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
USB BIOSIGNAL AMPLIFIER

g.USBamp C API V3.10.01

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**

Content

RELEASE NOTES 4

NEW FEATURES 4
CHANGES 4
VERSION COMPATIBILITY CONSIDERATIONS 5
RELATED DOCUMENTS 5

BEFORE USING G.USBAMP 6

REQUIREMENTS AND INSTALLATION 7

HARDWARE AND SOFTWARE REQUIREMENTS 7
INSTALLATION FROM A CD 8

SYNCHRONIZATION OF MULTIPLE G.USBAMPS: THE GUSBAMPSYNCDEMO PROJECT 10

CONNECTING TWO OR MORE G.USBAMP DEVICES 10
MODIFYING THE SOURCE CODE PARAMETERS 10
COMPILING THE SOURCE CODE 12
EXECUTING THE PROJECT (GUSBAMPSYNCDEMO.EXE) 12
OPENING THE RECEIVEDDATA.BIN FILE 12

DATA STORAGE 14

APPLICATION PROGRAM INTERFACE (API) 15

FUNCTION REFERENCE 16

DATA ACQUISITION 18
DIGITAL I/O 28
FILTER 33
MODE 39
CONNECT GROUND AND REFERENCE 41
BIPOLAR DERIVATION 45
SYNCHRONIZATION 48
CALIBRATION/DRL 49
ERROR HANDLING 53
GENERAL 54

Release Notes

Release notes help to learn about new features and changes of g.USBamp C API and tools when upgrading to a newer version of the software.

New features

Version 3.10.01 of g.USBamp C API also includes device drivers for Windows XP operating systems.

Version 3.10.00 of g.USBamp C API gives the user access to the new features and improvements implemented in g.USBamp version 3.0.
g.USBamp version 3.0 provides 8 digital inputs that are scanned synchronous with the analog channels. In the previous release of the amplifier only one synchronous digital input was available (trigger line).

There a some new driver calls to access to new functionality of g.USBamp version 3.0

- GT_GetHWVersion(…)
- GT_SetDigitalOutEx(…)
- GT_GetDigitalOut(…)

There is a new structure used for the digital outputs _DigitalOUT

Changes

The behavior of the asynchronous digital I/Os changed. The total number of digital outputs is 4 and there are no asynchronous digital inputs any more.

The signaling for the synchronization of multiple amplifiers has changed to LVDS technology for improved stability in noisy environments.

The counter available on channel 16 overruns at 1,000,000 now.

The decimation filter used for down sampling the 38 kHz input signal to the desired output sampling frequency has changed its type. The filter uses an IIR filter (butterworth) to increase the out of band signal attenuation (anti aliasing). In version 2.0 of the amplifier a simple FIR averaging filter was used.

The test application *g.USBamp Demo* and the C-API changed to reflect the hardware changes.

The serial number changed from **UA**-xxxx.xx.xx to **UB**-xxxx.xx.xx

Version Compatibility Considerations

For users of g.USBamp version 2.0 (devices with serial UA-xxxx.xx.xx)

- g.USBamp C API 3.10.01 provides the same functionality as version 2.07a
  it can drive all previous g.USBamp versions
- you can use this C API only with g.USBamp driver 3.10.01
- you cannot synchronize amplifiers of version 2.0 and 3.0 except with a special interconnection box

For users of g.USBamp 3.0 (devices with serial UB-xxxx.xx.xx)

- you cannot access g.USBamp 3.0 with g.USBamp C API 2.07a
- you cannot synchronize amplifiers of version 2.0 and 3.0 except with a special interconnection box

Note: only devices of the same version can be synchronized except with the g.INTERsync box.

Related documents

gUSBampInstructionsForUse.pdf – a detailed hardware description of the device (sockets, labeling …)

gUSBampDriver.pdf – a detailed description of the driver installation and the start up of the device

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

Requirements and Installation

Hardware and Software Requirements

g.USBamp requires a PC compatible desktop, notebook workstation or embedded computer running Microsoft Windows.

The table below lists optimal settings:

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

The g.USBamp C API software package requires a Microsoft Windows operating system. For the usage of the C++ application program interface (API) Microsoft Visual Studio 2005 is necessary

Software	**Version**
Windows	Windows 7 Professional English Win32 or Win64
Microsoft Visual Studio	2005
Acrobat Reader	9.3.2
g.USBamp driver	3.10.01

Make sure that your Microsoft Windows installation works correctly before installing the g.USBamp software. Other software packages except the packages listed above MUST NOT be installed on the Windows PC. During operation of g.USBamp other software as listed above MUST NOT be operated.

