                   long*           plBurstLen1,
                                                   long*           plBurstLen2,
                                                   long*           plBurstLen3,
                                                   long*           plBurstLen4,
                                                   long*           plBurstLen5,
                                                   long*           plBurstLen6,
                                                   long*           plBurstLen7,
                                                   long*           plBurstLen8 );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
plBurstLen1	O	:	channel 1st current burst length (pointer to long int, 0…16777215)
…			
plBurstLen8	O	:	channel 8th current burst length (pointer to long int, 0…16777215)

description: This function gets the current values for the respective burst length of the eight output channels.

```
/* C/C++ */  int  SEPIA2_SOM_SetBurstLengthArray   (int             iDevIdx,
                                                   int             iSlotId,
                                                   long            lBurstLen1,
                                                   long            lBurstLen2,
                                                   long            lBurstLen3,
                                                   long            lBurstLen4,
                                                   long            lBurstLen5,
                                                   long            lBurstLen6,
                                                   long            lBurstLen7,
                                                   long            lBurstLen8 );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
lBurstLen1	I	:	channel 1st desired burst length (long int, 0…16777215)
…			
lBurstLen8	I	:	channel 8th desired burst length (long int, 0…16777215)

description: This function sets the new values for the respective burst length of the eight output channels.

```
/* C/C++ */  int  SEPIA2_SOM_GetOutNSyncEnable     (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned char*  pbOutEnable,
                                                   unsigned char*  pbSyncEnable,
                                                   unsigned char*  pbSyncInverse );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
pbOutEnable	O	:	output channel enable mask, bitcoded (pointer to byte, 0…255)
pbSyncEnable	O	:	sync channel enable mask, bitcoded (pointer to byte, 0…255)
pbSyncInverse	O	:	sync function inverse, boolean (pointer to byte, 0…1)

description: This function gets the current values of the output control and sync signal composing.
(For the following illustrations refer to the screen shot of the main dialogue in the main manual and to the chapter on sync signal composition with SOM 828 modules.)
Each bit in the byte pointed at by <pbOutEnable> stands for an output enable boolean. Thus if all bits are set except of the second and fifth, this byte reads 0xED, which means all but the second and fifth output channel are enabled.
Each bit in the byte pointed at by <pbSyncEnable> stands for an sync enable boolean. Thus if all bits are clear except of the first and third, this byte reads 0x05, which means only the first and third output channel is mirrored to the sync signal composition.
The byte pointed at by <pbSyncInverse> stands for a boolean. It defines whether the sync mask length stands for the count of pulses first let through (bSyncInverse = true, 1) or for the count of pulses first blocked (bSyncInverse = false, 0)

```
/* C/C++ */  int  SEPIA2_SOM_SetOutNSyncEnable     (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned char   bOutEnable,
                                                   unsigned char   bSyncEnable,
                                                   unsigned char   bSyncInverse );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
bOutEnable	I	:	output channel enable mask, bitcoded (byte, 0…255)
bSyncEnable	I	:	sync channel enable mask, bitcoded (byte, 0…255)
bSyncInverse	I	:	sync mask inverse, boolean (byte, 0…1)

description:

This function sets the new values for the output control and sync signal composing.
(For the following illustrations refer to the screen shot of the main dialogue in the main manual and to the chapter on sync signal composition with SOM 828 modules.)
Each bit in the byte <bOutEnable> stands for an output enable boolean. Thus if all bits are set except of the second and fifth, this byte reads 0xED, which means all but the second and fifth output channel are enabled.
Each bit in the byte <bSyncEnable> stands for an sync enable boolean. Thus if all bits are clear except of the first and third, this byte reads 0x05, which means only the first and third output channel is mirrored to the sync signal composition.
The byte <bSyncInverse> stands for a boolean. It defines whether the sync mask length stands for the count of first pulses let through (bSyncInverse = true, 1) or for the count of first pulses blocked (bSyncInverse = false, 0) of each individual burst when composing the sync signal.

```
/* C/C++ */  int  SEPIA2_SOM_DecodeAUXINSequencerCtrl (int          iAUXInCtrl,
                                                      char*        cSequencerCtrl);
```

arguments:

iAUXInCtrl	I	:	sequencer control, integer, taking the byte value as returned by the SOM function GetAUXIOSequencerCtrl
cSequencerCtrl	O	:	sequencer control string, pointer to a buffer for at least 24 characters

description:

This function works “off line”, without a PQ Laser Device running, too. It decodes the sequencer control code returned by the SOM function GetAUXIOSequencerCtrl and returns the appropriate sequencer control string (ASCII–readable).

```
/* C/C++ */  int  SEPIA2_SOM_GetAUXIOSequencerCtrl (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned char*  pbAUXOutCtrl,
                                                   unsigned char*  pbAUXInCtrl );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
pbAUXOutCtrl	O	:	true, if sequence index pulse is enabled on AUX OUT, bool. (byte, 0…1)
pbAUXInCtrl	O	:	current restarting condition of the sequencer, (pointer to byte, 0…2)

description:

This function gets the current control values for AUX OUT and AUX IN.
The byte pointed at by <pbAUXOutCtrl> stands for a boolean “sequence index pulse enabled on AUX Out”. The value of the byte pointed at by <pbAUXInCtrl> stands for the current running/restart mode of the sequencer. The user can decode this value to a human readable string using the DecodeAUXINSequencerCtrl function. The sequencer knows three modes:

0 : free running,
1 : running / restarting, if AUX IN is on logical High level,
2 : running / restarting, if AUX IN is on logical Low level.

```
/* C/C++ */  int  SEPIA2_SOM_SetAUXIOSequencerCtrl (int             iDevIdx,
                                                    int             iSlotId,
                                                    unsigned char   bAUXOutCtrl,
                                                    unsigned char   bAUXInCtrl );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SOM module
bAUXOutCtrl	I	:	boolean byte; if true, sequence index pulse is enabled on AUX OUT
bAUXInCtrl	I	:	controls the restarting condition of the sequencer, (pointer to byte, 0…2)

description: This function sets the current control values for AUX OUT and AUX IN.
The byte given by <bAUXOutCtrl> stands for a boolean “sequence index pulse enabled on AUX Out”. The value of the byte <bAUXInCtrl> stands for the intended running/restart mode of the sequencer. The user can decode this value to a human readable string using the DecodeAUXINSequencerCtrl function. Refer to the sequencer modes as described at SOM function GetAUXIOSequencerCtrl.

2.3.2. Laser Driver Functions (SLM)

SLM 828 modules can interface the huge families of pulsed laser diode heads (LDH series) and pulsed LED heads (PLS series) from PicoQuant. These functions let the application control their working modes and intensity.

```
/* C/C++ */  int  SEPIA2_SLM_DecodeFreqTrigMode    (int             iFreq,
                                                   char*           cFreqTrigMode );
```

arguments:

iFreq	I :	index into the list of int. frequencies/ext. trigger modi, integer (0…7)
cFreqTrigMode	O :	frequency resp. trigger mode string, pointer to a buffer for at least 28 characters

description: Returns the frequency resp. trigger mode string at list position <iFreq> for any SLM module. This function also works “off line”, since all SLM modules provide the same list of int. frequencies resp. ext. trigger modi.

```
/* C/C++ */  int  SEPIA2_SLM_DecodeHeadType       (int             iHeadType,
                                                   char*           cHeadType );
```

arguments:

iHeadType	I :	index into the list of pulsed LED / laser head types, integer (0…3)
cHeadType	O :	head type string, pointer to a buffer for at least 18 characters

description: Returns the head type string at list position <iHeadType> for any SLM module. This function also works “off line”, since all SLM modules provide the same list of pulsed LED / laser head types.

```
/* C/C++ */  int  SEPIA2_SLM_GetIntensityFineStep (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned short* pwIntensity );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
pwIntensity	O :	intensity (as per mille of the ctrl. voltage; pointer to word, 0…1000)

description: This function gets the current intensity value of a given SLM driver module: The word pointed at by <pwIntensity> stands for the current per mille value of the laser head controlling voltage.

```
/* C/C++ */  int  SEPIA2_SLM_SetIntensityFineStep (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned short  wIntensity );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
wIntensity	I :	intensity (as per mille of the ctrl. voltage; pointer to word, 0…1000)

description: This function sets the intensity value of a given SLM driver module:
The word <wIntensity> stands for the desired per mille value of the laser head controlling voltage.

```
/* C/C++ */  int  SEPIA2_SLM_GetPulseParameters    (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piFreq,
                                                   unsigned char*  pbPulseMode,
                                                   int*            piHeadType );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
piFreq	O :	index into list of frequencies/trigger modi (pointer to integer, 0…7)
pbPulseMode	O :	pulse enabled, boolean (pointer to byte, 0…1)
piHeadType	O :	index into list of pulsed LED/laser head types (pointer to byte, 0…3)

description: This function gets the current pulse parameter values of a given SLM driver module:
The integer pointed at by <piFreq> stands for an index into the list of int. frequencies / ext. trigger modi. Decode this value using the function DecodeFreqTrigMode
The byte pointed at by <pbPulseMode> stands for a boolean and may be read as follows: 1 : “pulses enabled”; 0 : either “laser off” or “continuous wave”, depending on the capabilities of the used head.
The integer pointed at by <piHeadType> stands for an index into the list of pulsed LED / laser head types. Decode this value using the SLM function DecodeHeadType.

```
/* C/C++ */  int  SEPIA2_SLM_SetPulseParameters   (int            iDevIdx,
                                                  int            iSlotId,
                                                  int            iFreq,
                                                  unsigned char  bPulseMode );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
iFreq	I :	index into list of frequencies/trigger modi (integer, 0…7)
bPulseMode	I :	pulse enabled, boolean (byte, 0…1)

description:

This function gets the current pulse parameter values of a given SLM driver module:
The integer <iFreq> stands for an index into the list of int. frequencies / ext. trigger modi. Decode this value using the function DecodeFreqTrigMode
The byte <bPulseMode> stands for a boolean and may be read as follows: 1 : "pulses enabled"; 0 : either "laser off" or "continuous wave", depending on the capabilities of the used head.

```
/* C/C++ */  int  SEPIA2_SLM_GetParameters      (int            iDevIdx,
                                                 int            iSlotId,
                                                 int*           piFreq,
                                                 unsigned char* pbPulseMode,
                                                 int*           piHeadType,
                                                 unsigned char* pbIntensity );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
piFreq	O :	index into list of frequencies/trigger modi (pointer to integer, 0…7)
pbPulseMode	O :	pulse enabled, boolean (pointer to byte, 0…1)
piHeadType	O :	index into list of pulsed LED/laser head types (pointer to byte, 0…3)
pbIntensity	O :	intensity (as percentage of ctrl. voltage; pointer to byte, 0…100)

description:

deprecated, instead use
SEPIA2_SLM_GetIntensityFineStep,
SEPIA2_SLM_GetPulseParameters

This function gets the current values of a given SLM driver module:
The integer pointed at by <piFreq> stands for an index into the list of int. frequencies / ext. trigger modi for SLM modules. The byte pointed at by <pbPulseMode> stands for a boolean and may be read as follows: 1 : "pulses enabled"; 0 : either "laser off" or "continuous wave", depending on the capabilities of the used head. The integer pointed at by <piHeadType> stands for an index into the list of pulsed LED / laser head types. The byte pointed at by <pbIntensity> stands for the current percentage of the laser head controlling voltage.

```
/* C/C++ */  int  SEPIA2_SLM_SetParameters      (int            iDevIdx,
                                                 int            iSlotId,
                                                 int            iFreq,
                                                 unsigned char  bPulseMode,
                                                 unsigned char  bIntensity );
```

arguments:

iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SLM module
iFreq	I :	index into list of frequencies/trigger modi (integer, 0…7)
bPulseMode	I :	pulse enabled, boolean (byte, 0…1)
bIntensity	I :	intensity (as percentage of ctrl. voltage; byte, 0…100)

description:

deprecated, instead use
SEPIA2_SLM_SetIntensityFineStep,
SEPIA2_SLM_SetPulseParameters

This function gets the current values of a given SLM driver module:
The integer <iFreq> contains the new index into the list of int. frequencies / ext. trigger modi for SLM modules. The byte <bPulseMode> contains the new pulse mode (boolean) and may be read as follows: 1 : "pulses enabled"; 0 : either "laser off" or "continuous wave", depending on the capabilities of the used head. The byte <bIntensity> contains the desired percentage of the laser head controlling voltage.

