g.tec medical engineering GmbH
Sierningstrasse 14, 4521 Schiedlberg, Austria
Tel.: (43)-7251-22240-0
Fax: (43)-7251-22240-39
office@gtec.at, http://www.gtec.at

g.USBamp C API for Linux V1.11.00

How to contact g.tec

☎	++43-7251-22240-0	**Phone**
	++43-7251-22240-39	**Fax**
	g.tec medical engineering GmbH Sierningstrasse 14, 4521 Schiedlberg, Austria	**Mail**
	http://www.gtec.at	**Web**
@	office@gtec.at	**e-mail**
	AT/CA01/RC000989-00	**ÖBIG Reg. number**

Before using g.USBamp

Before using the device make yourself familiar with the *gUSBampInstructionsForUse.pdf* manual and carefully read following sections

- The intended function of the equipment
- Safe operation of g.USBamp

Table of contents

About this manual

This manual introduces the functionality of the libraries

- libgusbampapia.so.1.11.00 and
- libgusbampapiso.so.1.11.00.so .

The difference between these two libraries is the way in which they have linked their dependencies. The dependencies of libgusbampapia.so.1.11.00are linked statically. The dependencies of libgusbampapiso.so.1.11.00 are linked dynamically. The functionality of these two libraries is the same. The notation

- libgusbampapix.so.1.11.00

refers to them both and “x” means “a” or “so”.
libgusbampapiso.so.1.11.00 relies to more dependencies. libgusbampapia.so.1.11.00 has a clean symbol table and does not affect the usage of other libraries within your program.
Note: On Ubuntu Linux 11.04 only libgusbampapiso.so.1.11.00 is available.

Hardware and software requirements

g.USBamp requires a PC compatible desktop, notebook workstation or embedded computer. The table below lists optimal settings:

Hardware	**Properties**
CPU	Pentium working at 2000 MHz
Hard disk	20-30 GB
RAM	1GB – 2 GB
USB 2.0 port (EHCI – enhanced Host controller interface)	one free USB port for each g.USBamp

The g.USBamp software package is tested on Ubuntu Linux.

Tested Operating System	**Kernel**
64bit/32bit Ubuntu Linux 11.04	2.6.38-11-generic
Dependencies/Libraries:	
librt.so.1 libstdc++.so.6 libm.so.6 libgcc_s.so.1 libpthread.so.0 libc.so.6 To retrieve the dependencies of the delivered shared object use “ldd”. For more information please see “man ldd”.	
Additional dependencies/libraries for libgusbampapiso.so.1.11.00:	
libusb-1.0.so.0 libicudata.so.44 libicui18n.so.44 libicuuc.so.44 libboost_regex.so.1.42.0 libboost_date_time.so.1.42.0 libboost_signals.so.1.42.0 libboost_thread.so.1.42.0	

Make sure that your operating system works correctly before installing the g.USBamp library.
During operation of g.USBamp other major software MUST NOT be operated.

Installation

Installation of the g.USBamp C API for Linux

1. Please insert the g.USBamp C API for Linux CD into the CD-drive of your PC.
2. Use a terminal in order to change to the directory
   - cd gUSBampAPI_1_11_00
     and run the install script
   - sh install.sh

Note: Please make sure that you have the permissions to operate on the directories and to execute ldconfig correctly.

De-installation

1. Remove the header file from your computer
   - rm /usr/include/gAPI.h
2. Remove the library from your computer
   - rm /usr/include/libgusbampapix.1.11.00
   - call ldconfig
3. Remove the documentation from your computer
   - rm -rf /usr/share/doc/gtec/gUSBampAPI_1_11_00/

Note: Please make sure that you have the permissions to operate on the directories and to execute ldconfig correctly.

Files on your computer

The library and the include file
- /usr/lib/libgusbampapix.so.1.11.00
- /usr/include/gAPI.h

Filter files on your computer
- /etc/gtec/filter_files/DSPfilter.bin
- /etc/gtec/filter_files/DSPNotchfilter.bin

Further files on your computer
- /usr/share/doc/gtec/gUSBampAPI_1_11_00/
  gUSBamp_C_API_for_Linux_1_11_00.pdf
- /usr/share/doc/gtec/gUSBampAPI_1_11_00/license.pdf

Principles of usage

Find, open and close a device
In order to find devices which are enabled for the API you can use the functions

- GT_UpdateDevices (this function updates the API internal list of available devices),
- GT_GetDeviceListSize (this function returns the number of found devices),
- GT_GetDeviceList (this function returns a list of device names) and
- GT_FreeDeviceList (this function frees the memory allocated by GT_GetDeviceList).