Installation from a CD

Insert the g.tec product CD into your CD drive and direct to the folder `g.USBamp\g.USBamp C API`. Start

`Setup.exe`

Follow the instructions on the screen to install the g.USBamp C API.

Files on your Computer

g.USBamp application files – program files are stored under

`Your selected path \ gUSBampCAPI`

Documentation – documentation files are stored under

`Your selected path \ gUSBampCAPI \ Doc`

Application Program Interface (API) – API files are stored under

`Your selected path \ gUSBampCAPI \ API`

Driver files – driver files are stored under

`Your selected path \ gUSBampCAPI \ Driver`

Source files – the sources for the demo applications

`Your selected path \ gUSBampCAPI \ C++ Projects`

Note: If you want to build the project from this location you need write access to this folder

Deployment

If you want to distribute your own application for g.USBamp the most common way is to use the g.USBamp driver installation files (msi).
If you want to replace the g.USBamp driver 3.10.01 installation routine please copy the following files to the corresponding paths (default Windows installation assumed):

The driver files `DSPfilter.bin`, `DSPNotchfilter.bin`, `DSPfilter38k.bin`, `DSPNotchfilter38k.bin` and `gUSBamp.dll` must be in

`C:\Windows\System32`

The driver file `gusb.sys` must be in

`C:\Windows\System32\Drivers`

The installation file `gusb.inf` must be in

`C:\Windows\inf`

Synchronization of multiple g.USBamps: the gUSBampSyncDemo project

A Visual Studio 2005 C++ demo project comes with your g.USBamp C-API installation which is intended to demonstrate the usage of the g.USBamp C-API for one or more synchronized g.USBamp amplifiers. The Visual Studio 2005 solution file `gUSBampSyncDemo.sln` should be located under the following path:

```
C:\Program Files\gtec\gUSBampCAPI\C++ Projects\gUSBampSyncDemo\
```

On execution the demo project will open the specified g.USBamp devices, initialize them and record data for a specific amount of seconds to a binary file on the harddisk. Before you compile and execute the demo project please read the following section to configure it properly.

Connecting two or more g.USBamp devices

A g.USBamp 2.0 device can be connected with a g.USBamp 3.0 device through the g.INTERsync box. 4 g.USBamp devices or less of the same version can be connected through synchronization cables or a g.SYNCbox. The SYNC OUT of the master device have to be connected with the SYNC IN of the slave device(s). Ensure that all devices are switched on.

Modifying the source code parameters

The `gUSBampSyncDemo.cpp` file all the communication with the g.USBamp devices is done. At the beginning of the file you'll find a section commented with "//main program constants" and another one commented with "//device configuration settings" below where several global configuration parameters can be adjusted:

```
//main program constants
const int BUFFER_SIZE_SECONDS = 5;
const long NUM_SECONDS_RUNNING = 10;
const int QUEUE_SIZE = 4;

//device configuration settings
LPSTR _masterSerial = "UA-2006.00.00";
LPSTR _slaveSerials[] = { "UB-2010.03.43", "UB-2010.03.44", "UB-2010.03.47" };
const int SLAVE_SERIALS_SIZE = 3;
const int SAMPLE_RATE_HZ = 256;
const int NUMBER_OF_SCANS = 8;
const UCHAR NUMBER_OF_CHANNELS = 16;
UCHAR _channelsToAcquire[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16};
UCHAR _mode = M_NORMAL;
CHANNEL _bipolarSettings;
REF _commonReference = { FALSE, FALSE, FALSE, FALSE };
GND _commonGround = { FALSE, FALSE, FALSE, FALSE };
```

BUFFER_SIZE_SECONDS

The demo project executes two separate threads: one continuously receives data from the device and stores it into an internal cyclic buffer. The other one continuously reads data from that cyclic buffer and writes it to the harddisk. This ensures that data acquisition is independend from data processing avoiding possible loss of data when processing is too time consuming.

The **BUFFER_SIZE_SECONDS** constant represents the length of the internal cyclic buffer in seconds and is set to 5 second in this example.

NUM_SECONDS_RUNNING

The number of seconds that data will be recorded from the device.

QUEUE_SIZE

To avoid loss of data a number of `GT_GetData` calls will be queued at the beginning of data acquisition. This number is specified by the **QUEUE_SIZE** parameter. A value of 4 or 8 should fit the needs.

_masterSerial

Specify the serial number of the master device here. Only one device can be the master.

_slaveSerials[]

This array contains the serial numbers of all connected slave devices. They don't have to be in any order. It is very important that you set the SLAVE_SERIAL_SIZE parameter to the number of the contained slave devices in this array.