2.3.3. Multi Laser Driver Functions (SML)

In contrast to the SLM modules, the SML 828 multi laser driver module is not only generating the controlling voltage for an external laser head, but houses up to four laser diodes itself. These lasers are synchronized and combining coupled to a common output fiber, thus enhancing the optical output power of the individual lasers.

```
/* C/C++ */  int  SEPIA2_SML_DecodeHeadType      (int             iHeadType,
                                                 char*           cHeadType );
```

arguments:

iHeadType	I :	index into the list of pulsed LED / laser head types, integer (0…2)
cHeadType	O :	head type string, pointer to a buffer for at least 18 characters

description: Returns the head type string at list position <iHeadType> for any SML module. This function also works “off line”, since all SML modules provide the same list of pulsed LED / laser head types.

```
/* C/C++ */  int  SEPIA2_SML_GetParameters      (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned char*  pbPulseMode,
                                                 int*            piHead,
                                                 unsigned char*  pbIntensity );
```

arguments:

iDevIdx	I :	Sepia II device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SML module
pbPulseMode	O :	pulse enabled, boolean (pointer to byte, 0…1)
piHead	O :	index into list of pulsed LED/laser head types (pointer to byte, 0…2)
pbIntensity	O :	intensity (as percentage of optical power; pointer to byte, 0…100)

description: This function gets the current values of a given SML multi lasers driver module:
The byte pointed at by <pbPulseMode> stands for a boolean and may be read as follows:
1 : “pulses enabled”; 0 : “continuous wave”.
The integer pointed at by <piHead> stands for an index into the list of pulsed LED / laser head types
The byte pointed at by <pbIntensity> stands for the current percentage of the optical power of the laser heads.

```
/* C/C++ */  int  SEPIA2_SML_SetParameters      (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned char   bPulseMode,
                                                 unsigned char   bIntensity );
```

arguments:

iDevIdx	I :	Sepia II device index (USB channel number, 0…7)
iSlotId	I :	slot number of a SML module
bPulseMode	I :	pulse enabled, boolean (byte, 0…1)
bIntensity	I :	intensity (as percentage of the optical power; byte, 0…100)

description: This function gets the current values of a given SML driver module:
The byte <bPulseMode> contains the new pulse mode (boolean) and may be read as follows:
1 : “pulses enabled”; 0 : “continuous wave”.
The byte <bIntensity> contains the desired percentage of the optical power of the laser heads.

2.4. API Functions for “PPL 400” Specific Modules

PicoQuant's “Programmable Pulse Shape Laser Device” PPL 400 combines the already illustrated features of the SOM 828, allowing for variable sequences of burst pulses, with up to two of the new specialized waveform generation modules SWM 828. The output curves of these modules may be combined to be the modulating input of the VCL 828 voltage controlled laser module, where the latter is an integrated constant factor amplifier and laser modulator, thus without the need for an own set of API functions. The complete magic is provided by the SWM module(s).

2.4.1. Waveform Generation Module Functions (SWM)

Each SWM module can generate two independent scalable curves. All timing parameters of these curves are defined in per mille with respect to the individual curve's time base. At the output of the SWM module, the independent signals are overlaid to the sum of the curves. The following restrictions have to be taken into account:

For a given curve, the start point of the ramp must always lie on or behind the start point of the pulse.

```
/* C/C++ */  int  SEPIA2_SWM_DecodeRangeIdx      (int             iDevIdx,
                                                 int             iSlotId,
                                                 int             iTimeBaseIdx,
                                                 int*            piUpperLimit );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SLM module
	iTimeBaseIdx	I :	index into the list of time bases
	piUpperLimit	O :	upper limit of the range (pointer to an integer) in nsec
description:	This function returns the upper limit of the time base identified by iTimeBaseIdx in nano seconds.		

```
/* C/C++ */  int  SEPIA2_SWM_GetUIConstants      (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned char*  pbTimeBasesCnt
                                                 unsigned short* pwMaxAmplitude,
                                                 unsigned short* pwMaxSlewRate,
                                                 unsigned short* pwExpRampFctr,
                                                 unsigned short* pwMinUsrValue,
                                                 unsigned short* pwMaxUsrValue,
                                                 unsigned short* pwUserRes );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SLM module
	pbTimeBasesCnt	O :	count of entries in the list of time bases (pointer to a byte)
	pwMaxAmplitude	O :	maximum pulse amplitude (pointer to a word) in mV
	pwMaxSlewRate	O :	maximum ramp slew rate (pointer to a word) in mV/µsec
	pwExpRampFctr	O :	exponential factor for the ramp (pointer to a word)
	pwMinUsrValue	O :	lower limit for user entries (pointer to a word) in ‰
	pwMaxUsrValue	O :	upper limit for user entries (pointer to a word) in ‰
	pwUserRes	O :	user resolution i.e. stepwidth for user entries (pointer to a word) in ‰
description:	This function returns all necessary values to initialize a GUI for all legal entries.		

```
/* C/C++ */  int  SEPIA2_SWM_GetCurveParams      (int             iDevIdx,
                                                 int             iSlotId,
                                                 int             iCurveIdx,
                                                 unsigned char*  pbTimeBaseIdx,
                                                 unsigned short* pwPAPml,
                                                 unsigned short* pwRRPml,
                                                 unsigned short* pwPSPml,
                                                 unsigned short* pwRSPml,
                                                 unsigned short* pwWSPml );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SLM module
	iCurveIdx	I :	curve number (0; 1)
	pbTimeBaseIdx	O :	index into the list of time bases (pointer to a byte)
	pwPAPml	O :	pulse amplitude (pointer to a word, 0..1000) in ‰ of the max. amplitude
	pwRRPml	O :	ramp slew rate (pointer to a word, 0..1000) in ‰ of the max. slew rate
	pwPSPml	O :	pulse start delay (pointer to a word, 0..1000) in ‰ of the time base
	pwRSPml	O :	ramp start delay (pointer to a word, 0..1000) in ‰ of the time base
	pwWSPml	O :	wave stop delay (pointer to a word, 0..1000) in ‰ of the time base

description: This function returns the describing parameters of the curve identified by iCurveIdx.

```
/* C/C++ */  int  SEPIA2_SWM_SetCurveParams      (int             iDevIdx,
                                                 int             iSlotId,
                                                 int             iCurveIdx,
                                                 unsigned char   bTimeBaseIdx,
                                                 unsigned short  wPAPml,
                                                 unsigned short  wRRPml,
                                                 unsigned short  wPSPml,
                                                 unsigned short  wRSPml,
                                                 unsigned short  wWSPml );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SLM module
	iCurveIdx	I :	curve number (0; 1)
	bTimeBaseIdx	I :	index into the list of time bases (byte)
	wPAPml	I :	pulse amplitude (word, 0..1000) in ‰ of the max. amplitude
	wRRPml	I :	ramp slew rate (word, 0..1000) in ‰ of the max. slew rate
	wPSPml	I :	pulse start delay (word, 0..1000) in ‰ of the time base
	wRSPml	I :	ramp start delay (word, 0..1000) in ‰ of the time base
	wWSPml	I :	wave stop delay (word, 0..1000) in ‰ of the time base

description: This function sets the describing parameters for the curve identified by iCurveIdx.

```
/* C/C++ */  int  SEPIA2_SWM_GetCalTableVal      (int             iDevIdx,
                                                 int             iSlotId,
                                                 char*           cTableName
                                                 unsigned char   bTableRow
                                                 unsigned char   bTableColumn
                                                 unsigned short* pwValue );
```

arguments:	iDevIdx	I :	PQ Laser Device index (USB channel number, 0…7)
	iSlotId	I :	slot number of a SWM module
	cTableName	I :	pointer to a character buffer, containing the name of the table
	bTableRow	I :	byte, containing the table row (zero based) to be addressed
	bTableColumn	I :	byte, containing the table column (zero based) to be addressed
	pwValue	O :	pointer to a word, returning the calibration value as read from the table

description: Returns the content of a given cell of the internal calibration tables. The tables are identified by their names, the index into the table (line-number) is zero based. The most important table is “UI_Consts”, because it is the only fixed length table, but contains the lengths of all the other tables. The function is needed for documentation of the module's calibration parameters in case of a support request, beside this, it is solely informative.

Table Name	*Col.*	*Content*	*Unit(s)*	*Table Length*
UI_Consts	1	Constants for the user interface:		13
		00. DAC resolution	[bit]	
		01. min. user value	[‰]	
		02. max. user value	[‰]	
		03. user resolution	[‰]	
		04. max. pulse amplitude	[mV]	
		05. max. slew rate	[mV/µs]	
		06. exponential ramp effect		
		07. table length: number of time bases		
		08. table length: number of pulse cal-points		
		09. table length: number of ramp cal-points		
		10. table length: number of delay cal-points (tb1)		
		11. table length: number of delay cal-points (tb2)		
		12. table length: number of delay cal-points (tb3)		

Having read this table, we know the lengths of all other tables and may begin to read them:

Table Name	*Col.*	*Content*	*Unit(s)*	*Table Length*
TaW1TRng	1	Wave 1: Timebase upper range	[ns]	UI_Consts [7]
TaW1TStw	1	Wave 1: Timebase start value	[mV]	UI_Consts [7]
TaW1TSlr	1	Wave 1: Timebase slew rate	[mV/µs]	UI_Consts [7]
TaW2TRng	1	Wave 2: Timebase upper range	[ns]	UI_Consts [7]
TaW2TStw	1	Wave 2: Timebase start value	[mV]	UI_Consts [7]
TaW2TSlr	1	Wave 2: Timebase slew rate	[mV/µs]	UI_Consts [7]
TaW1PAmp	2	Wave 1: Pulse amplitude DAC calibration	([‰] | [a.u.])	UI_Consts [8]
TaW1PDyn	2	Wave 1: Pulse dynamics DAC calibration	([‰] | [a.u.])	UI_Consts [8]
TaW1RSlr	2	Wave 1: Ramp slew rate DAC calibration	([‰] | [a.u.])	UI_Consts [9]
TaW2PAmp	2	Wave 2: Pulse amplitude DAC calibration	([‰] | [a.u.])	UI_Consts [8]
TaW2PDyn	2	Wave 2: Pulse dynamics DAC calibration	([‰] | [a.u.])	UI_Consts [8]
TaW2RSlr	2	Wave 2: Ramp slew rate DAC calibration	([‰] | [a.u.])	UI_Consts [9]

The other tables hold the delay calibration (wave-shape timing correction) and could be understood as as many huge tables as there are timebases (as given in UI_Consts [7]), each of these tables with 7 columns and as many rows as given in the corresponding field of UI_Consts (i.e. UI_Consts [10+tb_idx] with tb_idx going from 0 to UI_Consts [7] – 1). All delay values are arbitrary values: They stand for DAC values, which compensate for proper timing.