The device name is also identifier of the device.
If you have selected a device of choice you can open it. If you are finished with the device you must close it. Therefore you can use the functions

- GT_OpenDevice and
- GT_CloseDevice.

About the configuration of a device
Before you can start the data acquisition you must configure the device. You have to set all parameters/members in/of the structure

- usbamp_config.

The configuration defined within the structure can be applied to the device by using the function

- GT_SetConfiguration.

The applied configuration can also be retrieved from the API by using the function

- GT_GetConfiguration.

The configuration cannot be changed during data acquisition.
The things you can change during data acquisition are defined within the structure

- usbamp_asynchron_config.

You must configure all members of usbamp_asynchron_config.
Applying the asynchronous configuration is done in two steps:

1. You must set the asynchronous configuration with the function
   GT_SetAsynchronConfiguration
   and then you can apply it to the device with
2. GT_ApplyAsynchronConfiguration.

The function GT_ApplyAsynchronConfiguration can be called before and also during data acquisition.

About the callback function
Each time new data samples are ready to be processed the API invokes a callback function. The callback function can be set with

- GT_SetDataReadyCallback.

Within your own callback function use

- GT_GetSamplesAvailable to get the number of available bytes and
- GT_GetData in order to copy the available number of bytes to a buffer specified by you.

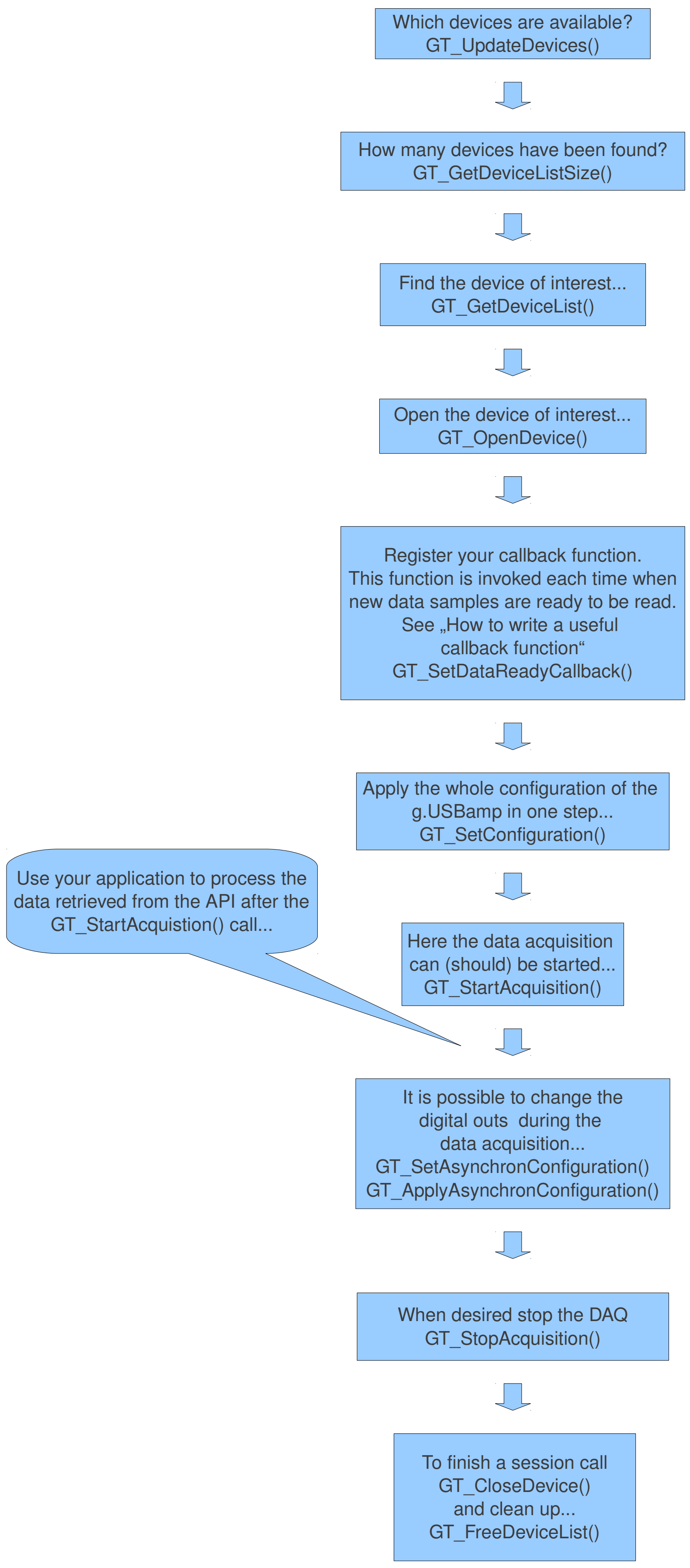
You can set your callback function directly after GT_OpenDevice. You cannot change the callback function during data acquisition.