SLAVE_SERIALS_SIZE

This value specifies the number of contained slave device serial numbers in the **_slaveSerials[]** array. This value mustn't be bigger than the number of elements in the **_slaveSerials[]** array otherwise the application will crash or have undefined behaviour. If this value is less than the elements in the **_slaveSerials[]** array only the first few slaves will be considered. You can set the value to 0 if you just want to record from the master device.

SAMPLE_RATE_HZ

Specifies the sample rate in Hz. Please see the API documentation of the `GT_SetSampleRate` function for valid values.

NUMBER_OF_SCANS

The number of complete scans to retrieve at once per GT_GetData call. In the API documentation this value is also referenced as buffer size and is set with the `GT_SetBufferSize` function and depends on the specified sample rate. Please see the API documentation of the `GT_SetSampleRate` function for valid values here.

NUMBER_OF_CHANNELS

This is the number of channels that should be acquired from each device! For example if this value equals 3 and if there are 2 devices, one master and one slave, three channels will be recorded from the master and three channels will be recorded from the slave (which results in a total of 6 recorded channels).
It is important to set this value exactly to the number of elements in the **_channelsToAcquire[]** array.

_channelsToAcquire[]

This array contains the channel numbers that should be acquired from each device. This array must contain exactly as many elements as specified by the **NUMBER_OF_CHANNELS** parameter. For example if **NUMBER_OF_CHANNELS** is set to 3 and **_channelsToAcquire[]** contains elements {2, 5, 6} and there are 2 devices (one master and one slave), channels 2, 5 and 6 of the master device will be recorded as will as channel 2, 5 and 6 of the slave device which results in a total of 6 recorded channels.

_mode

This is the mode the device should be set to. For the demo project please only select `M_NORMAL` or `M_COUNTER`. The modes `M_IMPEDANCE` (in combination with the `GT_GetImpedance` command) and `M_CALIBRATE` (in combination with `GT_SetDAC`) are not used in the demo. Please see the API documentation of the `GT_SetMode` and `GT_GetMode` function for the different modes and their meaning.

_bipolarSettings

This structure will be initialized to zero in the `OpenAndInitDevice(…)` function of the demo project meaning that no bipolar derivation will be performed at all. To change this please go to the `OpenAndInitDevice` method and change the appropriate values for the `_bipolarSettings.Channel1` … `_bipolarSettings.Channel16` fields. See the API documentation of the `GT_SetBipolar` function for the meaning of these values.

_commonReference

Defines the groups that should be connected to common reference. In this example no group is connected to common reference. Please see the API documentation of the `GT_SetReference` function for details.

_commonGround

Defines the groups that should be connected to common ground. In this example no group is connected to common ground. Please see the API documentation of the `GT_SetGround` function for details.

Compiling the source code

You can compile the source code for two different platforms: 32-bit (Win32) and 64-bit (x64). The desired platform can usually be selected through the Visual Studio toolbar or in the Visual Studio Configuration Manager. When compiling with x64 configuration you can execute the created `gUSBampSyncDemo.exe` file only on a 64-bit operating system with the 64-bit g.USBamp driver installed.

Executing the project (gUSBampSyncDemo.exe)

On executing the compiled `gUSBampSyncDemo.exe` file a console window will show up telling you that the g.USBamp devices with the specified serial numbers will be opened and initialized. When the device can be opened and initialized successfully the application starts to receive data for the specified amount of seconds. This data will be written to the file

```
receivedData.bin
```

in the same directory the `gUSBampSyncDemo.exe` was executed from.

Opening the receivedData.bin file

The receivedData.bin file contains the float values for each analog channel of each device and each complete scan in consecutive order.

This binary output file can be read by using MATLAB for example. The file consists of consecutive float values (4 bytes each) that are the read values in microvolts from the devices. If you multiply the number of channels per device times the number of devices you get the number of channels for one complete scan. The file contains a number of those scans, one complete scan following the other. Channel 1 of each scan represents the first channel of the first specified slave device in the slaveSerials array. All channels of the slave devices are in the order the slave devices were given in the array. The values of the channels of the master devices are placed at the end of each scan. So the last value of each scan is the value of the last channel of the master device.

The following MATLAB sample code demonstrates reading data from the receivedData.bin file which contains data of 24 channels recorded with 2 devices (12 channels per device; the value `24` in the second parameter of the fread command specifies the total number of channels recorded from all devices; the `inf` parameter means to read all samples/scans):

```
fid = fopen('receivedData.bin', 'r');
data = fread(fid, [24 inf], 'float32');
```

```
fclose(fid);
```

The `data` matrix now contains the recorded samples of all 24 recorded channels in microvolts. Using MATLAB's `plot` function you can visualize the recorded data.