Table Name	*Col.*	*Content*	*Unit(s)*	*Table Length*
TaPmlDly0	1	UI argument for DAC values (tb_idx==0)	[‰]	UI_Consts [10]
TaW1PDly0	1	Wave 1: Pulse start delay with timebase 1	[a.u.]	UI_Consts [10]
TaW1RDly0	1	Wave 1: Ramp start delay with timebase 1	[a.u.]	UI_Consts [10]
TaW1SDly0	1	Wave 1: Wave stop delay with timebase 1	[a.u.]	UI_Consts [10]
TaW2PDly0	1	Wave 2: Pulse start delay with timebase 1	[a.u.]	UI_Consts [10]
TaW2RDly0	1	Wave 2: Ramp start delay with timebase 1	[a.u.]	UI_Consts [10]
TaW2SDly0	1	Wave 2: Wave stop delay with timebase 1	[a.u.]	UI_Consts [10]

Table Name	*Col.*	*Content*	*Unit(s)*	*Table Length*
TaPmlDly1	1	UI argument for DAC values (tb_idx==1)	[‰]	UI_Consts [11]
TaW1PDly1	1	Wave 1: Pulse start delay with timebase 2	[a.u.]	UI_Consts [11]
TaW1RDly1	1	Wave 1: Ramp start delay with timebase 2	[a.u.]	UI_Consts [11]
TaW1SDly1	1	Wave 1: Wave stop delay with timebase 2	[a.u.]	UI_Consts [11]
TaW2PDly1	1	Wave 2: Pulse start delay with timebase 2	[a.u.]	UI_Consts [11]
TaW2RDly1	1	Wave 2: Ramp start delay with timebase 2	[a.u.]	UI_Consts [11]
TaW2SDly1	1	Wave 2: Wave stop delay with timebase 2	[a.u.]	UI_Consts [11]

Table Name	*Col.*	*Content*	*Unit(s)*	*Table Length*
TaPmlDly2	1	UI argument for DAC values (tb_idx==2)	[‰]	UI_Consts [12]
TaW1PDly2	1	Wave 1: Pulse start delay with timebase 3	[a.u.]	UI_Consts [12]
TaW1RDly2	1	Wave 1: Ramp start delay with timebase 3	[a.u.]	UI_Consts [12]
TaW1SDly2	1	Wave 1: Wave stop delay with timebase 3	[a.u.]	UI_Consts [12]
TaW2PDly2	1	Wave 2: Pulse start delay with timebase 3	[a.u.]	UI_Consts [12]
TaW2RDly2	1	Wave 2: Ramp start delay with timebase 3	[a.u.]	UI_Consts [12]
TaW2SDly2	1	Wave 2: Wave stop delay with timebase 3	[a.u.]	UI_Consts [12]

2.5. API Functions for “Solea” Specific Modules

On the first glance, PicoQuant's tunable laser device “Solea” looks rather monolithic than modular, but from the engineer's point of view however, it is designed and built highly modular. In spite of a very different external look of the two devices, the Solea is based on the same modular structure as the PDL 828 “Sepia II”, as far as device communication and controlling are concerned. In fact, Solea can even be driven as a group of external slave modules controlled by another Sepia II master.

This casts a new light on the first parameters of any module oriented function. The same Solea will be addressed by a different device index, when either driven as stand alone device or driven as a couple of slave modules of a Sepia II master. Even more, the slot identification numbers <iSlotId> of the Solea modules will differ, too, since in the first case they are calculated with respect to the Sepia II slot where the extension module is plugged in and thus will increment in tenner or even units, whilst in the later case the same modules show up as first level modules, all numbered in hundreds. A software that shall be run under both conditions therefore has to retrieve the actual slot ID for any Solea module by inspecting the device-internal list of modules, called the “map” with the firmware (FWR) function GetModuleInfoByMapIdx.

2.5.1. A Prior Note on Timing and Termination of “Solea” API Functions

In opposite to what we are used to experience from “Sepia II” API functions, some of the following function calls will return, although the desired state isn't established yet. This is due to the fact, that these functions desire computational and mechanical activities, that take much more time than a common USB vendor command is allowed to last. On the other hand we expect our API functions to return a result code on termination, telling us whether the desired state was successfully reached or not. Obviously, this is a significant, mutual contradiction.

In reaction on this conflict, we decided to modify the paradigm of the return code. For these potentially critical, time consumptive functions, the return code doesn't state on the termination and thus attendance to receive the next instruction but on the reception, error free interpretation and queueing of the command. Immediately, the internal **busy state** of the module in question is set, while the module autonomously completes the intended actions. The module will reject any further commands until this state was successfully cleared or otherwise changes on termination into a signalling **error pending** state. This state is prohibiting the reception of a new command, too. It stays active until the state and error code was read. Polling the state until not longer busy and reading an eventual error code afterwards is all done with the **GetStatusError** function of the respective module.

2.5.2. Seed Laser Module Functions (SSM)

The SSM module controls the seed laser of the Solea. For the user API, it provides functions to control the working mode with respect to the triggering of the laser. For internal calibrating use, it provides a set of abstract data, gathered in a write protected FRAM device. Anyway, it could come to the need of updating these data. For this purpose, the module provides functions to read and even alter the write protection state. Consider, that changes to this state aren't stored in the system and thus set back to protective after each power up.

```
/* C/C++ */  int  SEPIA2_SSM_DecodeFreqTrigMode    (int             iDevIdx,
                                                   int             iSlotId,
                                                   int             iFreqTrigIdx,
                                                   char*           cFreqTrig,
                                                   int*            piFreq,
                                                   byte*           pbTrigLevelEna);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
iFreqTrigIdx	I	:	index into the list of reference sources (integer, 0…iMaxIdx)
cFreqTrig	O	:	string representation of the frequency / trigger mode, pointer to a buffer for at least 15 characters
piFreq	O	:	numeric representation of the frequency / trigger mode in Hz, (pointer to an integer); 0 = off, -1 = external source
pbTrigLevelEna	O	:	boolean (pointer to byte) denoting if a trigger level is needed

description: Returns the frequency / trigger mode properties at the list position given by <iFreqTrigIdx> for a SSM module. The properties to retrieve are:

- a string representation of the frequency / trigger mode in <cFreqTrig>,
- a numerical representation thereof in <piFreq> in Hz (0 means off, -1 means external),
- a boolean in <pbTrigLevelEna>, denoting if the trigger mode needs additional trigger level information.

This function only works “on line”, with a “Solea” running, because each SSM may carry its individual list of reference sources. To get the whole table, loop over the list position index starting with 0 until the function terminates with an error.

```
/* C/C++ */  int  SEPIA2_SSM_GetTrigLevelRange     (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piUpperTL,
                                                   int*            piLowerTL,
                                                   int*            piResolTL );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
piUpperTL	O	:	upper trigger level (pointer to an integer) in mV
piLowerTL	O	:	upper trigger level (pointer to an integer) in mV
piResolTL	O	:	trigger level resolution (pointer to an integer) in mV

description: Retrieves the range and resolution of the trigger level in mV (needed as limits for adjustment controls, e.g. in the GUI)

```
/* C/C++ */  int  SEPIA2_SSM_GetTriggerData       (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piFreqTrigIdx,
                                                   int*            piTrigLevel );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
piFreqTrigIdx	O	:	index (pointer to an integer) into the list of reference sources,
piTrigLevel	O	:	pointer to an integer, returning the current value of the trigger level in mV

description: Returns the current index into the list of reference sources in <piFreqTrigIdx>; This value can be decoded using the function DecodeFreqTrigMode. Additionally, it returns the current trigger level in mV, if <piFreqTrigIdx> contains a value representative for external triggering.

```
/* C/C++ */  int  SEPIA2_SSM_SetTriggerData      (int             iDevIdx,
                                                 int             iSlotId,
                                                 int             iFreqTrigIdx,
                                                 int             iTrigLevel );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
iFreqTrigIdx	I	:	index into the list of reference sources (integer, 0…)
iTrigLevel	I	:	integer, giving the desired value of the trigger level in mV

description: Sets the current index into the list of reference sources in <piFreqTrigIdx>; This value can be decoded using the function DecodeFreqTrigMode. Additionally, it sets the current trigger level in mV, if <iFreqTrigIdx> contains a value representative for external triggering.

```
/* C/C++ */  int  SEPIA2_SSM_SetFRAMWriteProtect  (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned char   bWriteProtect );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
bWriteProtect	I	:	enable write protection, boolean (byte, 0…1)

description: Sets the write protection for the module's FRAM to the desired value. If protection was disabled, the FRAM stays writeable until revoked or next power down. On power up, write protection is set by default.

```
/* C/C++ */  int  SEPIA2_SSM_GetFRAMWriteProtect  (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned char*  pbWriteProtect);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SSM module
pbWriteProtect	O	:	is write protection enabled, boolean (pointer to byte)

description: Gets the write protection state for the module's FRAM.

2.5.3. Wavelength Selector Functions (SWS)

The SWS wavelength selector module is a kind of “intelligent” module, supported by its own processor. Its tasks are more complex than other modules and changes to the state of this module may elapse much more time to take effect than usual. These functions therefore terminate and return, before the desired state is implemented. The return code then isn't a means to find out if the desired change was successfully performed. It rather states that the command itself was successfully interpreted and queued. To overcome this obstacle, the modules internal state is retrievable by the function **GetStatusError** and should be checked for being ready before and after sending a new command. In case an error occurred, the module switches into an inoperable state (error pending) until this error state was retrieved.

```
/* C/C++ */  int  SEPIA2_SWS_DecodeModuleType      (int              iSWSType,
                                                   char*            cSWSType );
```

arguments: iSWSType I : SWS type number (integer, 0…255)
cSWSType O : SWS type string, pointer to a buffer for at least 32 characters

description: Decodes the SWS type number as retrieved by SWS function **GetModuleType** to a string.

```
/* C/C++ */  int  SEPIA2_SWS_DecodeModuleState     (unsigned short  wState,
                                                   char*            cStatusText );
```

arguments: wState I : module state (unsigned short, 0…65535)
cStatusText O : module status string, pointer to a buffer for at least 148 characters

description: Decodes the module state to a string. The module state is a bit-coded word; Each bit may decode to a certain string. So, the length of the string needed is depending on the bits set in the status word. Currently, all strings added produce an output with a length of 147 characters (terminator excluded).
To be ready for future changes and enhancements, consider this: None of the parts is longer than 30 characters. (We will strictly adhere to this in future versions.) The parts are linked by the sequence “, “ (with a length of two characters); So the maximum length ever needed, calculates to 16 times 30 plus 15 times 2 plus terminator, hence 511 bytes.

```
/* C/C++ */  int  SEPIA2_SWS_GetModuleType        (int              iDevIdx,
                                                   int              iSlotId,
                                                   int*             piSWSType );
```

arguments: iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
iSlotId I : slot number of a SWS module
piSWSType O : SWS type number (pointer to integer; 0...255)

description: There are different SWS module types due to different possible technologies used to select the wavelength. This function returns the code of the currently implemented technology. The SWS type number can be decoded by the SWS function **DecodeModuleType**.

```
/* C/C++ */  int  SEPIA2_SWS_GetStatusError       (int              iDevIdx,
                                                   int              iSlotId,
                                                   unsigned short*  pwState,
                                                   short*           piErrorCode );
```

arguments: iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
iSlotId I : slot number of a SWS module
pwState O : state of the SWS module, (pointer to an unsigned short; 0...65535)
piErrorCode O : error code (pointer to a short integer)

description: The state is bit coded and can be decoded by the SWS function **DecodeModuleState**. If the error state bit (0x0010) is set, the error code <piErrorCode> is transmitted as well, else this variable is zero. As a side effect, error state bit and error code are cleared, if there are no further errors pending. Decode the error codes received with the LIB function **DecodeError.**