How to start and stop the data acquisition
Now you can start or stop the data acquisition with the functions

- GT_StartAcquisition and
- GT_StopAcquisition.

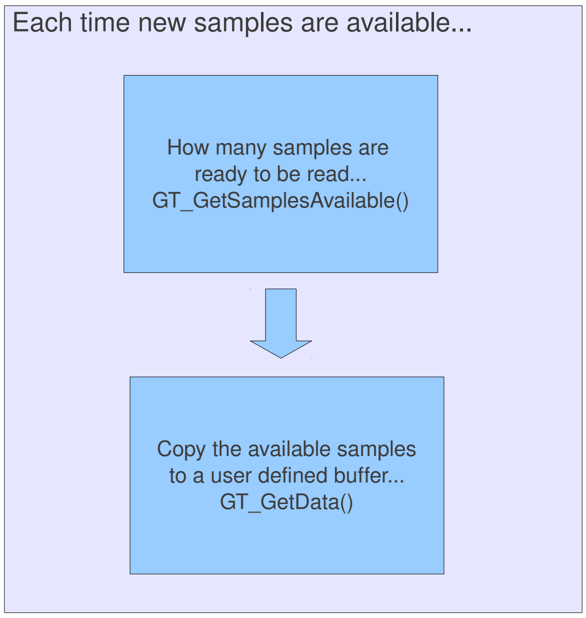
Flow diagram of a typical data acquisition

Please see the source of the demo applications shipped with the API for a more detailed view on this process.


Which devices are available?
GT_UpdateDevices()
How many devices have been found?
GT_GetDeviceListSize()
Find the device of interest...
GT_GetDeviceList()
Open the device of interest...
GT_OpenDevice()
Register your callback function.
This function is invoked each time when new data samples are ready to be read.
See „How to write a useful callback function“
GT_SetDataReadyCallback()
Apply the whole configuration of the g.USBamp in one step...
GT_SetConfiguration()
Use your application to process the data retrieved from the API after the GT_StartAcquistion() call...
Here the data acquisition can (should) be started...
GT_StartAcquisition()
It is possible to change the digital outs during the data acquisition...
GT_SetAsynchronConfiguration()
GT_ApplyAsynchronConfiguration()
When desired stop the DAQ
GT_StopAcquisition()
To finish a session call
GT_CloseDevice()
and clean up...
GT_FreeDeviceList()

How to write a useful callback function

The callback function can be used to retrieve the data samples from the API (without writing a thread). A separate thread is running in the background to acquire the data from the g.USBamp asynchronous to your application. The callback function blocks the internal data acquisition thread. It is important to process the callback function as fast as possible in order to avoid a loss of data samples.


Each time new samples are available...
How many samples are ready to be read... GT_GetSamplesAvailable()
Copy the available samples to a user defined buffer... GT_GetData()

g.tec g.USBamp C API overview Functions, types and structures

Common g.tec API functions

Function name	
	GT_ShowDebugInformation
Signature	*void GT_ShowDebugInformation(gt_bool show)*
Parameters	*gt_bool show*
Return value	*void*
Effects	If show is *true*, debug informations are printed to the console.
Preconditions	None
Postconditions	
Note	Could be useful to find wrong settings of the configuration. If show is *true*, API internal methods are enabled to print informations on the console, else these methods are disabled.
Release note	

Function name	
	GT_UpdateDevices
Signature	*gt_bool GT_UpdateDevices()*
Parameters	None
Return value	*true* if no error occurs, else *false*.
Effects	Searches for devices which are enabled for the API.
Preconditions	None
Postconditions	Could effect the return values of *gt_size GT_GetDeviceListSize()* and *char** GT_GetDeviceList()*.
Note	
Release note	

