REGATRON
www.regatron.ch

Application Programming Interface for Dynamic Link Library

For Version DLL 3.56(upwards)

as of: August 2012

Regatron AG
Kirchstrasse 11
CH-9400 Rorschach
Tel +41 71 846 67 44
Fax +41 71 846 67 77
www.regatron.com
TopCon@regatron.ch

Programming Manual for DLL 3.56

Device control using TopCon API (DLL 3.56 / rev 2.5,e)

Authors: Stefan Baldauf , Markus Mueller, Claus Eikemeier, Fredy Nüesch
Last changes: 08-2012, FN

Version history

Revision	DLL Version	Date of Release	Author	Remarks
1.0	1.01	02-2001	Stefan Baldauf	initial release
1.1	1.02	04-2001	Stefan Baldauf	correction of LabView calling convention
1.2	1.02	06-2001	Stefan Baldauf	changed resistance unit to mOhms
1.3	1.06	08-2001	Stefan Baldauf	additional info about TCGetPhysicalValuesIncrement
1.4	2.02	08-2001	Stefan Baldauf	added function TCGetPhysicalValuesLimits
2.0	2.90	10-2003	Markus Mueller	added description of TC4-functions (firmware 4.x) and table of content. Changed chapter numberings.
2.1	2.90	01-2010	Claus Eikemeier	Updated example section
2.2	2.90	01-2010	Claus Eikemeier	Note on using the pragma pack compiler command
2.3	3.40	12-2011	Fredy Nüesch	AAP functions added from document “DLL functions forTC_AAP”
2.4 a/b	3.42	03-2012	Claus Eikemeier	Further functions for TFE/AAP curves
2.5	3.56	08-2012	Fredy Nüesch	Q4 functions implemented AAP InputScaling amended How to use the specific function description in html file and general staff is described in this document

This Document describes the ‘Application Programming Interface (API)’ provided by the
TopCon I/O Module ‘tcio.dll’ / ‘serialio.lib’. The API is implemented as a ‘Dynamic Link Library (DLL)’ for the Windows operating system. Most functionality has also been compiled as a shared object to be used in Linux based systems.

Content

1 Provided functionality / API

This API provides the following functionality:

- Handling of communication to the device
- Getting the actual state of the device
- Controlling modes of operation
- Reading/Clearing faults/errors and warnings
- Remote Control input selection
- Setting reference values for output
- Getting actual output values
- Handling the number of modules in system
- Reading temperature, serialnumber
- Setting/reading controller parameterisation
- Setting/reading overvoltage/overcurrent settings
- Reading internal supply-voltages and currents
- Reading device-info [1]

The API functions are declared as ‘c functions’.
Each function returns an integer value called ‘DLL_DONE’ which shows if the action was successful. For easy implementation it is useful to define the return values as constants.

All DLL functions have a return value DLL_SUCCESS (0) or DLL_FAIL (-1).
For example – using the API in C/C++ include the following lines in your project:
#define DLL_SUCCESS 0
#define DLL_FAIL –1

Remarks on data types

Type	Mem requirement	Remarks
char	1 Byte	Integer 0..255
unsigned int	4 Bytes	Integer 0.. 4294967295
int	4 Bytes	Integer -2147483648.. 2147483647
double	8 Bytes	Floating-point
long	4 Bytes	Integer -2147483648.. 2147483647
unsigned long	4 Bytes	Integer 0.. 4294967295
short	2 Bytes	Integer -32767.. 32768
unsigned short	2 Bytes	Integer 65536

Remark on the implementation

The implementation of the DLL uses

```
#pragma pack(push,1)
…
#pragma pack(pop)
```

for the definition of the structs.
Make sure that your software also uses this alignment of the different variables in the structs so that data transmission from/to the TopCon device succeeds.

(For further information on the effect of the pragma pack / push command, see URL: http://www.geekpedia.com/tutorial61_Size-of-Structures.html)

In addition to the functions for some action on the device there are two important calls to implement in the application: The '*InitDLL'* and '*CloseDLL'* functions have to be called every time the application starts/ends. These functions are managing the construction/destruction of the objects used by the API. When things seem to be strange to the application (communication errors, device errors, …) it is possible to reinitialise using *CloseDLL* and *InitDLL*. Be sure to use *CloseDLL* after each *InitDLL* even if *InitDLL* returns *DLL_FAIL*. After *InitDLL* the communication has to be set up again with *DllSearchDevice* (see further explanation in this manual).

For detailed meaning of parameters please read TopCon manual.

In the following documentation a distinction is drawn between three different types of API-functions:

- functions to initialise/close the DLL (functions beginning with "Dll…"), see chapter 2.2.1
- functions for programming TopCon with firmware 4.x (functions beginning with "TC4…",

2 API specification

The function declarations used in this docuemt are 'ANSI C'-style.

2.1 DLL initialisation

2.1.1 Initialising

int DllInit()
Initialises Driver/Objects used by the API. Allocates Memory.

int DllClose()
Closes Driver/Objects used by the API. Releases Memory.

*int DllReadVersion(unsigned int *p_version, unsigned int *p_build, char *p_string)*
Reads version of API

Parameter list:

p_version	pointer to actual version of API upper 16 Bit: Version number (<number>.00) lower 16 Bit: Subversion number (01.<number>)
p_build	pointer to build counter
p_string	pointer to version string, max. 50 character

2.1.2 Device communication

*int DllSearchDevice(int fromPort, int toPort, int *p_portnrfound)*
Searches for TopCon device on available serial ports between the given port numbers.
If a device is found communication will be established automatically.
To connect on a specific port number set *fromPort* equal to *toPort*.
Usually the two serial ports on a PC have the numbers 1 and 2.

Parameter list:

fromPort	port number from which the search starts (1 to 255)
toPort	port number where the search stops (1 to 255) note: fromPort <= toPort
p_portnrfound	pointer to variable receiving the port number if device found otherwise unchanged

*int DllGetStatus(int *p_state, int *p_errorno)*

Reads actual state of communication and DLL.
This function reports the status of the low level interface (communication or communication commands). If an error occur the communication might be interrupted or the actual command could not be executed – a new initialisation is recommended.
This function does not report device faults or warnings!