The SWS states are listed in the following table:

Symbol	Value	DescriptionSymbol
SEPIA2_SWS_STATE_READY	0x0000	Module ready
SEPIA2_SWS_STATE_INIT	0x0001	Module initialising
SEPIA2_SWS_STATE_BUSY	0x0002	Motors running or calculating on update data
SEPIA2_SWS_STATE_WAVELENGTH	0x0004	Wavelength received, waiting for bandwidth
SEPIA2_SWS_STATE_BANDWIDTH	0x0008	Bandwidth received, waiting for wavelength
SEPIA2_SWS_STATE_HARDWAREERROR	0x0010	Error code pending
SEPIA2_SWS_STATE_FWUPDATERUNNING	0x0020	Firmware update running
SEPIA2_SWS_STATE_FRAM_WRITEPROTECTED	0x0040	FRAM write protected: set, write enabled: cleared
SEPIA2_SWS_STATE_CALIBRATING	0x0080	Calibration mode: set, normal operation: cleared
SEPIA2_SWS_STATE_GUIRANGES	0x0100	GUI Ranges known: set, unknown: cleared

```
/* C/C++ */  int  SEPIA2_SWS_GetParamRanges        (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned long*  pulUpperWL,
                                                   unsigned long*  pulLowerWL,
                                                   unsigned long*  pulIncrWL,
                                                   unsigned long*  pulPMToggleWL,
                                                   unsigned long*  pulUpperBW,
                                                   unsigned long*  pulLowerBW,
                                                   unsigned long*  pulIncrBW,
                                                   int*            piUpperBPos,
                                                   int*            piLowerBPos,
                                                   int*            piIncrBPos );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
pulUpperWL	O	:	Upper limit of the wavelength (pointer to an unsigned long) given in pm
pulLowerWL	O	:	Lower limit of the wavelength (pointer to an unsigned long) given in pm
pulIncrWL	O	:	Stepwidth of the wavelength (pointer to an unsigned long) given in pm
pulPMToggleWL	O	:	Power mode toggle wavelength (pointer to an unsigned long) in pm
pulUpperBW	O	:	Upper limit of the bandwidth (pointer to an unsigned long) given in pm
pulLowerBW	O	:	Lower limit of the bandwidth (pointer to an unsigned long) given in pm
pulIncrBW	O	:	Stepwidth of the bandwidth (pointer to an unsigned long) given in pm
piUpperBPos	O	:	Upper beam shifter position (+90°) (pointer to an integer) in motor steps
piLowerBPos	O	:	Lower beam shifter position (-90°) (pointer to an integer) in motor steps
piIncrBPos	O	:	Stepwidth of the beam shifter pos. (pointer to an integer) in motor steps

description: Gets ranges for the parameter values of the wavelength selector: These are the wavelength, the bandwidth and the beam shifter positions. Although there are two independent shifters (one per axis, i.e. x/y), the same range for the both of them is used. Additionally the function returns the wavelength at which (in dynamic power mode) the power state of the pump module should switch from ECO mode to BOOST mode or vice versa in <pulPMToggleWL>.

```
/* C/C++ */  int  SEPIA2_SWS_GetParameters        (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned long*  pulWaveLength,
                                                   unsigned long*  pulBandWidth );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
pulWaveLength	O	:	current wavelength, (pointer to an unsigned long) in pm
pulBandWidth	O	:	current bandwidth (pointer to an unsigned long) in pm

description: Returns the adjusted values of wavelength and bandwidth.

```
/* C/C++ */  int  SEPIA2_SWS_SetParameters        (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned long   ulWaveLength,
                                                  unsigned long   ulBandWidth );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
ulWaveLength	I	:	wavelength (unsigned long) in pm
ulBandWidth	I	:	bandwidth (unsigned long) in pm

description: Sets the values for wavelength and bandwidth. As a side effect, the beam shifter positions are also set to (nearly) optimal values for this very wavelength / bandwidth combination as are defined by an internal calibration table. Refer also to the SWS functions **Get/SetCalTableSize**, **GetCalPointInfo** and **SetCalPointValues**.

```
/* C/C++ */  int  SEPIA2_SWS_GetIntensity        (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned long*  pulIntensityRaw,
                                                  float*          pfIntensity );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
pulIntensityRaw	O	:	intensity (pointer to unsigned long)
pfIntensity	O	:	intensity (pointer to float)

description: Gives a measure of the intensity of the beam after it has passed the SWS filters. The function returns two different values: <pulIntensityRaw> contains the read-out of the logarithmic amplifier boosting the photo diode current, while <pfIntensity> gives a linearly equalized calculation thereof. Although the readout of <pfIntensity> is neither equal nor even directly proportional to the absolute optical power of the beam coupled into the fiber, it is, however, a qualified measure for its relative intensity as long as you don't vary the wavelength or bandwidth once set. It then even allows for calibration on the optimal output coupling or for controlled attenuation relative to this optimum by use of the beam shifters.

```
/* C/C++ */  int  SEPIA2_SWS_GetFWVersion        (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned long*  pulFWVersion );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
pulFWVersion	O	:	firmware version (pointer to unsigned long)

description: Firmware Version is coded (byte[3] = major-nr., byte[2] = minor-nr., byte[1] + byte[0] as word = build-nr.)

```
/* C/C++ */  int  SEPIA2_SWS_UpdateFirmware      (int             iDevIdx,
                                                  int             iSlotId,
                                                  char*           pcFWFileName );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
pcFWFileName	I	:	name of the firmware file, pointer to a character buffer

description: Updates the firmware of the SWS module. The system must be restarted (power down) after updating.

```
/* C/C++ */  int  SEPIA2_SWS_SetFRAMWriteProtect (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned char   bWriteProtect );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
bWriteProtect	I	:	boolean: true/false stands for enable/disable write protection

description: To write a *.bin file with new DCL-settings to the FRAM, the FRAM write protection must be disabled. For secure reasons it is enabled by default. The write protection state of the FRAM is coded in the module state. The module state can be read out by the SWS function **GetStatusError**.

```
/* C/C++ */  int  SEPIA2_SWS_GetBeamPos          (int             iDevIdx,
                                                 int             iSlotId,
                                                 short*          piBeamVPos,
                                                 short*          piBeamHPos );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SWS module
- piBeamVPos O : position of the vertical beam shifter (pointer to a short int) in steps
- piBeamHPos O : position of the horizontal beam shifter (pointer to a short int) in steps

description: Returns the current positions of the beam shifters, correcting the vertical as well as the horizontal beam deviation for maximal intensity.

```
/* C/C++ */  int  SEPIA2_SWS_SetBeamPos          (int             iDevIdx,
                                                 int             iSlotId,
                                                 short           iBeamVPos,
                                                 short           iBeamHPos );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SWS module
- iBeamVPos I : position of vertical beam shifter in steps
- iBeamHPos I : position in steps of horizontal beam shifter in steps

description: Sets the new position of the beam shifters, changing the vertical as well as the horizontal beam deviation for maximal intensity.

```
/* C/C++ */ int SEPIA2_SWS_SetCalibrationMode ( int      iDevIdx,
                int             iSlotId,
                unsigned char bCalMode );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SWS module
- bCalMode I : boolean: true/false, enable/disable calibration mode

description: To calibrate the SWS module, calibration mode must be enabled. Calibration mode is coded in the module state. The module state can be read out by the SWS function **GetStatusError**.

```
/* C/C++ */  int  SEPIA2_SWS_GetCalTableSize     (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned short* pwWLIdxCount
                                                 unsigned short* pwBWIdxCount );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SWS module
- pwWLIdxCount O : wavelength index count (pointer to unsigned short)
- pwBWIdxCount O : bandwidth index count (pointer to unsigned short)

description: Reads out the current calibration table size.

```
/* C/C++ */  int  SEPIA2_SWS_SetCalTableSize     (int             iDevIdx,
                                                 int             iSlotId,
                                                 unsigned short  wWLIdxCount
                                                 unsigned short  wBWIdxCount
                                                 byte            bInit );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SWS module
- wWLIdxCount I : wavelength index count
- wBWIdxCount I : bandwidth index count
- bInit I : boolean: true/false, reset calibration table values / keep current values

description: Number of calibration points for the wavelength and bandwidth must be given. If the new calibration table size differs from the current, the calibration table values will always be cleared. With equal size, it depends on the value given with the <bInit> flag, whether the table will be re-initialized (true) or the calibration values will be preserved (false). For robust and near to optimal interpolation of the table values, the function grants for a sufficient size to set. If there aren't enough values to grant for stable interpolations on all of the independent VersaChrome® filter sets, the function returns an appropriate error code (-7119). The former table size and its content is then preserved.

```
/* C/C++ */  int  SEPIA2_SWS_GetCalPointInfo      (int             iDevIdx,
                                                  int             iSlotId,
                                                  short           iWLIdx
                                                  short           iBWIdx
                                                  unsigned long*  pulWaveLength
                                                  unsigned long*  pulBandWidth
                                                  short*          piBeamVPos
                                                  short*          piBeamHPos );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
iWLIdx	I	:	wavelength index
iBWIdx	I	:	bandwidth index
pulWaveLength	O	:	wavelength (pointer to an unsigned long) in pm
pulBandWidth	O	:	bandwidth (pointer to an unsigned long) in pm
piBeamVPos	O	:	position of the vertical beam shifter (pointer to a short integer) in steps
piBeamHPos	O	:	position of the horizontal beam shifter (pointer to a short integer) in steps

description: Gets the values of the wavelength, bandwidth and the positions of the vertical and horizontal beam shifter for a given calibration point, defined by the wavelength and bandwidth indices in the calibration table.

```
/* C/C++ */  int  SEPIA2_SWS_SetCalPointValues    (int             iDevIdx,
                                                  int             iSlotId,
                                                  short           iWLIdx
                                                  short           iBWIdx
                                                  short           iBeamVPos
                                                  short           iBeamHPos );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SWS module
iWLIdx	I	:	wavelength index
iBWIdx	I	:	bandwidth index
iBeamVPos	I	:	position of vertical beam shifter in steps
iBeamHPos	I	:	position of horizontal beam shifter in steps

description: Sets new values of the vertical and horizontal beam shifter positions for the given calibration point, defined by the wavelength and bandwidth indices in the calibration table.