Function name	
	GT_GetDeviceListSize
Signature	*gt_size GT_GetDeviceListSize()*
Parameters	None
Return value	The number of found devices.
Effects	
Preconditions	*gt_bool GT_UpdateDevices()* has to be called before.
Postconditions	

Note	
Release note	

Function name

GT_GetDeviceList

Signature	*char** GT_GetDeviceList()*
Parameters	None
Return value	An array of char strings which represent the device identifier (for example the serial number).
Effects	
Preconditions	*gt_bool GT_UpdateDevices()* has to be called before.
Postconditions	Memory is allocated for the array of char strings.
Note	The handling of the list is within the responsibility of the user, the list can be freed with *gt_bool GT_FreeDeviceList(char** device_list, gt_size list_size)*.

Function name

GT_FreeDeviceList

Signature	*gt_bool GT_FreeDeviceList(char** device_list, gt_size list_size)*
Parameters	The pointer to the array *device_list* allocated by *char** GT_GetDeviceList()* and the length of the array given by *gt_size GT_GetDeviceListSize()* .
Return value	*true* on success.
Effects	
Preconditions	Memory for the *device_list* must have been allocated with *char** GT_GetDeviceList()* before.
Postconditions	Memory is freed and *device_list* is a pointer to *NULL*.
Note	

Function name

GT_OpenDevice

Signature	*gt_bool GT_OpenDevice(const char* device_name)*
Parameters	The name of the device.
Return value	*true* if the device can be opened.
Effects	Enables the interaction with the device.
Preconditions	
Postconditions	
Note	
Release note	

Function name

GT_CloseDevice

Signature	*gt_bool GT_CloseDevice(const char* device_name)*
Parameters	The name of the device.
Return value	*true* on success. *false* else.
Effects	
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before. The data acquisition must not be active.
Postconditions	
Note	
Release note	

Function name	**GT_SetConfiguration**
Signature	*gt_bool GT_SetConfiguration(const char* device_name, void* configuration)*
Parameters	The name of the device. The configuration for the device.
Return value	*true* when the configuration is valid and if this configuration can be applied to the device.
Effects	Sets the configuration for a given device.
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before. The data acquisition must not be active.
Postconditions	If successful, the given configuration is applied to the device.
Note	
Release note	

Function name	**GT_GetConfiguration**
Signature	*gt_bool GT_GetConfiguration(const char* device_name, void* configuration)*
Parameters	The name of the device. The configuration structure for the device.
Return value	*true* if a device's configuration was stored to *configuration*. *false* else.
Effects	Retrieves the current applied configuration of the devices and stores it in *configuration*.
Preconditions	*gt_bool GT_SetConfiguration(const char* device_name, void* configuration)* has to be called before.
Postconditions	
Note	*void* configuration* is changed by reference.
Release note	

Function name	**GT_SetAsynchronConfiguration**
Signature	*gt_bool GT_SetAsynchronConfiguration(const char* device_name, void* configuration)*
Parameters	The name of the device. The configuration for the device.
Return value	*true* if the configuration is valid.
Effects	The given configuration can be applied to the device.
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	*gt_bool GT_ApplyAsynchronConfiguration(const char* device_name)* can be called.
Note	This method can also be called during data acquisition.
Release note	

Function name	**GT_ApplyAsynchronConfiguration**
Signature	*gt_bool GT_ApplyAsynchronConfiguration(const char* device_name)*
Parameters	The name of the device.
Return value	*true* if the configuration was successful applied.
Effects	
Preconditions	*gt_bool GT_SetAsynchronConfiguration(const char* device_name, void* configuration)* has to be called before.
Postconditions	
Note	This method can also be called during data acquisition.
Release note	

Function name	**GT_GetAsynchronConfiguration**
Signature	*gt_bool GT_GetAsynchronConfiguration(const char* device_name, void* configuration)*
Parameters	The name of the device. The asynchron configuration structure for the device.
Return value	*true* if a device's asynchron configuration was stored to configuration.
Effects	Retrieves the current asynchron configuration of the device and stores it in *configuration*.
Preconditions	*gt_bool GT_ApplyAsynchronConfiguration(const char* device_name)* has to be called before.
Postconditions	
Note	*void* configuration* is changed by reference.
Release note	