Parameter list:

p_state		pointer to variable receiving actual state
	0	ok
	-10	communication error
	-100	device reported command execution error
p_errorno	reserved	

2.2 Programming TopCon with firmware 4.x, 4.2x

Before using any of these API-functions please read chapters 2.2.1 to 2.2.3.

2.2.1 Important initialising functions

We suppose DLL-initialisation has already been made (see chapter 2.1.1)

These two functions must always be called after DLL-initialisation or when changing the connection to an other TopCon module:

- *TC4GetPhysicalValuesIncrement*
- *TC4SetModuleSelector(64)*

API-function *TC4GetPhysicalValuesIncrement* will calculate internally used factors which other TC4-API-functions will use.
API-function *TC4SetModuleSelector(64)* will set the internal ModuleSelector to 64. This ensures that in multi-unit operation a TopCon Master will return *system*-values.

2.2.2 Translate between standardised values and physical values

Some API-functions use standardised values instead of physical values.
This chapter describes how to correctly translate between standardised values and physical values. For correct translation you need to know what standardised value corresponds with what physical value.

Translation is easy as long as just one single master module is used (single-unit operation)
The API-function *TC4GetPhysicalValues* gives you all the physical values that correspond to a standardised value of 4000.
This API-function returns 7 physical values. The following table tells you which of these physical values must be used to translate between what type of values:

this physical value…[1)]	…belongs to a standardised value of…	when translating a…
vphys [V]	4000	module output voltage
iphys [A]	4000	module output current
pphys [kW]	4000	module output power
rphys [mOhm]	4000	simulated module resistance
vzphys [V]	4000	dc-link voltage
ippys [A]	4000	primary transformer current
tphys [°C]	4000	temperature

[1)] returned by API-function *TC4GetPhysicalValues*

E.g.: to convert a standardised temperature value (which you get when calling API-function *TC4GetTempDigital*) into a physical temperature value:

$$\text{temp\_physical [°C]} = \frac{tphys}{4000} \times temp\_standardised$$

In a TopCon system with more than one module (multi-unit operation) a differenciation has to be made between *module* output value and *system* output values (also see chapter 2.2.3).
When translating between standardised and physical *module*-output values the same physical values are used as in single-module operation.
When translating between standardised and physical *system*-output values the corresponding physical values are different from those of one module, depending on how the system is configured (series-, parallel-, matrix-, or multiload-operation). To get the correnct physical *system* values, API-function *TC4GetSystemPhysicalValues* must be used.

The following table tells you which physical value must be used to translate between what types of values:

this physical value…[1)]	…belongs to a standardised value of…	when translating a…
vphys[V]	4000	system output voltage
iphys [A]	4000	system output current
pphys [kW]	4000	system output power
rphys [mOhm]	4000	simulated system resistance

[1)] returned by API-function *TC4GetSystemPhysicalValues*

Physical values for dc-link voltage, primary transformer current and temperature remain the same as in single-unit operation.

To tell whether *module*- or *system*-values are used, please see next chapter.

2.2.3 Difference between module- and system-values

Module-values are always used setting or reading values concerning just one module.
System-values are used in multi-unit operation when setting values concerning all connected modules.

E.g.: suppose we have three TopCon modules, each with the following nominal data:
Unom = 60V, Inom = 333A, Pnom = 16kW

when connecting them in series we get a system with the following nominal data:
Unom = 3x60V=180V, Inom = 333A, Pnom = 3x16kW = 48kW

When we use the API-function *TC4SetVoltageRef* to set a desired voltage set value of, lets say 90V, we set the *system*-value. Each module will deliver its *module*-value of 30V.

When connected to a TopCon Master in single-unit operation API-Get-functions return *module*-values and API-Set-functions require *module*-values.
When connected to a TopCon Master in multi-unit operation API-Get-functions return *system*-values and API-Set-functions require *system*-values.

When connected to a TopCon Slave API-Get-functions return *module*-values and API-Set-functions require *module*-values.

API-functions not following this standard will be denoted seperately.

API-functions do not directly know where they are connected to. They only can return correct values if they are told how to interpret the values.
This means that every time you connect to an other module you must call the initialisation-functions described in chapter 2.2.1 on page 8.

2.2.4 Device control

*int TC4GetSenseDigital(unsigned int *vsense)*

This function is not implemented in hardware yet. Calling this function will set **vsense* to 0 (zero).

2.3 GSS Related Functions

Functions related to power feedback path (load is power source, the TopCon device is the sink, feeding back energy to the mains (in case of TC.GSS and TC.ReGen systems) is not covered in this documentation.
It will be available in a set of help files (HTML format) or can be received from the TopCon support staff.

3 TopCon Function Engine / Handling AAP-slopes

This document highlights the special DLL-instruction set and procedures related to TopCon AAP functionality:

- Scaling of slopes in x- and y-direction
- Application of ‚soft blending' at transition from one slope to the next one
- Loading and storing of slopes from and into the internal flash memory

3.1 Compatibility of TFE (function generator) functionality

In order to ensure compatibility between different versions of firmware and those of the DLL, a version n umber (FnSeqVersion) is stored together with each function sequence.

FnSeqVersion	Comment
0	First version DLL V3.07.00 and MainDSP V4.11.62 or older.
1	Structure of T_FnBlock extended by "AAPFlags" Needs DLL V3.08.00 and MainDSP firmware V4.11.63 or newer
2	Structure of T_FnBlock extended by "AAPInputScaling" Needs DLL V3.10.00 and MainDSP firmware V4.11.67 or newer

Note: Query instruction for the actual supported version:

int TC4GetFnSeqMaxSupportedVersion(unsigned int *pMaxVersion)

The user may inform the DLL about the version intended to use. This is of particular interest, if features of FnSeqVersion > 0 are to be used. New features may add fields to the structure data, which aren't readable by older versions.
If a data structure (struct) is given to the DLL, it is assumed as struct of version 0. This is to prevent the DLL from accessing non-initialized structure values. → DLL assumes that the user is knowing version 0 only.

By the function

int TC4SetFnSeqMaxUserCompatibleVersion(unsigned int MaxUserVersion)

the user tells the DLL which version he's intending to use.

Example: The field "AAPInputScaling" of the structure T_FnBlock will be read or written only if the user initialized this function with the argument 2.

Note: To store a slope, the version field of the structure T_FnSeqHeader must be set to the same value. If not, all slope values not compatible to the actual version will be set to default values at the time of loading the slope.