With the g.USBamp C API software a console application is installed, showing how to program applications involving one or multiple amplifiers. example, open project `gUSBampSyncDemo.vcproj` stored in the folder

`C:\Program Files\gtec\gUSBampCAPI\C++Project\SyncDemo`

(default installation path is assumed)

In the file `gUSBampSyncDemo.cpp` define the following settings in the section `device configuration settings` to work with one amplifier:

Specify the serial number of the device used as master
```
LPSTR _masterSerial = "UA-2006.00.00";
```

Empty the serial number array for the slave device
```
LPSTR _slaveSerials[] = {};
```

Set the number of slave devices to 0
```
const int SLAVE_SERIALS_SIZE = 0;
```

Specify the sampling rate in Hz (see documentation of the g.USBamp C API for details on this value and the NUMBER_OF_SCANS!)
```
const int SAMPLE_RATE_HZ = 256;
```

set the number of scans that should be received simultaneously (depending on the `_sampleRate`; see C-API documentation for this value!)
```
const int NUMBER_OF_SCANS = 8;
```

The number of channels per device that should be acquired (must equal the size of the `_channelsToAcquire` array)
```
const UCHAR NUMBER_OF_CHANNELS = 16;
```

The channels that should be acquired from each device
```
UCHAR _channelsToAcquire[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16};
```

use normal acquisition mode
```
UCHAR _mode = M_NORMAL;
```

Don't use bipolar derivation (all values will be initialized to zero)
```
CHANNEL _bipolarSettings;
```

Don't connect groups to common reference
```
REF _commonReference = { FALSE, FALSE, FALSE, FALSE };
```

Don't connect groups to common ground
```
GND _commonGround = { FALSE, FALSE, FALSE, FALSE };
```

, build the project and start the generated application. The data recorded will be stored in a file named `receivedData.bin`. Data acquisition will run for 10 seconds and then the demo will stop automatically.

To adapt this example to work with multiple amplifiers, add the corresponding serial numbers as strings to `_slaveSerials[]` and set the number of slave devices in `SLAVE_SERIALS_SIZE`.

Data storage

Both programs store the data in float32 format. Use the following MATLAB code to read in the data:

```
Channels=17;

FileName='test.bin';

fid=fopen(FileName,'rb');

data=fread(fid,[Channels inf],'float32');

fclose(fid);
```

Application Program Interface (API)

The API can be accessed through a standard dynamic link library (gUSBamp.dll). To include this interface in your custom application perform the following steps:

- Copy the files *gUSBamp.h* and *gUSBamp.lib* to your Microsoft Visual Studio project directory.

- Be sure that the file *gUSBamp.dll* is on a global search path otherwise copy the file to the output directory of your Microsoft Visual Studio project.

- Add the following lines to your code to access the API:

```
#include "gUSBamp.h"
#pragma comment(lib,"gUSBamp.lib")
```

Function Reference

Data acquisition
GT_OpenDevice(int iPortNumber);
GT_OpenDeviceEx(LPSTR lpSerial);
GT_CloseDevice(HANDLE *hDevice);
GT_GetData(HANDLE hDevice, BYTE *pData, DWORD dwSzBuffer, OVERLAPPED *ov);
GT_SetBufferSize(HANDLE hDevice, WORD wBufferSize);
GT_SetSampleRate(HANDLE hDevice, WORD wSampleRate);
GT_Start(HANDLE hDevice);
GT_Stop(HANDLE hDevice);
GT_SetChannels(HANDLE hDevice, UCHAR *ucChannels, UCHAR ucSizeChannels);
GT_ResetTransfer(HANDLE hDevice);

Digital I/O
GT_SetDigitalOut(HANDLE hDevice, UCHAR ucNumber, UCHAR ucValue);
GT_SetDigitalOutEx(HANDLE hDevice, DigitalOUT dout);
GT_GetDigitalIO(HANDLE hDevice, PdigitalIO pDIO);
GT_GetDigitalOut(HANDLE hDevice, PDigitalOUT pDOUT);
GT_EnableTriggerLine(HANDLE hDevice, BOOL bEnable);

Filter
GT_GetFilterSpec(_FILT *FilterSpec);
GT_GetNumberOfFilter(int* nof);
GT_SetBandPass(HANDLE hDevice, UCHAR ucChannel, int index);
GT_GetNotchSpec(_FILT *FilterSpec);
GT_GetNumberOfNotch(int* nof);
GT_SetNotch(HANDLE hDevice, UCHAR ucChannel, int index);