2.5.4. Pump Control Module (SPM)

The SPM pump control module is a kind of “intelligent” module, supported by its own processor, too. Alike the SWS module, the user has to check for the internal state by means of the **GetStatusError** function. In case an error occurred, the module switches into an inoperable state (error pending) until this error state was retrieved.

```
/* C/C++ */  int  SEPIA2_SPM_DecodeModuleState     (unsigned short  wState,
                                                   char*           cStatusText );
```

arguments:
- wState I : module state number (unsigned short, 0…65535)
- cStatusText O : module status string, pointer to a buffer for at least 79 characters

description: Decodes the module state to a string. The module state is a bit-coded word; Each bit may decode to a certain string. So, the length of the string needed is depending on the bits set in the status word. Currently, all strings added produce an output with a length of 78 characters (terminator excluded).
To be ready for future changes and enhancements, consider this: None of the parts is longer than 30 characters. (We will strictly adhere to this in future versions.) The parts are linked by the sequence “, “ (with a length of two characters); So the maximum length ever needed, calculates to 16 times 30 plus 15 times 2 plus terminator, hence 511 bytes..

```
/* C/C++ */  int  SEPIA2_SPM_GetStatusError       (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned short* pwState,
                                                  short*          piErrorCode );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- pwState O : state of the SPM module, (pointer to an unsigned short integer)
- piErrorCode O : error code (pointer to a short integer)

description: The state is bit coded and can be decoded by the SPM function **DecodeModuleState**. If the error state bit (0x0010) is set, the error code <piErrorCode> is transmitted as well, else this variable is zero. As a side effect, error state bit and error code are cleared, if there are no further errors pending. Decode the error codes received with the LIB function **DecodeError**.

The SPM states are listed in the following table:

Symbol	Value	DescriptionSymbol
SEPIA2_SPM_STATE_READY	0x0000	Module ready
SEPIA2_SPM_STATE_INIT	0x0001	Module initialising
SEPIA2_SPM_STATE_HARDWAREERROR	0x0010	Error code pending
SEPIA2_SPM_STATE_FWUPDATERUNNING	0x0020	Firmware update running
SEPIA2_SPM_STATE_FRAM_WRITEPROTECTED	0x0040	FRAM write protected: set, write enabled: cleared

```
/* C/C++ */  int  SEPIA2_SPM_GetFWVersion         (int             iDevIdx,
                                                  int             iSlotId,
                                                  unsigned long*  pulFWVersion );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- pulFWVersion O : firmware version (pointer to unsigned long)

description: Firmware Version is coded (byte[3] = major-nr., byte[2] = minor-nr., byte[1] + byte[0] = word = build-nr.)

```
/* C/C++ */  int  SEPIA2_SPM_GetFiberAmplifierFail (int             iDevIdx,
                                                   int             iSlotId,
                                                   byte*           pbFbrAmpFail );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- pbFbrAmpFail O : fiber amplifier failure (pointer to byte; 0/1 → amp OK / amp failure)

description: The value of <pbFbrAmpFail> states, whether the fiber amplifier is working OK (0) or a failure was detected (1).

```
/* C/C++ */  int  SEPIA2_SPM_ResetFiberAmplifierFail              (int iDevIdx,
                                                   int             iSlotId,
                                                   byte            bFbrAmpFail );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- bFbrAmpFail I : fiber amplifier failure (byte; 0/1 → amp OK / amp failure)

description: With this function the value of <bFbrAmpFail> can be reset to 0 (amp OK) after repair.

```
/* C/C++ */  int  SEPIA2_SPM_GetPumpPowerState     (int             iDevIdx,
                                                   int             iSlotId,
                                                   byte*           pbPumpState
                                                   byte*           pbPumpMode );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- pbPumpState O : pump state, boolean (pointer to a byte; 0/1 → BOOST / ECO state)
- pbPumpMode O : pump mode, boolean (pointer to a byte; 0/1 → manual / dynamic)

description: Gets current pump mode and state. If the mode is set to “dynamic”, the state is controlled by the firmware, staying as long as appropriate in ECO mode and only changing to BOOST mode, where otherwise the output power would be too low. This mode is recommended to reduce the influence of fiber degradation.

```
/* C/C++ */  int  SEPIA2_SPM_SetPumpPowerState     (int             iDevIdx,
                                                   int             iSlotId,
                                                   byte            bPumpState
                                                   byte            bPumpMode );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- bPumpState O : pump state, boolean (byte; 0/1 → BOOST / ECO state)
- bPumpMode O : pump mode, boolean (byte); 0/1 → manual / dynamic)

description: Sets pump mode and state. If the mode is set to “dynamic”, the state is controlled by the firmware, staying as long as appropriate in ECO mode and only changing to BOOST mode, where otherwise the output power would be too low. This mode is recommended to reduce the influence of fiber degradation.

```
/* C/C++ */  int  SEPIA2_SPM_GetOperationTimers    (int             iDevIdx,
                                                   int             iSlotId,
                                                   unsigned long*  pMainPwrSwitch
                                                   unsigned long*  pUTOverAll
                                                   unsigned long*  pUTDelivery
                                                   unsigned long*  pUTFiberChg );
```

arguments:
- iDevIdx I : PQ Laser Device index (USB channel number, 0…7)
- iSlotId I : slot number of a SPM module
- pMainPwrSwitch O : main power switched on (pointer to unsigned long)
- pUTOverAll O : uptime over all (pointer to unsigned long) in seconds
- pUTDelivery O : uptime since delivery (pointer to unsigned long) in seconds
- pUTFiberChg O : uptime since fiber change (pointer to unsigned long) in seconds

description: Gets the operation timers.
- pMainPwrSwitch: counts how many times the SPM module was switched on
- pUTOverAll: shows the uptime in seconds
- pUTDelivery shows the uptime since delivery in seconds
- pUTFiberChg shows the uptime since fiber change in seconds

```
/* C/C++ */  int  SEPIA2_SPM_GetUpTimePowerTable  (int              iDevIdx,
                                                  int              iSlotId,
                                                  byte             bPumpState
                                                  T_pUpTimePwrTbl  pUpTimePwrTbl);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
bPumpState	I	:	boolean (true/false → activate/deactivate manual mode)
pUpTimePwrTbl	O	:	up time power table (pointer to an array of 10 unsigned integer)

description: Laser up time each second is ranked into one of 10 power classes. Each counter represent a power class. The first value in the array represents the lowest, the last value the highest power class. The sum of all is a measure for the operation time under load condition.

```
/* C/C++ */  int  SEPIA2_SPM_SetFRAMWriteProtect  (int              iDevIdx,
                                                  int              iSlotId,
                                                  unsigned char    bWriteProtect );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
bWriteProtect	I	:	boolean: true/false, enable/disable write protection

description: To write a *.bin file with new DCL-settings to the FRAM, the FRAM write protection must be disabled. For secure reasons it is be enabled by default. The write protection state of the FRAM is coded in the module state. The module state can be read out by the SPM function **GetStatusError**.

```
/* C/C++ */  int  SEPIA2_SPM_UpdateFirmware       (int              iDevIdx,
                                                  int              iSlotId,
                                                  char*            pcFWFileName );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
pcFWFileName	I	:	firmware file

description: Updates the firmware of the SPM module. The system must be restarted (power down) after updating.

```
/* C/C++ */  int  SEPIA2_SPM_GetControlMode       (int              iDevIdx,
                                                  int              iSlotId,
                                                  byte*            pbCtrlMode );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
pbCtrlMode	O	:	boolean (pointer to a byte), (0/1 → automatic / manual mode)

description: If control mode is manual (1), pump control values are controlled manually, else pump control values are controlled automatically (default; common users can not switch the mode, since this is a PQ-internal maintenance function, so this function is solely for documentation purposes). Don't mix-up with the parameter bPumpMode as used with the functions **GetPumpPowerState** and **SetPumpPowerState**.

```
/* C/C++ */  int  SEPIA2_SPM_GetManualPumpCurrent  (int              iDevIdx,
                                                  int              iSlotId,
                                                  word*            pwPumpCurrent1
                                                  word*            pwPumpCurrent2
                                                  word*            pwPumpCurrent3);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
pwPumpCurrent1	O	:	pump control value of first pump stage (pointer to an unsigned short)
pwPumpCurrent2	O	:	pump control value of second pump stage (pointer to an unsigned short)
pwPumpCurrent3	O	:	pump control value of third pump stage (pointer to an unsigned short)

description: These pump stage current values (in arbitrary units) are used if manual mode is activated. (Common users can not switch the mode, since this is a PQ-internal maintenance function, so this function is solely for documentation purposes.)

```
/* C/C++ */  int  SEPIA2_SPM_GetPumpCtrlParams     (int             iDevIdx,
                                                   int             iSlotId,
                                                   byte            bPumpState
                                                   T_pPumpCtrlPar  pPumpCtrlPar );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
bPumpState	I	:	boolean (byte; 0/1 → BOOST mode / ECO mode)
pPumpCtrlPar	O	:	pump control parameters (pointer to an array of 24 unsigned short integer)

description: Gets the set of parameters of the automatic pump control <pPumpCtrlPar> for the pump state given by <bPumpState>. The parameter set is organized as follows. It consists of four arrays, each of which has six entries (containing legal values between 0 to 1023). These arrays are:

- input value for the sampling point
- resulting current to set for pump 1 in arbitrary units
- resulting current to set for pump 2 in arbitrary units
- resulting current to set for pump 3 in arbitrary units

The firmware calculates the controlling curves for the pump currents by interpolating between these up to six sampling points.

```
/* C/C++ */  int  SEPIA2_SPM_GetPhotoDiodeCurrents (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piPhDCurrent1
                                                   int*            piPhDCurrent2
                                                   int*            piPhDCurrent3 );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
piPhDCurrent1	O	:	photo diode current (pointer to an integer) of the 1st pump stage in nA
piPhDCurrent2	O	:	photo diode current (pointer to an integer) of the 2nd pump stage in nA
piPhDCurrent3	O	:	photo diode current (pointer to an integer) of the 3rd pump stage in nA

description: Gets the filtered photo diode currents of all pump control stages.

```
/* C/C++ */  int  SEPIA2_SPM_GetPumpCurrents       (int             iDevIdx,
                                                   int             iSlotId,
                                                   int*            piPumpCurrent1
                                                   int*            piPumpCurrent2
                                                   int*            piPumpCurrent3);
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
piPumpCurrent1	O	:	current value (pointer to an integer) of the 1st pump stage current [a.u.]
piPumpCurrent2	O	:	current value (pointer to an integer) of the 2nd pump stage current [a.u.]
piPumpCurrent3	O	:	current value (pointer to an integer) of the 3rd pump stage current [a.u.]

description: Gets the pump stage currents, the pump control currently set, in arbitrary units.

```
/* C/C++ */  int  SEPIA2_SPM_GetSensorData          (int             iDevIdx,
                                                    int             iSlotId,
                                                    T_pSensorData   pSensorData );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
pSensorData	O	:	temperatures of pump stages and over all current (pointer to an array of 9 unsigned short integer)

description: Gets the current sensor data of all pump control stages. <pSensorData> points to an array, that could also be read as a struct, containing the following data:

- temperature of pump 1
- temperature of pump 2
- temperature of pump 3
- temperature of pump 4
- temperature of the fiber stacker
- temperature of an auxiliary control point (reserved)
- resulting over all current
- auxiliary sensor 1 (reserved)
- auxiliary sensor 2 (reserved)

The temperatures are given as ADC values and have to be calculated as follows

$$\frac{\vartheta}{^{\circ}C} = \frac{1}{\frac{1}{3988} * \ln\left(\frac{4700}{10000 * \left(\frac{1.5 * 1024}{2.5 * value} - 1\right)}\right) + \frac{1}{298.15}} - 273.15$$

while the sensed currents are given as

$$\frac{I}{A} = \frac{25}{1024} \cdot value$$