3.2 Loading of AAP slopes during operation

Requirements: DLL version V2.94.00 or up necessary.

From firmware V4.11.52 and up it is possible to sequencially load new slopes during full operation of the TopCon power supply. Former versions allowed loading only during ‚Voltage OFF' state.

From firmware V4.11.67 and up and DLL V3.10.00 and up a new feature can be used: Definition of „AAP soft-cutting" from the existing AAP dataset to the newly loaded set is now possible. This means a continuous transition from one slope to the next one in a freely programmable ramp time is available.

Please refer to **chapter 3.4.2** for this feature.

3.3 Internals

The following sections need in-depth knowledge of the internal structure of the TopCon function engine. The next sections are excerpts from the programmer documentation and therefore should only be used by trained personal.

3.3.1.1 Structure T_FnSeqHeader

```
struct T_FnSeqHeader
{
        unsigned int    TotalSize;s
        unsigned int    UserDefSize;
        unsigned int    Version;
        unsigned int    SeqNumber;
        char                    SeqName[32];
        unsigned int    DateYear;
        unsigned int    DateMonth;
        unsigned int    DateDay;
        unsigned int    TimeHour;
        unsigned int    TimeMinute;
        unsigned int    TimeSecond;
        unsigned int    SysNomVoltage;
        unsigned int    SysMaxCurrent;
        unsigned int    SysNomPower;
};
```

Entry	Meaning
TotalSize	Ignored, this value is computed internally May be set to zero
UserDefSize	Ignored, this value is computed internally May be set to zero
Version	FnSeqVersion Shows added features of the Function Sequence 0: First version 1: Additionally Supports AAP-Flags, → T_FnBlock 2: Additionally Supports AAP-Input-Scaling, → T_FnBlock FnSeqVersion has to be set correctly, otherwise all slope values not corresponding to the actual version are set to default values at the time of loading the slope.
SeqNumber	Sequence number of slope.
SeqName	String with max. 32 characters (incl.. zero termination mark)
DateYear	Date of storage (year, 1971...2038)
DateMonth	Date of storage (month, 1...12)
DateDay	Date of storage (day, 1..31)
TimeHour	Time of storage (hour, 0..23)
TimeMinute	Time of storage (minute, 0..59)
TimeSecond	Time of storage (second, 0..59)
SysNomVoltage	Nominal voltage of system (*)
SysMaxCurent	Max. current of system (*)
SysNomPower	Nominal power of system (*)

(*) These values can be determined by function

```
int TC4GetSystemPhysicalValues(int *vphys, int *iphys, int *pphys, int *rphys)
```

*vphys: Nominal voltage [V], *iphys: max. current [A]
*pphys: nominal power [kW], *rphys: max value for internal resistance simulation [mOhm]

3.3.1.2 Structure T_FnSeq

Basic settings for all function blocks (voltage, current, power)

```
struct T_FnSeq
{
        unsigned int        StartTrigger;
        unsigned int        EndAction;
        double              Delay;
        unsigned int        Repeat;
        unsigned int        EnabledFnBlocks;
        int                 GeneralEnable;
};
```

Entry	Meaning
StartTrigger	Start mode of the function sequence 0: at VoltageON 1: manual trigger via TopControl or HMI (*) 2: manual trigger by Pin 19 of interface connector X105 (high state) (*) (*) Condition: The power supply is already at ‚VoltageON' condition
EndAction	Action after termination of time-dependent sequence 0: DC output disabled (VoltageOFF) 1: hold the actual DC voltage 2: set voltage to a value defined by HMI / TopControl / interface connector X105 If using AAP – which is not time-dependent – no automatic termination will be given
Delay	Time delay in [50µs] between two repetitions of function Sequences (0... 4'294'967'295) ; Ignored if no time-dependent function is active
Repeat	Number of repetitions of Function Sequences (0... 4'294'967'295, 0= continuous) ; Ignored if no time-dependent function is active
EnabledFnBlocks	Used Function Blocks 1: Voltage (voltage determined by a) (AAP) slope or b) time function) 2: Current (current determined by a) (AAP) slope or b) time function) 4: Power (power level determined by a) (AAP) slope or b) time function) **Addition of values** means simultaneous operation of different Function Blocks (*e.g.* 3: Function Blocks for voltage and current are operative at the same time) A value of 0 means no Function Block operative.
GeneralEnable	Global ON/OFF Flag 0: Function engine TFE disabled 1: Function engine TFE enabled

3.3.1.3 Structure T_FnBlock

```
struct T_FnBlock
{
        unsigned int    BaseFunction;
        double          Amplitude;
        double          Offset;
        double          Symmetry;
        double          Frequency;
        int                     BipolarAmplitude;
        int                     RectifyAmplitude;
        double          ExpTimeConstant;
        unsigned int    NumPeriods;
        unsigned int    UserDefAmplitude;
        unsigned int    UserDefTimePrescaler;
        double          UserDefPeriodLength;
        unsigned int    UserDefNumPoints;
        unsigned int    InactiveLevelType;
        double          InactiveLevel;
        unsigned int    AAPInputType;
        unsigned int    AAPInputFilterKoeff;
        unsigned int    AAPFlags;
        double          AAPInputScaling;
};
```

Entry	Meaning
BaseFunction	Type of slope 0: Time dependent sine slope 1: Time dependent square slope 2: Time dependent triangle slope 3: Time dependent 'user-defined' slope 4: Functional (AAP) slope → non time domain!
Amplitude	Amplitude in [V], [A], or [kW], depending on *type*
Offset	Offset in [V], [A], or [kW], depending on *type*
Symmetry	0.00..100.00 [%], Duty cycle of square and triangular slopes (BaseFunction 1 or 2)
Frequency	0.01...some 100 [Hz] Frequency of sine, square and triangle functions (BaseFunction 1..3) → The obtainable upper frequency will depend on load time constant, modulation amplitude and the settings of controllers. If running 'user-defined' slopes (BaseFunction 3) this value has no meaning.
BipolarAmplitude	0: for unipolar voltages (0 volts to 'amplitude') 1: for bipolar voltages (-amplitude...+amplitude) → additional circuitry needed → If using 'user-defined' slopes or AAP slopes (BaseFunction 3 oder 4), set this value to 1. (to avoid an unwanted amplitude offset)
RectifyAmplitude	0: rectifying of negative amplitude disabled (default) 1: negative amplitude values will be turned into positive values (prior to offsetting)
ExpTimeConstant	0... 4'294'967'295 in intervals of [50µs], Usage: Create an exponentially falling amplitude envelope. Step interval is [50µs], e.g. (20'000 equals a time constant of 1s) A value of 0 will deactivate the function (default) AAP will ignore this function
NumPeriods	0...65535 Number of repetitions, valid for time-dependent slopes. (AAP ignores this value) 0 means continuous operation