Mode
GT_SetMode(HANDLE hDevice, UCHAR ucMode);
GT_GetMode(HANDLE hDevice, UCHAR* ucMode);
GT_SetGround(HANDLE hDevice, GND CommonGround) ;
GT_GetGround(HANDLE hDevice, GND* CommonGround);
GT_SetReference(HANDLE hDevice, REF CommonReference);
GT_GetReference(HANDLE hDevice, REF* CommonReference)

Bipolar
GT_SetBipolar(HANDLE hDevice, BIPOLAR bipoChannel);

DRL
GT_SetDRLChannel(HANDLE hDevice, CHANNEL drlChannel);

Short Cut
GT_EnableSC(HANDLE hDevice, BOOL bEnable);

Synchronization
GT_SetSlave(HANDLE hDevice, BOOL bSlave);

Calibration / DRL
GT_SetDAC(HANDLE hDevice, DAC AnalogOut);
GT_SetScale(HANDLE hDevice, PSCALE Scaling);
GT_GetScale(HANDLE hDevice, PSCALE Scaling);
GT_Calibrate(HANDLE hDevice, PSCALE Scaling);

Error handling
GT_GetLastError(WORD *wErrorCode, char *pLastError);

General

GT_GetDriverVersion(void);
GT_GetHWVersion(HANDLE hDevice);
GT_GetSerial(HANDLE hDevice, LPSTR lpstrSerial,UINT uiSize);
GT_GetImpedance(HANDLE hDevice, UCHAR Channel, double* Impedance);

Data Acquisition

hDevice = GT_OpenDevice(int iPortNumber);

Opens the specified g.USBamp and returns a handle to it. The handle must be stored in the program because all function calls of the API need this handle as an input parameter. The handle must be deleted when the application is closed with GT_CloseDevice.

Input:

iPortNumber	INT	number of USB port

Output:

hDevice	HANDLE	handle of the device
		returns NULL if the opening fails

hDevice = GT_OpenDeviceEx(LPSTR lpSerial);

Opens the specified g.USBamp and returns a handle to it. The handle must be stored in the program because all function calls of the API need this handle as an input parameter. The handle must be deleted when the application is closed with GT_CloseDevice.

Input:

lpSerial	LPSTR	pointer to a NULL terminated string containing the device serial e.g. “UA-2005.02.01”

Output:

hDevice	HANDLE	handle of the device
		returns NULL if the opening fails

Status = GT_CloseDevice(HANDLE *hDevice);

Closes the g.USBamp identified by the handle hDevice. The function returns true if the call succeeded otherwise it will return false. Use GT_OpenDevice to retrieve a handle to the g.USBamp.

Input:

*hDevice	HANDLE	handle of the device	

Output:

Status	BOOL	Status=0	not able to close the device
		Status=1	device closed successfully

Status = GT_GetData(HANDLE hDevice, BYTE *pData, DWORD dwSzBuffer, OVERLAPPED ov);

Extract data from the driver buffer. The function does not block the calling thread because of the overlapped mode. The function call returns immediately but data is not valid until the event in the OVERLAPPED structure is triggered. Use WaitForSingleObject() to determine if the transfer has finished. Use GetOverlappedResult() to retrieve the number of bytes that are available.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
*pData	BYTE	pointer to buffer to hold data retrieved from the driver
dwSzBuffer	DWORD	buffer size defines the number of bytes received from the driver. dwSzBuffer must correspond to the value wBufferSize in GT_SetBufferSize. Furthermore HEADER_SIZE bytes precede the acquired data and have to be discarded. e.g. 16 channels sampled at 128 Hz, wBufferSize set to 8: dwSzBuffer = 8 scans * 16 channels * sizeof(float) + HEADER_SIZE;
*ov	OVERLAPPED	pointer to OVERLAPPED structure used to perform the overlapped I/O transfer

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: This function should be used in a separated thread to give best performance and to take advantage from this asynchronous function call. The calling thread is idle while waiting for data and does not consume any CPU power. See the demo code for a reference implementation.

See also GT_SetBufferSize

Status = GT_SetBufferSize(HANDLE hDevice, WORD wBufferSize);

Sets the buffer size of the driver.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
wBufferSize	WORD	number of scans to receive per buffer. Buffer size should be at least 20-30 ms (60 ms recommended). To calculate a 60 ms buffer use following equation: (wBufferSize) >= (sample rate)*(60*10^-3) Example: sample rate = 128 Hz: 128*60*10^-3 = 7.68 so buffer size is 8 scans. See also GT_GetData().