```
/* C/C++ */  int  SEPIA2_SPM_GetTemperatureAdjust  (int             iDevIdx,
                                                    int             iSlotId,
                                                    T_pTemperature  pTempAdjust );
```

arguments:

iDevIdx	I	:	PQ Laser Device index (USB channel number, 0…7)
iSlotId	I	:	slot number of a SPM module
pTempAdjust	O	:	params for temperature controlling (pointer to an array of 6 unsigned short integer)

description: Gets the temperature control parameters (legal values from 0 to 1023). Structure of the data is the same as for **GetSensorData**.

3. PQ Laser Device – Demo Programs

Please note that all demo code provided is correct to our best knowledge, however, we must disclaim all warranties as to fitness for a particular purpose of this code. It is provided 'as is' for no more than explanatory purposes.

The demos are kept as simple as possible to maintain focus on the key issues of accessing the PQ Laser Device. It is neither their task, to show all degrees of freedom of your PQ Laser Device nor to illustrate the functionality of all modules possibly installed. This is why most of the demos have a minimalistic user interface and/or run from a simple DOS box (console). For the same reason, the parameter settings are mostly hard-coded and thereby fixed at compile time. It may therefore be necessary to change the source code and re-compile the demos in order to run them in a way that is matched to your individual system setup. Running them unmodified may result in useless settings because of inappropriate trigger levels etc. and although it should be taken as most extreme unlikely it might even cause damage to your laser equipment.

There are demos for MS Visual Studio C/C++ and Delphi provided in their language specific subfolder; Other languages may join the collection by and by without explicit mentioning it in this manual. We tried to implement the respective demos as close to identical behaviour in the different languages as could be, yet we don't guarantee for this. For each of these programming languages/systems there are different demo programs for at least two dedicated tasks: system – wide inquiry of the actual settings (named "ReadAllData…") and exemplarily change some settings (named "SetSomeData…"). Note that the latter needs to be executed in an directory in which write access is granted. It tries to write a data recovery file before altering the PQ Laser Device's working parameters. Invoked twice it restores the original data from this file removing it afterwards. Furthermore it expects to find at least a SOM 828 in slot 100 and a SLM 828 in slot 200.

All demos have in common, that they presume to find a PQ Laser Device at USB – channel 0. If you are running other PicoQuant products, that use the same hardware driver, the PQ Laser Device might get other channel numbers during USB enumeration. There are two different strategies to overcome this situation: You might

a) alter the device– resp. channel–index variable iDevIdx (set to 0 by default) to the actual value and recompile the demo or
b) force PQ Laser Device to be the first device enumerated by your computer. This is done by drawing off **all** devices from the USB port for a few ten seconds and putting them all back online, **but the PQ Laser Device first**.

The demo programs commonly illustrate the typical structure of PQ Laser Device sessions:

- Get library version and check it comparing to system constant LIB_VERSION_REFERENCE (optional)
- Open PQ Laser Device on the desired USB channels (mandatory)
- Get firmware version and USB string descriptors (just for information and service purposes) (optional)
- Get current module map from firmware (mandatory)
- Get last error detected by firmware and decode it if necessary (optional)
- Insert implementation of your desired behaviour here
  …
  …
- Free module map (recommended)
- Close PQ Laser Device (mandatory)

4. Appendix: Tables Concerning the PQ Laser Device – API

4.1. Table of Data Types

The Sepia2_Lib.dll is written in C and its data types correspond to standard C/C++ data types on 32 bit platforms as follows:

used data types (C/C++)	bits	remarks
`char`	8	character
`char*`	?	pointer to char; pointer to string (0–terminated)
`unsigned char`	8	byte
`short int`	16	signed integer
`unsigned short`	16	unsigned integer (word)
`int`	32	signed integer
`long`	32	signed integer
`float`	32	floating point number (7 to 8 significant digits)
`double`	64	floating point number (15 to 16 significant digits)
`__int64`	64	signed integer

These types are supported by most of the major programming languages…

4.2. Table of Error Codes

Symbol	Nr.	Error Text
SEPIA2_ERR_NO_ERROR	0	no error
SEPIA2_ERR_FW_MEMORY_ALLOCATION_ERROR	-1001	FW: memory allocation error
SEPIA2_ERR_FW_CRC_ERROR_WHILE_CHECKING_SCM_828_MODULE	-1002	FW: CRC error while checking SCM 828 module
SEPIA2_ERR_FW_CRC_ERROR_WHILE_CHECKING_BACKPLANE	-1003	FW: CRC error while checking backplane
SEPIA2_ERR_FW_CRC_ERROR_WHILE_CHECKING_MODULE	-1004	FW: CRC error while checking module
SEPIA2_ERR_FW_MAPSIZE_ERROR	-1005	FW: map size error
SEPIA2_ERR_FW_UNKNOWN_ERROR_PHASE	-1006	FW: unknown FW error phase
SEPIA2_ERR_FW_ILLEGAL_MODULE_CHANGE	-1111	FW: illegal module change
SEPIA2_ERR_USB_WRONG_DRIVER_VERSION	-2001	USB: wrong driver version
SEPIA2_ERR_USB_OPEN_DEVICE_ERROR	-2002	USB: open device error
SEPIA2_ERR_USB_DEVICE_BUSY	-2003	USB: device busy
SEPIA2_ERR_USB_CLOSE_DEVICE_ERROR	-2005	USB: close device error
SEPIA2_ERR_USB_DEVICE_CHANGED	-2006	USB: device changed
SEPIA2_ERR_I2C_ADDRESS_ERROR	-2010	I2C: address error
SEPIA2_ERR_DEVICE_INDEX_ERROR	-2011	USB: device index error
SEPIA2_ERR_ILLEGAL_MULTIPLEXER_PATH	-2012	I2C: illegal multiplexer path
SEPIA2_ERR_ILLEGAL_MULTIPLEXER_LEVEL	-2013	I2C: illegal multiplexer level
SEPIA2_ERR_ILLEGAL_SLOT_ID	-2014	I2C: illegal slot id
SEPIA2_ERR_NO_UPTIMECOUNTER	-2015	FRAM: no uptime counter
SEPIA2_ERR_FRAM_BLOCKWRITE_ERROR	-2020	FRAM: blockwrite error
SEPIA2_ERR_FRAM_BLOCKREAD_ERROR	-2021	FRAM: blockread error
SEPIA2_ERR_FRAM_CRC_BLOCKCHECK_ERROR	-2022	FRAM: CRC blockcheck error
SEPIA2_ERR_RAM_BLOCK_ALLOCATION_ERROR	-2023	RAM: block allocation error
SEPIA2_ERR_I2C_INITIALISING_COMMAND_EXECUTION_ERROR	-2100	I2C: initialising command execution error
SEPIA2_ERR_I2C_FETCHING_INITIALISING_COMMANDS_ERROR	-2101	I2C: fetching initialising commands error
SEPIA2_ERR_I2C_WRITING_INITIALISING_COMMANDS_ERROR	-2102	I2C: writing initialising commands error
SEPIA2_ERR_I2C_MODULE_CALIBRATING_ERROR	-2200	I2C: module calibrating error
SEPIA2_ERR_I2C_FETCHING_CALIBRATING_COMMANDS_ERROR	-2201	I2C: fetching calibrating commands error
SEPIA2_ERR_I2C_WRITING_CALIBRATING_COMMANDS_ERROR	-2202	I2C: writing calibrating commands error
SEPIA2_ERR_DCL_FILE_OPEN_ERROR	-2301	DCL: file open error
SEPIA2_ERR_DCL_WRONG_FILE_LENGTH	-2302	DCL: wrong file length
SEPIA2_ERR_DCL_FILE_READ_ERROR	-2303	DCL: file read error
SEPIA2_ERR_FRAM_IS_WRITE_PROTECTED	-2304	FRAM: is write protected
SEPIA2_ERR_DCL_FILE_SPECIFIES_DIFFERENT_MODULETYPE	-2305	DCL: file specifies different moduletype
SEPIA2_ERR_DCL_FILE_SPECIFIES_DIFFERENT_SERIAL_NUMBER	-2306	DCL: file specifies different serial number
SEPIA2_ERR_I2C_INVALID_ARGUMENT	-3001	I2C: invalid argument
SEPIA2_ERR_I2C_NO_ACKNOWLEDGE_ON_WRITE_ADRESSBYTE	-3002	I2C: no acknowledge on write adressbyte
SEPIA2_ERR_I2C_NO_ACKNOWLEDGE_ON_READ_ADRESSBYTE	-3003	I2C: no acknowledge on read adressbyte
SEPIA2_ERR_I2C_NO_ACKNOWLEDGE_ON_WRITE_DATABYTE	-3004	I2C: no acknowledge on write databyte
SEPIA2_ERR_I2C_READ_BACK_ERROR	-3005	I2C: read back error
SEPIA2_ERR_I2C_READ_ERROR	-3006	I2C: read error
SEPIA2_ERR_I2C_WRITE_ERROR	-3007	I2C: write error

Symbol	Nr.	Error Text
SEPIA2_ERR_I_O_FILE_ERROR	-3009	I/O: file error
SEPIA2_ERR_I2C_MULTIPLEXER_ERROR	-3014	I2C: multiplexer error
SEPIA2_ERR_I2C_MULTIPLEXER_PATH_ERROR	-3015	I2C: multiplexer path error
SEPIA2_ERR_USB_INVALID_ARGUMENT	-3201	USB: invalid argument
SEPIA2_ERR_USB_DEVICE_STILL_OPEN	-3202	USB: device still open
SEPIA2_ERR_USB_NO_MEMORY	-3203	USB: no memory
SEPIA2_ERR_USB_OPEN_FAILED	-3204	USB: open failed
SEPIA2_ERR_USB_GET_DESCRIPTOR_FAILED	-3205	USB: get descriptor failed
SEPIA2_ERR_USB_INAPPROPRIATE_DEVICE	-3206	USB: inappropriate device
SEPIA2_ERR_USB_BUSY_DEVICE	-3207	USB: busy device
SEPIA2_ERR_USB_INVALID_HANDLE	-3208	USB: invalid handle
SEPIA2_ERR_USB_INVALID_DESCRIPTOR_BUFFER	-3209	USB: invalid descriptor buffer
SEPIA2_ERR_USB_IOCTRL_FAILED	-3210	USB: IOCTRL failed
SEPIA2_ERR_USB_VCMD_FAILED	-3211	USB: vcmd failed
SEPIA2_ERR_USB_NO_SUCH_PIPE	-3212	USB: no such pipe
SEPIA2_ERR_USB_REGISTER_NOTIFICATION_FAILED	-3213	USB: register notification failed
SEPIA2_ERR_I2C_DEVICE_ERROR	-3256	I2C: device error
SEPIA2_ERR_LMP_ADC_TABLES_NOT_FOUND	-3501	LMP: ADC tables not found
SEPIA2_ERR_LMP_ADC_OVERFLOW	-3502	LMP: ADC overflow
SEPIA2_ERR_LMP_ADC_UNDERFLOW	-3503	LMP: ADC underflow
SEPIA2_ERR_SCM_VOLTAGE_LIMITS_TABLE_NOT_FOUND	-4001	SCM: voltage limits table not found
SEPIA2_ERR_SCM_VOLTAGE_SCALING_LIST_NOT_FOUND	-4002	SCM: voltage scaling list not found
SEPIA2_ERR_SCM_REPEATEDLY_MEASURED_VOLTAGE_FAILURE	-4003	SCM: repeatedly measured voltage failure