Entry (cont.)	Meaning
UserDefAmplitude	Absolute maximum amplitude value for ‚user-defined functions' (BaseFunction 3 and 4), this is used for internal scaling → Define as normalized values (0...4000) corresponding to system parameters; If type = 0: 4000 equals the unit's nominal voltage = 1: 4000 equals the unit's nominal current = 2: 4000 equals the unit's nominal power For further info on power supply parameters see chapter 3.3.1.
UserDefTimePrescaler	(1...65535) Multiplier for ‚user-defined' functions time base (only BaseFunction 3)
UserDefPeriodLength	Not used A value of 0 is applicable
UserDefNumPoints	Number of value points in 'user-defined' functions (BaseFunction 3 and 4) Value range: BaseFunction 3 ('user-defined' time function): 2...1000 BaseFunction 4 (AAP): 2...64 All other BaseFunctions: Use 0
InactiveLevelType	Not yet used. Value of 0 applicable
InactiveLevel	Not yet used. Value of 0 applicable
AAPInputType	Input values for AAP slopes → independent variable of AAP function 0: Actual value of voltage 1: Actual value of current 2: Actual value of power *Example: Create an U(I) – slope !* → choose AAPInputType = 1 to create a voltage Function Block
AAPInputFilterKoeff	In order to avoid input noise from influencing the AAP calculation, a first-order low pass filter can be defined to smooth AAP input values. Filter frequencies : 0: Filter disabled 1: 1600 Hz 2: 800 Hz 3: 400 Hz 4: 200 Hz 5: 100 Hz 6: 50 Hz 7: 25 Hz 8: 12.4 Hz 9: 6.2 Hz 10: 3.1 Hz 11: 1.6 Hz 12: 0.8 Hz 13: 0.4 Hz 14: 0.2 Hz 15: 0.1 Hz
AAPFlags	Not yet used, available from FnSeqVersion 1 and up. Set to 0 prior to saving.
AAP-Input-Scaling	Available from FnSeqVersion 2 and up. Used for scaling of AAP curves in x-axis direction. Value range: 0.125 ... 10.0 ; this is the actual scale multiplier The scaling function implemented can be calculated according to : Xorig * 4096 = Xscaled * InputScalingValue Example "Shrink curve in Xdirection by 0.5":

	Xorig/Xscale = 2 --> curve changed by ½ --> InputScalingValue = 2*4096=8192

3.4 Manipulating AAP slopes

3.4.1 Amplitude - , offset - and AAP-input-scaling

int TC4GetFnBlockSettings(struct T_FnBlock *p_fnblock, unsigned int type)
int TC4SetFnBlockSettings(struct T_FnBlock *p_fnblock, unsigned int type)

type: FunctionBlock Type: voltage (0), current (1) or power (2)

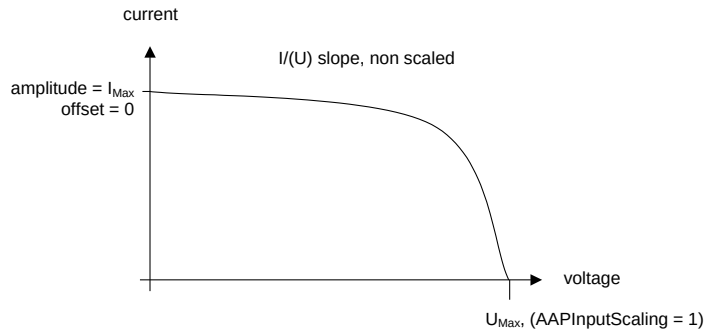
This value determines the type of AAP output (dependent variable of AAP function)

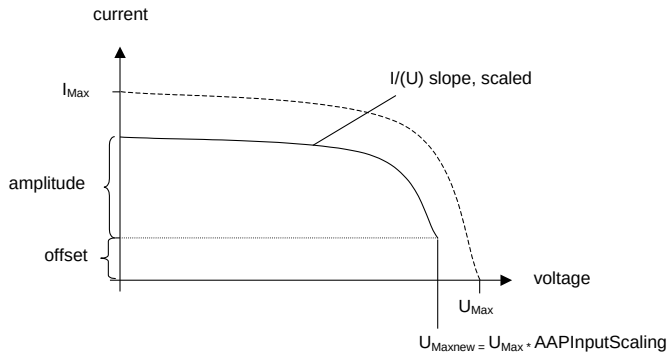
Example:
Type = 1 means: Output of AAP functionality is a **current set value.**
(TopCon TFE function engine will generate a current output based on an input voltage, if input is defined as voltage in T_FnBlock)

After initialising structure **T_FnBlock** (chapter 3.3.3) correctly by :
a) DLL function *TC4SetFnBlockSettings* , or
b) loading of a slope from internal flash and reading by *TC4GetFnBlockSettings*
the following entries can be used to alter the slopes online:

- Amplitude
- Offset
- AAPInputFilterKoeff
- AAP-Input-Scaling

Example:
The following I vs.U - slope show the influence of the various parameters:





3.4.2 Ramp time definition

A ramp time allow for smooth and continuous process flow of scaling, amplitude modification and AAP slope changeover actions.

Note that ramping actions will run independently in the firmware of TopCon power supply, so the communication between TopCon and DLL is not affected.

Set ramp time via the following functions:

```
int TC4GetFuncgenRampTime(double *p_ramptime)
int TC4SetFuncgenRampTime(double p_ramptime)
```

Ramptime value is given in seconds.
Value range: 0.05ms ... 86400s (1 day)

Note: Requirements for ‚Ramp timing':

- Firmware V4.11.67 or newer
- DLL 3.10.00 or newer

Note: The ‚Ramp time' can also be used to ensure a soft blending from a running AAP slope to the next one just loading from flash. This will avoid a sudden step change of operating point. See also chapter 3.6 for further information.