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: The wBufferSize should not exceed 512
See also GT_SetSampleRate(…) for recommended values for wBufferSize.

Status = GT_SetSampleRate(HANDLE hDevice, WORD wSampleRate);

Set the sampling frequency of the g.USBamp. The sampling frequency value must correspond to a value defined in the table below. The over sampling rate depends directly on the selected sampling frequency. A small sampling frequency will result in a higher over sampling rate.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
wSampleRate	WORD	sample rate	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

The possible sampling rates and the corresponding over sampling ratios are shown in the following table. The third column recommends a buffer size that should be used in GT_SetBufferSize(…)

Sampling Rate [Hz]	**Over sampling Rate [Number]**	**Buffer size [number of scans]**
32	1200	1
64	600	2
128	300	4
256	150	8
512	75	16
600	64	32
1200	32	64
2400	16	128
4800	8	256
9600	4	512
19200	2	512
38400	1	512

See also GT_SetBufferSize

Status = GT_Start(HANDLE hDevice);

Start the data acquisition. Note that the sampling frequency, buffer configuration and channels must be set before.

The function returns true if the call succeeded otherwise it will return false.

Note: You have to extract data permanently from driver buffer to prevent a buffer overrun. Please refer to the sample code if you are not sure about this topic. See also GT_GetData.

Input:

hDevice	HANDLE	handle of the device

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_Stop(HANDLE hDevice);

Stop the acquisition.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetChannels(HANDLE hDevice, UCHAR *ucChannels, …

UCHAR ucSizeChannels);

Define the channels that should be recorded.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
*ucChannels	UCHAR	define the channels that should be acquired

[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16] acquires all 16 channels
[1 2 3 4] acquires channels 1 - 4

ucSizeChannels	UCHAR	total number of acquired channels (1-16)
	[16]	acquires 16 channels
	[4]	acquires 4 channels

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_ResetTransfer(HANDLE hDevice);

Reset the driver data pipe after data transmission error (e.g. time out).

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Digital I/O

Status = GT_SetDigitalOut(HANDLE hDevice, UCHAR ucNumber, UCHAR ucValue);

Set digital outputs for g.USBamp version 2.0.

The function returns true if the call succeeded otherwise it will return false.
The function returns false for g.USBamp version 3.0.

Input:

hDevice	HANDLE	handle of the device	
ucNumber	UCHAR	digital output id 1 or 2	
ucValue	UCHAR	TRUE | FALSE	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetDigitalOutEx(HANDLE hDevice, DigitalOUT dout);

Set the digital outputs for g.USBamp version 3.0.

The function returns true if the call succeeded otherwise it will return false.
The function returns false for g.USBamp version 2.0

Input:

hDevice	HANDLE	handle of the device	
dout	DigitalOUT	digital output structure members:	
	BOOL SET_0;	TRUE if digital OUT0 should be set to VALUE_0	
	BOOL VALUE_0;	TRUE – high; FALSE – low	
	BOOL SET_1;	TRUE if digital OUT1 should be set to VALUE_1	
	BOOL VALUE_1;	TRUE – high; FALSE – low	
	BOOL SET_2;	TRUE if digital OUT2 should be set to VALUE_2	
	BOOL VALUE_2;	TRUE – high; FALSE – low	
	BOOL SET_3;	TRUE if digital OUT3 should be set to VALUE_3	
	BOOL VALUE_3;	TRUE – high; FALSE – low	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetDigitalIO(HANDLE hDevice, PDigitalIO pDIO);

Read the state of the digital inputs and outputs of g.USBamp version 2.0.

The function returns true if the call succeeded otherwise it will return false.
The function returns false for g.USBamp version 3.0.

Input:

hDevice	HANDLE	handle of the device
pDIO	PdigitalIO	pointer to DIO structure members:
	BOOL DIN1	TRUE – high; FALSE – low
	BOOL DIN2	TRUE – high; FALSE – low
	BOOL DOUT1	TRUE – high; FALSE – low
	BOOL DOUT2	TRUE – high; FALSE – low

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetDigitalOut(HANDLE hDevice, PDigitalOUT pDOUT);

Read the state of the digital outputs of g.USBamp version 3.0.

The function returns true if the call succeeded otherwise it will return false.
The function returns false for g.USBamp version 2.0.