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_0_VOLTAGE_TOO_LOW	-4010	SCM: power supply line 0: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_1_VOLTAGE_TOO_LOW	-4011	SCM: power supply line 1: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_2_VOLTAGE_TOO_LOW	-4012	SCM: power supply line 2: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_3_VOLTAGE_TOO_LOW	-4013	SCM: power supply line 3: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_4_VOLTAGE_TOO_LOW	-4014	SCM: power supply line 4: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_5_VOLTAGE_TOO_LOW	-4015	SCM: power supply line 5: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_6_VOLTAGE_TOO_LOW	-4016	SCM: power supply line 6: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_7_VOLTAGE_TOO_LOW	-4017	SCM: power supply line 7: voltage too low
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_0_VOLTAGE_TOO_HIGH	-4020	SCM: power supply line 0: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_1_VOLTAGE_TOO_HIGH	-4021	SCM: power supply line 1: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_2_VOLTAGE_TOO_HIGH	-4022	SCM: power supply line 2: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_3_VOLTAGE_TOO_HIGH	-4023	SCM: power supply line 3: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_4_VOLTAGE_TOO_HIGH	-4024	SCM: power supply line 4: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_5_VOLTAGE_TOO_HIGH	-4025	SCM: power supply line 5: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_6_VOLTAGE_TOO_HIGH	-4026	SCM: power supply line 6: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LINE_7_VOLTAGE_TOO_HIGH	-4027	SCM: power supply line 7: voltage too high
SEPIA2_ERR_SCM_POWER_SUPPLY_LASER_TURNING_OFF_VOLTAGE_TOO_HIGH	-4030	SCM: power supply laser turn-off-voltage too high
SEPIA2_ERR_SOM_INT_OSCILLATOR_S_FREQ_LIST_NOT_FOUND	-5001	SOM: int. oscillator's freq.-list not found
SEPIA2_ERR_SOM_TRIGGER_MODE_LIST_NOT_FOUND	-5002	SOM: trigger mode list not found
SEPIA2_ERR_SOM_TRIGGER_LEVEL_NOT_FOUND	-5003	SOM: trigger level not found
SEPIA2_ERR_SOM_PREDIVIDER_PRETRIGGER_OR_TRIGGERMASK_NOT_FOUND	-5004	SOM: predivider, pretrigger, triggermask not found
SEPIA2_ERR_SOM_BURSTLENGTH_NOT_FOUND	-5005	SOM: burstlength not found

Symbol	Nr.	Error Text
SEPIA2_ERR_SOM_OUTPUT_AND_SYNC_ENABLE_NOT_FOUND	-5006	SOM: output and sync enable not found
SEPIA2_ERR_SOM_TRIGGER_LEVEL_OUT_OF_BOUNDS	-5007	SOM: trigger level out of bounds
SEPIA2_ERR_SOM_ILLEGAL_FREQUENCY_TRIGGERMODE	-5008	SOM: illegal frequency / triggermode
SEPIA2_ERR_SOM_ILLEGAL_FREQUENCY_DIVIDER	-5009	SOM: illegal frequency divider (equal 0)
SEPIA2_ERR_SOM_ILLEGAL_PRESYNC	-5010	SOM: illegal presync (greater than divider)
SEPIA2_ERR_SOM_ILLEGAL_BURST_LENGTH	-5011	SOM: illegal burst length (>/= 2^24 or < 0)
SEPIA2_ERR_SOM_AUX_IO_CTRL_NOT_FOUND	-5012	SOM: AUX I/O control data not found
SEPIA2_ERR_SOM_ILLEGAL_AUX_OUT_CTRL	-5013	SOM: illegal AUX output control data
SEPIA2_ERR_SOM_ILLEGAL_AUX_IN_CTRL	-5014	SOM: illegal AUX input control data
SEPIA2_ERR_SLM_ILLEGAL_FREQUENCY_TRIGGERMODE	-6001	SLM: illegal frequency / triggermode
SEPIA2_ERR_SLM_ILLEGAL_INTENSITY	-6002	SLM: illegal intensity (> 100% or < 0%)
SEPIA2_ERR_SLM_ILLEGAL_HEAD_TYPE	-6003	SLM: illegal head type
SEPIA2_ERR_SML_ILLEGAL_INTENSITY	-6501	SML: illegal intensity (> 100% or < 0%)
SEPIA2_ERR_SML_POWER_SCALE_TABLES_NOT_FOUND	-6502	SML: power scale tables not found
SEPIA2_ERR_SML_ILLEGAL_HEAD_TYPE	-6503	SML: illegal head type
SEPIA2_ERR_SWM_CALIBRATION_TABLES_NOT_FOUND	-6701	SWM: calibration tables not found
SEPIA2_ERR_SWM_ILLEGAL_CURVE_INDEX	-6702	SWM: illegal curve index
SEPIA2_ERR_SWM_ILLEGAL_TIMBASE_RANGE_INDEX	-6703	SWM: illegal timebase range index
SEPIA2_ERR_SWM_ILLEGAL_PULSE_AMPLITUDE	-6704	SWM: illegal pulse amplitude
SEPIA2_ERR_SWM_ILLEGAL_RAMP_SLEW_RATE	-6705	SWM: illegal ramp slew rate
SEPIA2_ERR_SWM_ILLEGAL_PULSE_START_DELAY	-6706	SWM: illegal pulse start delay
SEPIA2_ERR_SWM_ILLEGAL_RAMP_START_DELAY	-6707	SWM: illegal ramp start delay
SEPIA2_ERR_SWM_ILLEGAL_WAVE_STOP_DELAY	-6708	SWM: illegal wave stop delay
SEPIA2_ERR_SWM_ILLEGAL_TABLENAME	-6709	SWM: illegal tablename
SEPIA2_ERR_SWM_ILLEGAL_TABLE_INDEX	-6710	SWM: illegal table index
SEPIA2_ERR_SWM_ILLEGAL_TABLE_FIELD	-6711	SWM: illegal table field
SEPIA2_ERR_SPM_ILLEGAL_INPUT_VALUE	-7001	Solea SPM: illegal input value
SEPIA2_ERR_SPM_VALUE_OUT_OF_BOUNDS	-7006	Solea SPM: value out of bounds
SEPIA2_ERR_SPM_FW_OUT_OF_MEMORY	-7011	Solea SPM FW: out of memory
SEPIA2_ERR_SPM_FW_UPDATE_FAILED	-7013	Solea SPM FW: update failed
SEPIA2_ERR_SPM_FW_CRC_CHECK_FAILED	-7014	Solea SPM FW: CRC check failed
SEPIA2_ERR_SPM_FW_ERROR_ON_FLASH_DELETION	-7015	Solea SPM FW: error on flash deletion
SEPIA2_ERR_SPM_FW_FILE_OPEN_ERROR	-7021	Solea SPM FW: file open error
SEPIA2_ERR_SPM_FW_FILE_READ_ERROR	-7022	Solea SPM FW: file read error
SEPIA2_ERR_SSM_SCALING_TABLES_NOT_FOUND	-7051	Solea SSM: scaling tables not found
SEPIA2_ERR_SSM_ILLEGAL_TRIGGER_MODE	-7052	Solea SSM: illegal trigger mode
SEPIA2_ERR_SSM_ILLEGAL_TRIGGER_LEVEL_VALUE	-7053	Solea SSM: illegal trigger level value
SEPIA2_ERR_SSM_ILLEGAL_CORRECTION_VALUE	-7054	Solea SSM: illegal correction value
SEPIA2_ERR_SSM_TRIGGER_DATA_NOT_FOUND	-7055	Solea SSM: trigger data not found
SEPIA2_ERR_SSM_CORRECTION_DATA_COMMAND_NOT_FOUND	-7056	Solea SSM: correction data command not found
SEPIA2_ERR_SWS_SCALING_TABLES_NOT_FOUND	-7101	Solea SWS: scaling tables not found
SEPIA2_ERR_SWS_ILLEGAL_HW_MODULETYPE	-7102	Solea SWS: illegal HW moduletype
SEPIA2_ERR_SWS_MODULE_NOT_FUNCTIONAL	-7103	Solea SWS: module not functional
SEPIA2_ERR_SWS_ILLEGAL_CENTER_WAVELENGTH	-7104	Solea SWS: illegal center wavelength
SEPIA2_ERR_SWS_ILLEGAL_BANDWIDTH	-7105	Solea SWS: illegal bandwidth

Symbol	Nr.	Error Text
SEPIA2_ERR_SWS_VALUE_OUT_OF_BOUNDS	-7106	Solea SWS: value out of bounds
SEPIA2_ERR_SWS_MODULE_BUSY	-7107	Solea SWS: module busy
SEPIA2_ERR_SWS_FW_WRONG_COMPONENT_ANSWERING	-7109	Solea SWS FW: wrong component answering
SEPIA2_ERR_SWS_FW_UNKNOWN_HW_MODULETYPE	-7110	Solea SWS FW: unknown HW moduletype
SEPIA2_ERR_SWS_FW_OUT_OF_MEMORY	-7111	Solea SWS FW: out of memory
SEPIA2_ERR_SWS_FW_VERSION_CONFLICT	-7112	Solea SWS FW: version conflict
SEPIA2_ERR_SWS_FW_UPDATE_FAILED	-7113	Solea SWS FW: update failed
SEPIA2_ERR_SWS_FW_CRC_CHECK_FAILED	-7114	Solea SWS FW: CRC check failed
SEPIA2_ERR_SWS_FW_ERROR_ON_FLASH_DELETION	-7115	Solea SWS FW: error on flash deletion
SEPIA2_ERR_SWS_FW_CALIBRATION_MODE_ERROR	-7116	Solea SWS FW: calibration mode error
SEPIA2_ERR_SWS_FW_FUNCTION_NOT_IMPLEMENTED_YET	-7117	Solea SWS FW: function not implemented yet
SEPIA2_ERR_SWS_FW_WRONG_CALIBRATION_TABLE_ENTRY	-7118	Solea SWS FW: wrong calibration table entry
SEPIA2_ERR_SWS_FW_INSUFFICIENT_CALIBRATION_TABLE_SIZE	-7119	Solea SWS FW: insufficient calibration table size
SEPIA2_ERR_SWS_FW_FILE_OPEN_ERROR	-7151	Solea SWS FW: file open error
SEPIA2_ERR_SWS_FW_FILE_READ_ERROR	-7152	Solea SWS FW: file read error
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_INIT_TIMEOUT	-7201	Solea SWS HW: module 0, all motors: init timeout
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_PLAUSI_CHECK	-7202	Solea SWS HW: module 0, all motors: plausi check
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_DAC_SET_CURRENT	-7203	Solea SWS HW: module 0, all motors: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_TIMEOUT	-7204	Solea SWS HW: module 0, all motors: timeout
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_FLASH_WRITE_ERROR	-7205	Solea SWS HW: module 0, all motors: flash write error
SEPIA2_ERR_SWS_HW_MODULE_0_ALL_MOTORS_OUT_OF_BOUNDS	-7206	Solea SWS HW: module 0, all motors: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_0_I2C_FAILURE	-7207	Solea SWS HW: module 0: I2C failure
SEPIA2_ERR_SWS_HW_MODULE_0_INIT_FAILURE	-7208	Solea SWS HW: module 0: init failure
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_DATA_NOT_FOUND	-7210	Solea SWS HW: module 0, motor 1: data not found
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_INIT_TIMEOUT	-7211	Solea SWS HW: module 0, motor 1: init timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_PLAUSI_CHECK	-7212	Solea SWS HW: module 0, motor 1: plausi check
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_DAC_SET_CURRENT	-7213	Solea SWS HW: module 0, motor 1: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_TIMEOUT	-7214	Solea SWS HW: module 0, motor 1: timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_FLASH_WRITE_ERROR	-7215	Solea SWS HW: module 0, motor 1: flash write error
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_1_OUT_OF_BOUNDS	-7216	Solea SWS HW: module 0, motor 1: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_DATA_NOT_FOUND	-7220	Solea SWS HW: module 0, motor 2: data not found
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_INIT_TIMEOUT	-7221	Solea SWS HW: module 0, motor 2: init timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_PLAUSI_CHECK	-7222	Solea SWS HW: module 0, motor 2: plausi check
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_DAC_SET_CURRENT	-7223	Solea SWS HW: module 0, motor 2: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_TIMEOUT	-7224	Solea SWS HW: module 0, motor 2: timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_FLASH_WRITE_ERROR	-7225	Solea SWS HW: module 0, motor 2: flash write error
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_2_OUT_OF_BOUNDS	-7226	Solea SWS HW: module 0, motor 2: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_DATA_NOT_FOUND	-7230	Solea SWS HW: module 0, motor 3: data not found