A change of ramp time value will not affect an already running ramp. The changes will take place just after completion of the running ramp cycle.

3.5 Storing AAP slopes

To store AAP slopes, the following functions are available. Functions described in chapters 3.5.1 up to 3.5.3 may be programmed in any order. Please note, that functions 3.5.4 and 3.5.5 have to follow functions 3.5.1 to 3.5.3 and have to be in ascending order.

3.5.1 Storing header information

Call for DLL- function

```
int TC4SetFnSeqHeader(struct T_FnSeqHeader *p_header)
```

For information on T_FnSeqHeader see chapter 3.3.3.1.1.

3.5.2 Function Sequence : Storing settings

Call for DLL-function

int TC4SetFnSeqSettings(struct T_FnSeq *p_fnseq)

For information on T_FnSeq see chapter 3.3.1.2.

3.5.3 Store function block settings

Call for DLL-function

int TC4SetFnBlockSettings(struct T_FnBlock *p_fnblock, unsigned int type)

Call this function for each active function block defined in "EnabledFnBlocks", (see T_FnSeq)

Parameter	Meaning
type	Type of value: Voltage (0); current (1); power (2)

For information on T_FnBlock refer to chapter 3.3.1.3.

4 Examples

4.1 General

Initialisation (use in the listed order):

```
DllInit                          // init API
DllReadVersion                   // check API version
DllSearchDevice(1,4, &pnr)       // search for device from COM1 to COM4
DllGetStatus(&state, &errno)     // check state of API is ok (zero)
```

Only for Firmware 4.x:

```
TC4GetPhysicalValuesIncrement(&vf, &cf, &pf, &rf, &vsf, &isf, &psf,
&rsf)                            // read factors
TC4SetModuleSelector(64)         // see that API-function
TC4StateActSystem                // read actual device state
```

After this sequence the communication to the device is established.
To ensure everything is ok check return values (equal to DLL_SUCCESS – zero).
TC(4)GetPhysicalValuesIncrement has to be called even if you use the factors or not.

Closing Application:

```
DllClose                         // close AP
```

Reinit after communication-error:

```
DllClose                         // don't forget
DllInit                          // (re-)init API
DllSearchDevice (1,4, &pnr)      // search for device from COM1 to COM4
DllGetStatus (&state, &errno)        // check state of API is ok (zero)
```

For Firmware 4.x:

```
TC4GetPhysicalValuesIncrement(&vf, &cf, &pf, &rf, &vsf, &isf, &pfs,
&rfs)                            // read factors
TC4SetModuleSelector(64)         // see that API-function
TC4ClearError                    // force device to clear errors
TC4StateActSystem                // read actual device state
```

4.2 Using the API in C++

Define constants

```
#define DLL_SUCCESS          0
#define DLL_FAIL            -1
#define DLL_STOK             0
#define DLL_STCOMMERROR    -10
#define DLL_STDEVICEERROR  -100
```

Define types for each function

```
typedef int (__cdecl* T_DllInit)();
typedef int (__cdecl* T_DllClose)();
typedef int (__cdecl* T_DllSearchDevice)(int fromPort,
                                         int toPort, int *p_portnrfound);
```

4.3 Define Function Pointers

```
T_DllInit          pf_DllInit;
T_DllClose         pf_DllClose;
T_DllSearchDevice  pf_DllSearchDevice;
```

Define Handle to Library

```
HINSTANCE hDlltcio;
```

at an initialisation point in your application (i.e. CMyAPP::InitInstance)

```
hDlltcio = LoadLibrary("tcio.dll");
if (hDlltcio != NULL) {
        pf_DllInit = (T_DllInit)GetProcAddress(hDlltcio, "DllInit");
        pf_DllClose = (T_DllClose)GetProcAddress(hDlltcio, "DllClose");
        pf_DllSearchDevice = (T_DllSearchDevice)GetProcAddress(hDlltcio,
                                                    "DllSearchDevice");
}

pf_DllInit();
result = pf_DllSearchDevice(1, 4, &pnr);
…(proceed with DllGetStatus, TC4GetPhysicalValuesIncrement)
pf_DllClose();
```

Check function pointers if they are equal to NULL before calling.
Call other functions like *pf_DllSearchDevice*.

4.4 Using the API in LabView

This API can also be used with LabView.
Create the desired VI (Library) and direct it to ‘tcio.dll’.
Specify the DLL calls as ‘C-type’ – do not use ‘Standard’, because LabView crashes otherwise.
After specifying the parameters you can use the VI as usual.