Input:

hDevice	HANDLE	handle of the device
pDOUT	PDigitalOUT	pointer to digital output structure members:
	BOOL SET_0;	not used
	BOOL VALUE_0;	TRUE – high; FALSE – low
	BOOL SET_1;	not used
	BOOL VALUE_1;	TRUE – high; FALSE – low
	BOOL SET_2;	not used
	BOOL VALUE_2;	TRUE – high; FALSE – low
	BOOL SET_3;	not used
	BOOL VALUE_3;	TRUE – high; FALSE – low

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_EnableTriggerLine(HANDLE hDevice, BOOL bEnable);

Enable or disable the digital trigger line.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
bEnable	BOOL	TRUE - enable, FALSE - disable	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: If enabled, the trigger lines are sampled synchronous with the analog channels data rate. Therefore an additional float value is attached to the analog channels values.
There is a difference between g.USBamp version 3.0 and version 2.0. In version 2.0 there is just one trigger line so the values of the trigger channel can be 0 (LOW) and 250000 (HIGH). In version 3.0 there are 8 trigger lines coded as UINT8 on the trigger channel. If all inputs are HIGH the value of the channel is 255.0. If e.g. input 0 to 3 are HIGH the result is 15.0 ….

Filter

Status = GT_GetFilterSpec(FILT *FilterSpec);

Read in the available bandpass filter settings. Use GT_GetNumberOfFilter() to determine the size of the filter specifying data.
Filter description data is copied to *FilterSpec.

The function returns true if the call succeeded otherwise it will return false.

Input:

*FilterSpec	FILT	pointer to filter structure FILT	
	FLOAT	fu	lower border frequency of filter
	FLOAT	fo	upper border frequency of filter
	FLOAT	fs	sampling rate
	FLOAT	type	filter type 2 - CHEBYSHEV 1 - BUTTERWORTH
	FLOAT	order	filter order

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetNumberOfFilter(int* nof);

Read in the total number of available filter settings.

The function returns true if the call succeeded otherwise it will return false.

Input:

*nof	INT	number of filters	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetBandPass(HANDLE hDevice, UCHAR ucChannel, int index);

Set the digital bandpass filter coefficients for a specific channel.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
ucChannel	UCHAR	channel number (can be 1 to 16)
index	INT	filter id. Use GT_GetFilterSpec to get the filter matrix index set index to -1 if no filter should be applied

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: there is a hardware anti-aliasing filter

Status = GT_GetNotchSpec(FILT *FilterSpec);

Read in the available notch filter settings. Use GT_GetNumberOfNotch() to determine the size of the filter specifying data.
Filter description data is copied to *FilterSpec.

The function returns true if the call succeeded otherwise it will return false

Input:

*FilterSpec	FILT	pointer to filter structure FILT	
	FLOAT	fu	lower border frequency of filter
	FLOAT	fo	upper border frequency of filter
	FLOAT	fs	sampling rate
	FLOAT	type	filter type 1 - CHEBYSHEV 2 - BUTTERWORTH
	FLOAT	order	filter order

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetNumberOfNotch(int* nof);

Read in the total number of available filter settings.

The function returns true if the call succeeded otherwise it will return false.

Input:

*nof	INT	number of filters	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetNotch(HANDLE hDevice, UCHAR ucChannel, int index);

Set the digital notch filter coefficients for a specific channel.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
ucChannel	UCHAR	channel number (can be 1 to 16)
index	INT	filter id. Use GT_GetNotchSpec to get the filter matrix index. set index to -1 if no filter should be applied

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Mode

Status = GT_SetMode(HANDLE hDevice, UCHAR ucMode);

Set the operation mode of the device.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
ucMode	UCHAR	define the operating mode of the amplifier	
	M_NORMAL		acquire data from the 16 input channels
	M_CALIBRATE		calibrate the input channels. Applies a calibration signal onto all input channels
	M_IMPEDANCE		measure the electrode impedance
	M_COUNTER		if channel 16 is selected there is a counter on this channel (overrun at 1e6)

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetMode(HANDLE hDevice, UCHAR* ucMode);

Get the operation mode of the device.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
ucMode	UCHAR*	get the operating mode of the amplifier
	M_NORMAL	acquire data from the 16 input channels
	M_CALIBRATE	calibrate the input channels; applies a calibration signal onto all input channels
	M_IMPEDANCE	measure the electrode impedance

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Connect Ground and Reference

Status = GT_SetGround(HANDLE hDevice, GND CommonGround);

Connect or disconnect the grounds of the 4 groups (A, B, C, D).