Function name	
	GT_StartAcquisition
Signature	*gt_bool GT_StartAcquisition(const char* device_name)*
Parameters	The name of the device.
Return value	*true* if the acquisition started successful.
Effects	The device starts to stream data.
Preconditions	*gt_bool GT_SetConfiguration(const char* device_name, void* configuration)* has to be called before.
Postconditions	*int GT_GetSamplesAvailable(const char* device_name)* can be used to check the number of valid bytes. *int GT_GetData(const char* device_name, unsigned char* buffer, gt_size num_bytes)* can be used to copy the data to a *buffer* specified by the user. If a callback function has been registered (with *gt_bool GT_SetDataReadyCallBack(const char* device_name, void (*callback_function)(void)),* void* usr_data), this function will be called each time when new samples are ready to be read.
Note	
Release note	

Function name	
	GT_StopAcquisition
Signature	*gt_bool GT_StopAcquisition(const char* device_name)*
Parameters	The name of the device.
Return value	*true* if the device has stopped.
Effects	The device stops to stream data.
Preconditions	*gt_bool GT_StartAcquisition(const char* device_name)* has to be called before.
Postconditions	
Note	
Release note	

Function name	
	GT_GetSamplesAvailable
Signature	*int GT_GetSamplesAvailable(const char* device_name)*
Parameters	The name of the device.
Return value	The number of bytes ready to be read. A negative number if an error occured.
Effects	
Preconditions	*gt_bool GT_StartAcquisition(const char* device_name)* has to be called before.
Postconditions	

Note	
Release note	

Function name

GT_GetData

Signature	*int GT_GetData(const char* device_name, unsigned char* buffer, gt_size num_bytes)*
Parameters	The name of the device. A buffer. The amount of samples which should be copied to the *buffer*.
Return value	The number of copied samples. A negative number if samples have been dropped (in the API internal buffer) .
Effects	
Preconditions	*gt_bool GT_StartAcquisition(const char* device_name)* has to be called before.
Postconditions	
Note	
Release note	

Function name	
	GT_SetDataReadyCallBack
Signature	*gt_bool GT_SetDataReadyCallBack(const char* device_name, void (*callback_function)(void), void* usr_data)*
Parameters	The name of the device. A callback function: *void FunctionName(void* data).* User data to pass to callback function: *void* usr_data.*
Return value	*true* if the callback function is registered.
Effects	Each time new samples are ready to be read the callback function is called.
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	
Note	A callback function could be very useful in combination with *int GT_GetSamplesAvailable(const char* device_name)* and *int GT_GetData(const char* device_name, unsigned char* buffer, gt_size num_bytes)*.
Release note	

Common g.tec API data types and structures

Name	**filter_specification**	
type	struct	
typedef	gt_filter_specification	
See also	gt_bool GT_GetBandpassFilterList(const char* device_name, gt_size sample_rate, gt_filter_specification* filter, gt_size filter_size) gt_bool GT_GetNotchFilterList(const char* device_name, gt_size sample_rate, gt_filter_specification* filter, gt_size filter_size)	
Data fields / members	Possible values	Notes
float f_upper		
float f_lower		
float sample_rate		
float order		
float type		
gt_size id		

Name	**gt_size**
type	Unsigned long int
typedef	gt_size
See also	

Name	**gt_bool**
type	Unsigned char
typedef	gt_bool
See also	GT_TRUE GT_FALSE

Name	**GT_TRUE**
type	#define
value	1
See also	gt_bool

Name	**GT_FALSE**

type	#define
value	0
See also	gt_bool

Name	**GT_NOS_AUTOSET**
type	#define
value	-1
Note	NOS = Number Of Scans

Name	**GT_BIPOLAR_DERIVATION_NONE**
type	#define
value	-2

Name	**GT_FILTER_AUTOSET**
type	#define
value	-1
note	Applies the first filter which fits to the given sample rate.