4.5 Example (programming language: C)

4.5.1 TFE Example - program code

```
// Example: using the TopCon DLL to control the TopCon Function Generator
//
// Valid for TopCon-DLL V3.00.00 and higher
//
// 2005.12.07/MM; 2010-02-08/CE;
//

// ================================================================================
// *** Loading a function sequence from the non volatile memory in the TopCon Device RAM
// ================================================================================

unsigned int SeqNr; // Function Sequence to be loaded, identified by Sequence No(1...1000)

TC4SetFnSeqActionLoad(SeqNr);

int result;
do
{
  TC4GetFnSeqActionResult(&result);
  Sleep(500); // wait for 500ms (do not block other processes during wait)
}
while(result > 0)

// result > 0: Function Sequence loading not yet finished
// result = 0: loading finished successful
// result < 0: error during loading


// ================================================================================
// *** Reading of the function sequence into the application       ***
// ================================================================================

TC4GetFnSeqHeader(&header);

TC4GetFnSeqSettings(&fnseq);

// *** read function block „Voltage“ ***
unsigned int type = 0;
TC4GetFnBlockSettings(&fnblock, type);

if (fnblock.BaseFunction == 3) // User defined time function
{
  unsigned int readsize;
  unsigned int timeDelta[8]; // normalized, computed according to remark at 'fnblock.UserDefTimePrescaler', see blow
  int amplitude[8]          // normalized, 0...4000, 4000=100% nominal voltage (100% maximal current, 100% nominal power
respectively)
  int size;

  TC4SetFnBlockUserDataStartAddr(type);

  size = fnblock.UserDefNumPoints;
  while (size > 0)
  {
    TC4GetNextUserTimeData(type, &timeDelta[0], &amplitude[0], fnblock.UserDefAmplitude, &readsize);

    size -= readsize;

    // save timeDelta and amplitude in separate variables, because these are overwritten
    // during next run
    // In 'readsize' the effective number of read points is given (8 or less)
  }
}
else if (fnblock.BaseFunction == 4) // AAP curve
{
  unsigned int readsize;
  int input[8];
  int amplitude[8];
  int size;

  TC4SetFnBlockUserDataStartAddr(type);
```

```
  size = fnblock.UserDefNumPoints;
  while (size > 0)
  {
    TC4GetNextUserAAPData(type, &input[0], &amplitude[0], fnblock.UserDefAmplitude, &readsize);

    size -= readsize;

    // todo:
    // store input and amplitude in separate variables  because these are overwritten
    // during the next run
    // In 'readsize' the effective number of points read is given (8 or less)
}

// *** End of  read function block “Voltage” ***



// read  current and power function block in an analog way
// (type=1 --> current, type=2 --> power)


// ##############################################################################################
// ##############################################################################################

// *** Storing a function sequence to flash memory by the application ***

// prepare Header-Informationen
struct T_FnSeqHeader header;

header.TotalSize = 0;   // do not care (re-evaluated internally)
header.UserDefSize = 0; // d.n.c.
header.Version = 0;     // Current version is 0 (for future purpose)
header.SeqNumber =      // 1...1000, Sequence-Nummer for identification
                        // Caution: if a different sequence number is given during store to
                        // flash function, this value is replaced.
header.SeqName =        // arbirarily choosable, max. 32 characters incl. trailing zero byte
header.DateYear =       // 1970...2037, arbitrarily choosable
header.DateMonth =      // 1...12
header.DateDay =        // 1...31
header.TimeHour =       // 0...23
header.TimeMinute =     // 0...59
header.TimeSecond =     // 0...59
header.SysNomVoltage =  // Nominal voltage of system [V]
header.SysMaxCurrent =  // Maximal current of system [A]
header.SysNomPower =    // Nominal power of system [kW]

// Store header in RAM of device
TC4SetFnSeqHeader(&header);


// Prepare function sequence settings
struct T_FnSeq fnseq;

fnseq.StartTrigger =    // 0...2, 0: start at VoltageON, 1: manuel via TopControl / API, 2: manuel by interface X105/Pin19
fnseq.EndAction =       // 0...2, 0: VoltageOFF, 1: keep actual preset value, 2: chose preset value from standard input (RS232,
HMI, analog)
fnseq.Delay =           // in seconds, 50us...100'000s (50us per step)
fnseq.Repeat =          // number of repetitions, 0...2'000'000'000, (0=continuously)
fnseq.EnabledFnBlocks = // Used function blocks, Bit 0: voltage, Bit 1: current, Bit 2: power
fnseq.GeneralEnable =   // 0 or 1, deactivates or activates the whole function sequence

// Function Sequence: Store settings in RAM of device
TC4SetFnSeqSettings(&fnseq);


// Prepare function block settings

int type, i;
for (type=0;type<=2;type++) // type: 0=voltage, 1=current, 2=power
{

  struct T_FnBlock fnblock;

  fnblock.BaseFunction =     // 0: sinus, 1: rectangle, 2: triangle, 3: user def. time based function, 4: application area
programming (AAP)
  fnblock.Amplitude =       // 0...100% of nominal values of system. In [V], [A] or [kW]
```

```
  fnblock.Offset =          // -100%...+100% of nominal values of system. In [V], [A] or [kW]
  fnblock.Symmetry =        // 0...100, Dutycycle [%]. Nur bei Rechteck und Dreieck
  fnblock.Frequency =       // 0.001...1000 [Hz]. Only for sinus, rectangle and triangle  fnblock.BipolarAmplitude =   // 0 or 1, only
for sinus, rectangle and triangle
fnblock.RectifyAmplitude =   // 0 or 1:  1= negative amplitudes are rectified (e.g. for sinus
                                        // rectification)- only for sinus, rectangle and triangle
  fnblock.ExpTimeConstant =   // 0...100'000, time in [s,  50 µs steps] for exponentially
                                        // decreasing amplitude
                        //    should be set to 0 if no fading wanted
  fnblock.NumPeriods =      // 0...65535, number of periods of base function, not for AAP
                                        // (AAP functions operate continously)
                        // 0 for continuous repetition
  fnblock.UserDefAmplitude = // 0...4000, only for user defined time based function and AAP
                                        // A reference level has to be stored as user def. points are
                                        // stored in a normalized way.
                                        // The biggest value (according to amoung) of a user def. curve // has to be stored here
                                        a amount.
                        // Example:
                                        // A TopCon system with 200V nominal voltage carries a U=func(I) // curve. The value
                                        for UserDefAmplitude is now
                                        //  3000 = (4000 * |U_UserDefMax| / U_Nenn)
  fnblock.UserDefTimePrescaler =
                                        // 1...65535, time base for user def. functions
                                        // The time differece between two user def. points is stored in // a normalized way.
                                        // Conversion between normalized time and time in seconds:
                         // TimeDelta [s] =
                                        //     TimeDelta [Normalized] * UserDefTimePrescaler * 50us
  fnblock.UserDefPeriodLength = 0; // not used, can be set to 0
  fnblock.UserDefNumPoints =     // Number of user def. poitns, only for baseFunction 3 und 4,
                          // for other base functions: set to 0
                          // 0...1000 (user def. time based function)
                          // 0...64 (AAP)
  fnblock.InactiveLevelType = 0; // 0, further values for future purpose
  fnblock.InactiveLevel = 0;     // 0, further values for future purpose
  fnblock.AAPInputType =         // With AAP functions: type of X-axis (input)
                          // 0: voltage, 1: current, 2: power
                                    // (example: U(I) characteristics: AAPInputType = 1)
  fnblock.AAPInputFilterKoeff =  // 0...15, filter bandwidth for AAP input signal
                          // 0:no filter , 1:1600Hz, 2:800Hz, 3:400Hz, ...,15: 0.1Hz


  // Store function block setting in RAM of device
  TC4SetFnBlockSettings(&fnblock, type);

} // Repetition for voltage, current and power


// After storing header, function sequence settings and function block settings in RAM of the device, storing of these data in the
non-volatile memory (flash) of the device is initiated.

unsigned int seqNr =  // 1...1000, Sequence Nummer of the function sequence to be stored
                                  // !! CAUTION: a previously used function with this number is
                                  // overwritten without any further inquiry.

int copyUserData = 0;

TC4SetFnSeqActionStore(SeqNr, copyUserData);

int result;
do
{
  TC4GetFnSeqActionResult(&result);
  Sleep(500); // wait for 500ms (do not block other processes)
}
while (result > 0)

// result: >0 storing not yet finished
//         =0 storing finished successfully
//         <0: error during storing, process aborted


// User defined point have not been stored up to this point.
// Reason for this: User defined points cannot be stored in RAM due to missing space in RAM.
// Thus they cannot be stored automatically in the FLASH memory.
//
//User defined points need to be stored separately:
// *** Storing user defined points in FLASH    ***
```

```
//     (along the example „voltage“            ***)

int type=0; // 0:voltage, 1: current, 2: power
struct T_FnBlock fnblock = VoltageFnBlock;

// Setting the starting address in FLASH, thus storing the points correctly in memory.
TC4SetFnBlockUserDataStartAddr(type);

if (fnblock.BaseFunction == 3) // User defined time based function
{
  unsigned int writesize;
  unsigned int timeDelta[8]; // Time difference between actual point and previous point
                                          //(always positive value)
                        // for 1st point, the amplitude is 0
                        // normalized, processing according to remark
                                          // 'fnblock.UserDefTimePrescaler', see above
int amplitude[8]              // normalized, 0...4000, 4000=100% nominal voltage (= 100% max
                                          // current, = 100% nominal power)
  int size = fnblock.UserDefNumPoints;

  while (size > 0)
  {
    if (size > 8)
      writesize = 8;
    else
      writesize = size;

    // todo:
    // initialize timeDelta[] und amplitude[], to always get the the next “writesize” number
              // of points stored.

    TC4SetNextUserTimeData(type, &timeDelta[0], &amplitude[0], fnblock.UserDefAmplitude, writesize);

    size -= writesize;
  }
}
else if (fnblock.BaseFunction == 4) // AAP
{
  unsigned int writesize;
  int input[8];   // normiert, 0...4000, 4000=100% Nennspannung (bzw. 100% Maximalstrom, 100% Nennleistung)
  int amplitude[8]; // normiert, 0...4000, dito
  int size = fnblock.UserDefNumPoints;

  while (size > 0)
  {
    if (size > 8)
      writesize = 8;
    else
      writesize = size;

    // todo:
    // initialize input[] und amplitude[], --> the next 'writesize' number of points can be
              // stored

    TC4SetNextUserAAPData(type, &input[0], &amplitude[0], fnblock.UserDefAmplitude, writeSize);

    size -= writesize;
  }
}
// *** END of Storing user defined points in FLASH  ***

// If there exist user defined points for current and/or power (fnblock.UserDefNumPoints > 0),
// they need to be stored according to the above mentioned schema as well.
```