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_INIT_TIMEOUT	-7231	Solea SWS HW: module 0, motor 3: init timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_PLAUSI_CHECK	-7232	Solea SWS HW: module 0, motor 3: plausi check
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_DAC_SET_CURRENT	-7233	Solea SWS HW: module 0, motor 3: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_TIMEOUT	-7234	Solea SWS HW: module 0, motor 3: timeout
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_FLASH_WRITE_ERROR	-7235	Solea SWS HW: module 0, motor 3: flash write error
SEPIA2_ERR_SWS_HW_MODULE_0_MOTOR_3_OUT_OF_BOUNDS	-7236	Solea SWS HW: module 0, motor 3: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_INIT_TIMEOUT	-7301	Solea SWS HW: module 1, all motors: init timeout

Symbol	Nr.	Error Text
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_PLAUSI_CHECK	-7302	Solea SWS HW: module 1, all motors: plausi check
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_DAC_SET_CURRENT	-7303	Solea SWS HW: module 1, all motors: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_TIMEOUT	-7304	Solea SWS HW: module 1, all motors: timeout
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_FLASH_WRITE_ERROR	-7305	Solea SWS HW: module 1, all motors: flash write error
SEPIA2_ERR_SWS_HW_MODULE_1_ALL_MOTORS_OUT_OF_BOUNDS	-7306	Solea SWS HW: module 1, all motors: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_1_I2C_FAILURE	-7307	Solea SWS HW: module 1: I2C failure
SEPIA2_ERR_SWS_HW_MODULE_1_INIT_FAILURE	-7308	Solea SWS HW: module 1: init failure
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_DATA_NOT_FOUND	-7310	Solea SWS HW: module 1, motor 1: data not found
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_INIT_TIMEOUT	-7311	Solea SWS HW: module 1, motor 1: init timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_PLAUSI_CHECK	-7312	Solea SWS HW: module 1, motor 1: plausi check
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_DAC_SET_CURRENT	-7313	Solea SWS HW: module 1, motor 1: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_TIMEOUT	-7314	Solea SWS HW: module 1, motor 1: timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_FLASH_WRITE_ERROR	-7315	Solea SWS HW: module 1, motor 1: flash write error
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_1_OUT_OF_BOUNDS	-7316	Solea SWS HW: module 1, motor 1: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_DATA_NOT_FOUND	-7320	Solea SWS HW: module 1, motor 2: data not found
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_INIT_TIMEOUT	-7321	Solea SWS HW: module 1, motor 2: init timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_PLAUSI_CHECK	-7322	Solea SWS HW: module 1, motor 2: plausi check
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_DAC_SET_CURRENT	-7323	Solea SWS HW: module 1, motor 2: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_TIMEOUT	-7324	Solea SWS HW: module 1, motor 2: timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_FLASH_WRITE_ERROR	-7325	Solea SWS HW: module 1, motor 2: flash write error
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_2_OUT_OF_BOUNDS	-7326	Solea SWS HW: module 1, motor 2: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_DATA_NOT_FOUND	-7330	Solea SWS HW: module 1, motor 3: data not found
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_INIT_TIMEOUT	-7331	Solea SWS HW: module 1, motor 3: init timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_PLAUSI_CHECK	-7332	Solea SWS HW: module 1, motor 3: plausi check
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_DAC_SET_CURRENT	-7333	Solea SWS HW: module 1, motor 3: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_TIMEOUT	-7334	Solea SWS HW: module 1, motor 3: timeout
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_FLASH_WRITE_ERROR	-7335	Solea SWS HW: module 1, motor 3: flash write error
SEPIA2_ERR_SWS_HW_MODULE_1_MOTOR_3_OUT_OF_BOUNDS	-7336	Solea SWS HW: module 1, motor 3: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_INIT_TIMEOUT	-7401	Solea SWS HW: module 2, all motors: init timeout
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_PLAUSI_CHECK	-7402	Solea SWS HW: module 2, all motors: plausi check
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_DAC_SET_CURRENT	-7403	Solea SWS HW: module 2, all motors: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_TIMEOUT	-7404	Solea SWS HW: module 2, all motors: timeout
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_FLASH_WRITE_ERROR	-7405	Solea SWS HW: module 2, all motors: flash write error
SEPIA2_ERR_SWS_HW_MODULE_2_ALL_MOTORS_OUT_OF_BOUNDS	-7406	Solea SWS HW: module 2, all motors: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_2_I2C_FAILURE	-7407	Solea SWS HW: module 2: I2C failure
SEPIA2_ERR_SWS_HW_MODULE_2_INIT_FAILURE	-7408	Solea SWS HW: module 2: init failure
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_DATA_NOT_FOUND	-7410	Solea SWS HW: module 2, motor 1: data not found
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_INIT_TIMEOUT	-7411	Solea SWS HW: module 2, motor 1: init timeout
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_PLAUSI_CHECK	-7412	Solea SWS HW: module 2, motor 1: plausi check
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_DAC_SET_CURRENT	-7413	Solea SWS HW: module 2, motor 1: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_TIMEOUT	-7414	Solea SWS HW: module 2, motor 1: timeout
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_FLASH_WRITE_ERROR	-7415	Solea SWS HW: module 2, motor 1: flash write error
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_1_OUT_OF_BOUNDS	-7416	Solea SWS HW: module 2, motor 1: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_DATA_NOT_FOUND	-7420	Solea SWS HW: module 2, motor 2: data not found
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_INIT_TIMEOUT	-7421	Solea SWS HW: module 2, motor 2: init timeout

Symbol	Nr.	Error Text
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_PLAUSI_CHECK	-7422	Solea SWS HW: module 2, motor 2: plausi check
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_DAC_SET_CURRENT	-7423	Solea SWS HW: module 2, motor 2: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_TIMEOUT	-7424	Solea SWS HW: module 2, motor 2: timeout
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_FLASH_WRITE_ERROR	-7425	Solea SWS HW: module 2, motor 2: flash write error
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_2_OUT_OF_BOUNDS	-7426	Solea SWS HW: module 2, motor 2: out of bounds
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_DATA_NOT_FOUND	-7430	Solea SWS HW: module 2, motor 3: data not found
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_INIT_TIMEOUT	-7431	Solea SWS HW: module 2, motor 3: init timeout
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_PLAUSI_CHECK	-7432	Solea SWS HW: module 2, motor 3: plausi check
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_DAC_SET_CURRENT	-7433	Solea SWS HW: module 2, motor 3: DAC set current
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_TIMEOUT	-7434	Solea SWS HW: module 2, motor 3: timeout
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_FLASH_WRITE_ERROR	-7435	Solea SWS HW: module 2, motor 3: flash write error
SEPIA2_ERR_SWS_HW_MODULE_2_MOTOR_3_OUT_OF_BOUNDS	-7436	Solea SWS HW: module 2, motor 3: out of bounds
SEPIA2_ERR_LIB_TOO_MANY_USB_HANDLES	-9001	LIB: too many USB handles
SEPIA2_ERR_LIB_ILLEGAL_DEVICE_INDEX	-9002	LIB: illegal device index
SEPIA2_ERR_LIB_USB_DEVICE_OPEN_ERROR	-9003	LIB: USB device open error
SEPIA2_ERR_LIB_USB_DEVICE_BUSY_OR_BLOCKED	-9004	LIB: USB device busy or blocked
SEPIA2_ERR_LIB_USB_DEVICE_ALREADY_OPENED	-9005	LIB: USB device already opened
SEPIA2_ERR_LIB_UNKNOWN_USB_HANDLE	-9006	LIB: unknown USB handle
SEPIA2_ERR_LIB_SCM_828_MODULE_NOT_FOUND	-9007	LIB: SCM 828 module not found
SEPIA2_ERR_LIB_ILLEGAL_SLOT_NUMBER	-9008	LIB: illegal slot number
SEPIA2_ERR_LIB_REFERENCED_SLOT_IS_NOT_IN_USE	-9009	LIB: referenced slot is not in use
SEPIA2_ERR_LIB_THIS_IS_NO_SCM_828_MODULE	-9010	LIB: this is no SCM 828 module
SEPIA2_ERR_LIB_THIS_IS_NO_SOM_828_MODULE	-9011	LIB: this is no SOM 828 module
SEPIA2_ERR_LIB_THIS_IS_NO_SLM_828_MODULE	-9012	LIB: this is no SLM 828 module
SEPIA2_ERR_LIB_THIS_IS_NO_SML_828_MODULE	-9013	LIB: this is no SML 828 module
SEPIA2_ERR_LIB_THIS_IS_NO_SWM_828_MODULE	-9014	LIB: this is no SWM 828 module
SEPIA2_ERR_LIB_THIS_IS_NO_SOLEA_SSM_MODULE	-9015	LIB: this is no Solea SSM module
SEPIA2_ERR_LIB_THIS_IS_NO_SOLEA_SWS_MODULE	-9016	LIB: this is no Solea SWS module
SEPIA2_ERR_LIB_THIS_IS_NO_SOLEA_SPM_MODULE	-9017	LIB: this is no Solea SPM module
SEPIA2_ERR_LIB_THIS_IS_NO_LMP_828_MODULE	-9018	LIB: this is no laser test site module
SEPIA2_ERR_LIB_THIS_IS_NO_SOM_828_D_MODULE	-9019	LIB: this is no SOM 828 D module
SEPIA2_ERR_LIB_NO_MAP_FOUND	-9020	LIB: no map found
SEPIA2_ERR_LIB_DEVICE_CHANGED_RE_INITIALISE_USB_DEVICE_LIST	-9025	LIB: device changed, re-initialise USB device list
SEPIA2_ERR_LIB_INAPPROP_USBDEVICE	-9026	LIB: Inappropriate USB device
SEPIA2_ERR_LIB_WRONG_USBDRIVER_VERSION	-9090	LIB: wrong USB driver version
SEPIA2_ERR_LIB_UNKNOWN_LIBFUNCTION	-9900	LIB: unknown library function
SEPIA2_ERR_LIB_ILLEGAL_PARAMETER	-9910	LIB: illegal parameter on library function call
SEPIA2_ERR_LIB_UNKNOWN_ERROR_CODE	-9999	LIB: unknown error code

4.3. Index

Table of API-Functions

PicoQuant GmbH
Rudower Chaussee 29 (IGZ)
12489 Berlin
Germany

P +49-(0)30-6392-6929
F +49-(0)30-6392-6561
info@picoquant.com
http://www.picoquant.com