Name	**GT_FILTER_NONE**
type	#define
value	-2

g.USBamp specific API functions

Function name	**GT_GetBandpassFilterListSize**
Signature	*gt_size GT_GetBandpassFilterListSize(const char* device_name, gt_size sample_rate)*
Parameters	The name of the device. The sample rate of interest.
Return value	The number of found filters to the according sample rate.
Effects	
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	
Note	The method for the notch filters work similar.
Release note	

Function name	**GT_GetBandpassFilterList**
Signature	*gt_bool GT_GetBandpassFilterList(const char* device_name, gt_size sample_rate, gt_filter_specification* filter, gt_size filter_size)*
Parameters	The name of the device. The sample rate of interest. The start pointer to the list of *gt_filter_specification*.
Return value	true if filters are found.
Effects	
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	
Note	Memory for the *gt_filter_specification* must be allocated by the user. The method for the notch filters work similar.
Release note	

Function name	**GT_Calibrate**
Signature	*gt_bool GT_Calibrate(const char* device_name, gt_usbamp_channel_calibration* calibration)*
Parameters	The name of the device. A pointer to *gt_usbamp_channel_calibration.*
Return value	true on success.
Effects	Determines offset and scaling for each channel. Memory for the *gt_usbamp_channel_calibration* must be allocated by the user.

Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before. *gt_bool GT_StartAcquisition(const char* device_name)* must not be called before.
Postconditions	
Note	
Release note	

Function name	**GT_SetChannelCalibration**
Signature	*gt_bool GT_SetChannelCalibration(const char* device_name, gt_usbamp_channel_calibration* calibration)*
Parameters	The name of the device. A pointer to the *gt_usbamp_channel_calibration.*
Return value	true on success.
Effects	The channel calibration field in the permanent memory of the device is set to the values specified by *gt_usbamp_channel_calibration.* **Calibration values are changed permanent.** Calculation: y = (x – d) * k y … values retrieved with GT_GetData (calculated values) in µV x … acquired data d … offset value in µV k … scale factor
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	
Note	Memory for the *gt_usbamp_channel_calibration* must be allocated by the user. The function call is blocking for a few seconds. Use *gt_bool GT_Calibrate(const char* device_name, gt_usbamp_channel_calibration* calibration)* in order to retrieve the necessary values from the device. The device is already calibrated by the manufacturer. There is no need to call this function.
Release note	

Function name	**GT_GetChannelCalibration**
Signature	*gt_bool GT_GetChannelCalibration(const char* device_name, gt_usbamp_channel_calibration* calibration)*
Parameters	The name of the device. A pointer to the *gt_usbamp_channel_calibration*
Return value	true on success.
Effects	Memory for the *gt_usbamp_channel_calibration* must be allocated by the user. The calibration settings of the device are stored at *gt_usbamp_channel_calibration* calibration*.

Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before.
Postconditions	
Note	
Release note	

Function name	**GT_GetImpedance**
Signature	*gt_bool GT_GetImpedance(const char* device_name, gt_size channel, int* impedance)*
Parameters	The name of the device. The channel of interrest. A pointer to an integer.
Return value	true on success.
Effects	Retrieves the Resistance according to the given channel.
Preconditions	*gt_bool GT_OpenDevice(const char* device_name)* has to be called before. *gt_bool GT_StartAcquisition(const char* device_name)* must not be called before.
Postconditions	
Note	All grounds are connected to common ground. Impedance is measured between ground and specified channel. If you want to get the impedance of the reference electrodes following channel numbers must be entered: 1 … 16 for the channel 1 … 16 17 REFA 18 REFB 19 REFC 20 REFD This function is blocking for a view seconds.
Release note	

g.USBamp specific data types and structures

Name	**GT_USBAMP_NUM_DIGITAL_OUT**
type	#define
value	4

Name	**GT_USBAMP_NUM_REFERENCE**
type	#define
value	4

Name	**GT_USBAMP_RECOMMENDED_BUFFER_SIZE**
type	#define
value	

Name	**GT_USBAMP_NUM_GROUND**
type	#define
value	4

Name	**GT_USBAMP_NUM_ANALOG_IN**
type	#define
value	4

Name	**usbamp_device_mode**
type	enum
typedef	gt_usbamp_device_mode
enumerations	GT_MODE_NORMAL, GT_MODE_IMPEDANCE, GT_MODE_CALIBRATE, GT_MODE_COUNTER

Name	**usbamp_analog_out_shape**
type	enum
typedef	gt_usbamp_analog_out_shape
enumerations	GT_ANALOGOUT_SQUARE, GT_ANALOGOUT_SAWTOOTH, GT_ANALOGOUT_SINE, GT_ANALOGOUT_NOISE

Name	**usbamp_channel_calibration**
type	struct
typedef	gt_usbamp_channel_calibration
See also	gt_bool GT_Calibrate(const char* device_name, gt_usbamp_channel_calibration* calibration) gt_bool GT_GetChannelCalibration(const char* device_name, gt_usbamp_channel_calibration* calibration) gt_bool GT_SetChannelCalibration(const char* device_name, gt_usbamp_channel_calibration* calibration)

Data fields / members	Possible values	Notes
float scale[GT_USBAMP_NUM_ANALOG_IN] float offset[GT_USBAMP_NUM_ANALOG_IN]		The scale factor. Offset in µV.