// !! Important !!

```
// The above used functions
//
//  - TC4SetNextUserTimeData(...)
//  - TC4SetNextUserAAPData(...)
//
// for storing user defined points in FLASH may be called only immediately after calling

//
//   TC4SetFnSeqActionStore(..)
//
```

```
// for to store a function sequence. Once the points are stored, they can only be stored
// another time after having called TC4SetFnSeqActionStore(..)  again.


//
################################################################################################################
########
//
################################################################################################################
########

// For to get an overview about the function sequences that are stored in the TopCon device,
// use the following functions:
//
// - TC4SetFnSeqActionReadFirstHeader()
// - TC4SetFnSeqActionReadNextHeader()
//
// They read the headers of all function sequences existing in the power supply.


struct T_FnSeqHeader header;
int result;
bool first = true;

do
{
  if (first)
  {
    TC4SetFnSeqActionReadFirstHeader(&header);
    first = false;
  }
  else
  {
    TC4SetFnSeqActionReadNextHeader(&header);
  }

  // Wait until header has been read
  do
  {
    TC4GetFnSeqActionResult(&result);
    Sleep(500); // wait for 500ms (do not block other processes)
  }
  while (result > 0)

  if (result == 0)
  {
    // todo
    // 'header' is valid --> save it somewhere
  }
}
while (result >= 0) // repeat, as long as there are other header available and no error occurred.


//
################################################################################################################
########
//
################################################################################################################
########

// The execution status of the functions
//
// - TC4SetFnSeqActionLoad(unsigned int SeqNr)
// - TC4SetFnSeqActionStore(unsigned int SeqNr, int copyUserData)
// - TC4SetFnSeqActionDelete(unsigned int SeqNr)
// - TC4SetFnSeqActionReadFirstHeader()
// - TC4SetFnSeqActionReadNextHeader()
//
// can be determined by calling the function
//
// - TC4GetFnSeqActionResult(&result)
//
// The parameter 'result' return the following values:
//
// > 0: action not yet finished
// = 0: action finished successful
```

```
// < 0: error during execution:
//
//      -1: Timeout for flash access
//      -2: Flash not empty for writing
//      -3: The given function sequence does not exist (Action Load oder Delete)
//      -4: Flash-Page full, no space for further function sequences.
//      -5: Action does not exist.
//      -6: no further function sequences available (Action ReadFirstHeader or ReadNextHeader)
//

//#########################################################################################
//#########################################################################################

// Determine the state of the TFE:

unsigned int state; // 0: running
                    // 1: waiting for trigger
                    // 2: executing End-Action

TC4GetFnSeqState(&state);

//#########################################################################################
//#########################################################################################
```

4.5.2 TFE example - .h file

```
// Functions and definitions of the TopCon DLL
// to control the TopCon function generator
//
// Valid for TopCon-DLL V3.00.00 and higher
//
// Created: 2005-12-07/MM
//

struct T_FnSeqHeader {
    unsigned int  TotalSize;
    unsigned int  UserDefSize;
    unsigned int  Version;
    unsigned int  SeqNumber;
    char          SeqName[32];
    unsigned int  DateYear;
    unsigned int  DateMonth;
    unsigned int  DateDay;
    unsigned int  TimeHour;
    unsigned int  TimeMinute;
    unsigned int  TimeSecond;
    unsigned int  SysNomVoltage;
    unsigned int  SysMaxCurrent;
    unsigned int  SysNomPower;
};

struct T_FnSeq {
    unsigned int    StartTrigger;
    unsigned int    EndAction;
    double          Delay;
    unsigned int    Repeat;
    unsigned int    EnabledFnBlocks;
    int             GeneralEnable;
};

struct T_FnBlock {
    unsigned int    BaseFunction;
    double          Amplitude;
    double          Offset;
    double          Symmetry;
```

```
    double         Frequency;
    int            BipolarAmplitude;
    int            RectifyAmplitude;
    double         ExpTimeConstant;
    unsigned int   NumPeriods;
    unsigned int   UserDefAmplitude;
    unsigned int   UserDefTimePrescaler;
    double         UserDefPeriodLength;  /* Gesamt-Periodendauer */
    unsigned int   UserDefNumPoints;
    unsigned int   InactiveLevelType;
    double         InactiveLevel;
    unsigned int   AAPInputType;
    unsigned int   AAPInputFilterKoeff;
};


// functions to read/write header-data from/to Controllerboard-RAM
DllExport int TC4GetFnSeqHeader(struct T_FnSeqHeader *p_header);
DllExport int TC4SetFnSeqHeader(struct T_FnSeqHeader *p_header);

// functions to read/write function-sequence settings from/to Controllerboard-RAM
DllExport int TC4GetFnSeqSettings(struct T_FnSeq *p_fnseq);
DllExport int TC4SetFnSeqSettings(struct T_FnSeq *p_fnseq);

// functions to read/write function-block settings from/to Controllerboard-RAM
DllExport int TC4GetFnBlockSettings(struct T_FnBlock *p_fnblock, unsigned int type);
DllExport int TC4SetFnBlockSettings(struct T_FnBlock *p_fnblock, unsigned int type);

// function to set startaddress for reading/writing user-defined points from/to flash
DllExport int TC4SetFnBlockUserDataStartAddr(unsigned int type);

// functions to read/write user defined data (time based) from/to flash
DllExport int TC4GetNextUserTimeData(unsigned int type, unsigned int *pTimeDelta, int *pAmplitude, unsigned int
maxAmplitude, unsigned int *readSize);
DllExport int TC4SetNextUserTimeData(unsigned int type, unsigned int *pTimeDelta, int *pAmplitude, unsigned int
maxAmplitude, unsigned int writeSize);

// functions to read/write user defined AAP data from/to flash
DllExport int TC4GetNextUserAAPData(unsigned int type, int *pInput, int *pAmplitude, unsigned int maxAmplitude, unsigned int
*readSize);
DllExport int TC4SetNextUserAAPData(unsigned int type, int *pInput, int *pAmplitude, unsigned int maxAmplitude, unsigned int
writeSize);

// functions to request an internal action
DllExport int TC4SetFnSeqActionLoad(unsigned int SeqNr);
DllExport int TC4SetFnSeqActionStore(unsigned int SeqNr, int copyUserData);
DllExport int TC4SetFnSeqActionDelete(unsigned int SeqNr);
DllExport int TC4SetFnSeqActionReadFirstHeader();
DllExport int TC4SetFnSeqActionReadNextHeader();

// function to read status of internal action
DllExport int TC4GetFnSeqActionResult(int *p_result);

// function to read actual function-sequence state
DllExport int TC4GetFnSeqState(unsigned int *p_state);

// function to manually trigger start of function-sequence
DllExport int TC4SetFnSeqManualTrigger();
```

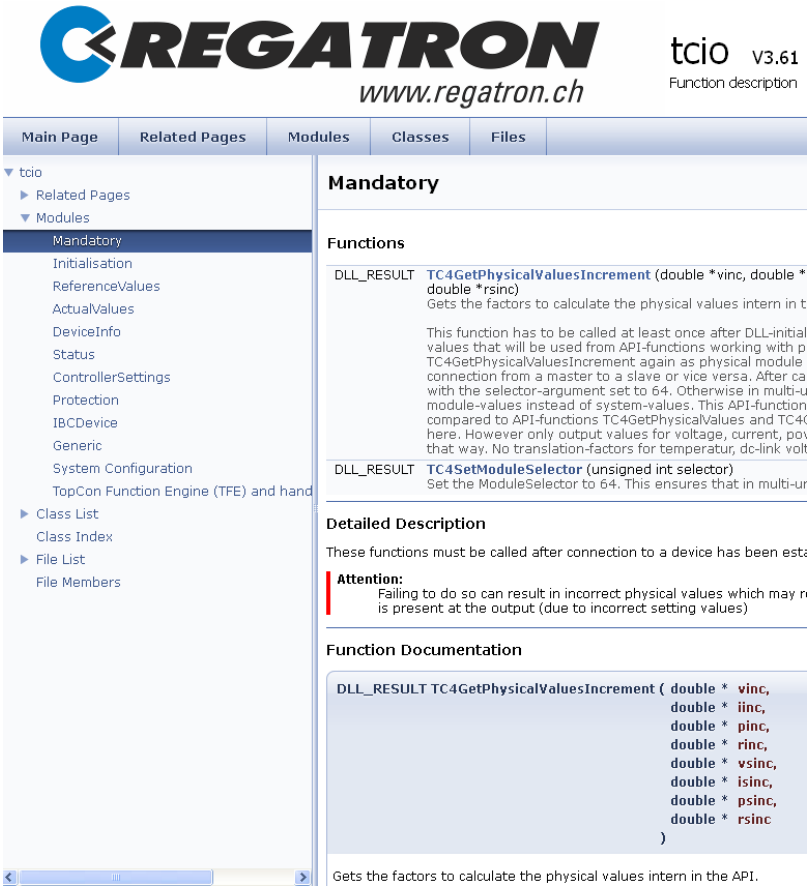
---------- end of document ----------

5 Detail function description in html file

All functions without DLL at the beginning of the function-name has been described in this document.

Each function is assigned to a module.

E.g. if you open a module it includes a list of all functions in this module. The “Detail Description” informs in general about this module. After is it followed by the Function Description in detail.



In the slide files you got on the click of „serialiolib.cpp“ the whole list of all functions in the tcio.dll.

Attention:
If there is written “See also: chapter Translate between standardised values and physical values please refer chapter 2.2.2