Name	**usbamp_config**	
type	struct	
typedef	gt_usbamp_config	
See also	gt_bool GT_SetConfiguration(const char* device_name, void* configuration) gt_bool GT_GetConfiguration(const char* device_name, void* configuration) gt_usbamp_analog_out_config	
Data fields / members	Possible values	Notes
unsigned short int sample_rate	32, 64, 128, 256, 512, 600, 1200, 2400, 4800, 9600, 19200, 38400	
int number_of_scans	GT_NOS_AUTOSET	It is strongly recommended to use the GT_NOS_AUTOSET define.
gt_bool enable_trigger_line	GT_TRUE GT_FALSE	Enables the trigger line. Disables the trigger line.
gt_bool slave_mode	GT_TRUE GT_FALSE	Enables slave mode. Disables slave mode. 2 amplifiers are supported.
gt_bool enable_sc	GT_TRUE GT_FALSE	Enables the short cut. Disables the short cut.
gt_bool common_ground[GT_USBAMP_NUM_GROUND]	GT_TRUE GT_FALSE	Set common ground on group. Unset common ground on group.
gt_bool common_reference[GT_USBAMP_NUM_REFERENCE]	GT_TRUE GT_FALSE	Set common reference on group. Unset common reference on group.
gt_usbamp_device_mode mode	GT_MODE_NORMAL GT_MODE_IMPEDANCE GT_MODE_CALIBRATE	Mode for measurement Untested yet. In this mode the internal signal generator can be observed on all channels. Sample rate

	GT_MODE_COUNTER	must be <= 512. The wave shape of the signal is specified with the member ao_config. The counter can be observed on channel 16
gt_bool scan_dio	GT_TRUE GT_FALSE	The digital in appears as channel 17
float version		
int bandpass[GT_USBAMP_NUM_ANALOG_IN]	The id of the bandpass	The selected filter must match the sample rate.
int notch[GT_USBAMP_NUM_ANALOG_IN]	The id of the notch	The selected filter must match the sample rate.
int bipolar[GT_USBAMP_NUM_ANALOG_IN]	The channels for the bipolar derivation	Range from [1 ... 16]
unsigned char analog_in_channel[GT_USBAMP_NUM_ANALOG_IN]	The channels for the data acquisition	Range from [1 ... 16] Specifies the channel to be observed.
gt_size num_analog_in	The number of selected analog channels	Specifies how many channels to be observed.
gt_usbamp_analog_out_config* ao_config	gt_usbamp_analog_out_config*	Is used to define the properties of the analog out signal. See also member mode.

Name	**usbamp_analog_out_config**	
type	struct	
typedef	gt_usbamp_analog_out_config	
See also	gt_usbamp_analog_out_config	
Data fields / members	Possible values	Notes
gt_usbamp_analog_out_shape shape	GT_ANALOGOUT_SQUARE GT_ANALOGOUT_SAWTOOTH GT_ANALOGOUT_SINE GT_ANALOGOUT_NOISE	
short int offset	[-200 ... 200]	Offset in mV
short int frequency	[1 ... 100]	Frequency in Hz
short int amplitude	[-250 ... 250]	Amplitude in mV

Name	**usbamp_asynchron_config**

type	struct	
typedef	gt_usbamp_asynchron_config	
See also	gt_bool GT_SetAsynchronConfiguration(const char* device_name, void* configuration) gt_bool GT_ApplyAsynchronConfiguration(const char* device_name) gt_bool GT_GetAsynchronConfiguration(const char* device_name, void* configuration)	
Data fields / members	Possible values	Notes
gt_bool digital_out[GT_USBAMP_NUM_DIGITAL_OUT]	GT_TRUE GT_FALSE	Enables the digital out. Disables the digital out.

g·tec

GUGER
TECHNOLOGIES

contact information

g.tec medical engineering GmbH
Sierningstrasse 14
4521 Schiedlberg
Austria

tel. +43 7251 22240
fax. +43 7251 22240 39
web: www.gtec.at
e-mail: office@gtec.at