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
CommonGround	GND	structure	
	GND1	BOOL	1 – connect, 0 – disconnect group A to common ground
	GND2	BOOL	1 – connect, 0 – disconnect group B to common ground
	GND3	BOOL	1 – connect, 0 – disconnect group C to common ground
	GND4	BOOL	1 – connect, 0 – disconnect group D to common ground

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetGround(HANDLE hDevice, GND* CommonGround);

Get the state of the grounds switches of the 4 groups (A, B, C, D).

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
CommonGround	GND*	pointer to GND structure	
	GND1	BOOL	1 – connect, 0 – disconnect group A to common ground
	GND2	BOOL	1 – connect, 0 – disconnect group B to common ground
	GND3	BOOL	1 – connect, 0 – disconnect group C to common ground
	GND4	BOOL	1 – connect, 0 – disconnect group D to common ground

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetReference(HANDLE hDevice, REF CommonReference);

Connect or disconnect the references of the 4 groups (A, B, C, D).

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
CommonReference	REF		
	ref1	BOOL	1 – connect, 0 – disconnect group A to common reference
	ref2	BOOL	1 – connect, 0 – disconnect group B to common reference
	ref3	BOOL	1 – connect, 0 – disconnect group C to common reference
	ref4	BOOL	1 – connect, 0 – disconnect group D to common reference

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetReference(HANDLE hDevice, REF* CommonReference);

Get the state of the reference switches of the 4 groups (A, B, C, D).

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
CommonReference	REF*	pointer to REF structure	
	ref1	BOOL	1 – connect, 0 – disconnect group A to common reference
	ref2	BOOL	1 – connect, 0 – disconnect group B to common reference
	ref3	BOOL	1 – connect, 0 – disconnect group C to common reference
	ref4	BOOL	1 – connect, 0 – disconnect group D to common reference

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Bipolar Derivation

Status = GT_SetBipolar(HANDLE hDevice, BIPOLAR bipoChannel);

Define the channels for a bipolar derivation.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
bipoChannel	BIPOLAR		
	Channel1	UCHARdefine the channel number for the bipolar derivation with channel 1. If channel 2 is selected, g.USBamp performs a bipolar derivation between channel 1 and 2. Set the channel number to zero if no bipolar derivation should be performed.	
	Channel2	UCHAR	set to channel number or 0
	Channel3	UCHAR	set to channel number or 0
	Channel4	UCHAR	set to channel number or 0
	Channel5	UCHAR	set to channel number or 0
	Channel6	UCHAR	set to channel number or 0
	Channel7	UCHAR	set to channel number or 0
	Channel8	UCHAR	set to channel number or 0
	Channel9	UCHAR	set to channel number or 0
	Channel10	UCHAR	set to channel number or 0
	Channel11	UCHAR	set to channel number or 0
	Channel12	UCHAR	set to channel number or 0
	Channel13	UCHAR	set to channel number or 0
	Channel14	UCHAR	set to channel number or 0
	Channel15	UCHAR	set to channel number or 0
	Channel16	UCHAR	set to channel number or 0

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

DRL

Status = GT_SetDRLChannel(HANDLE hDevice, CHANNEL drlChannel);

Define the channels for the DRL calculation.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
drlChannel	CHANNEL		
	Channel1	UCHAR	set to 1 if the channel should be used for driven right leg calculation, otherwise to 0
	Channel2	UCHAR	0/1
	Channel3	UCHAR	0/1
	Channel4	UCHAR	0/1
	Channel5	UCHAR	0/1
	Channel6	UCHAR	0/1
	Channel7	UCHAR	0/1
	Channel8	UCHAR	0/1
	Channel9	UCHAR	0/1
	Channel10	UCHAR	0/1
	Channel11	UCHAR	0/1
	Channel12	UCHAR	0/1
	Channel13	UCHAR	0/1
	Channel14	UCHAR	0/1
	Channel15	UCHAR	0/1
	Channel16	UCHAR	0/1

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

SHORT CUT

Status = GT_EnableSC(HANDLE hDevice, BOOL bEnable);

Enable or disable the short cut function. If short cut is enabled a high level on the SC input socket of the amplifier disconnects the electrodes from the amplifier input stage.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
bEnable	BOOL	TRUE – enable, FALSE – disable	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

See also GT_SetDAC

Synchronization

Status = GT_SetSlave(HANDLE hDevice, BOOL bSlave);

Set the amplifier to slave/master mode.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
bSlave	BOOL	TRUE – slave mode, FALSE – master mode

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: To synchronize multiple g.USBamps perform the following steps in your application

1. There must be only one device configured as master the others must be configured as slave devices.
2. The sampling rate has to be the same for all amplifiers.
3. Call GT_Start() for the slave devices first. The master device has to be called last.
4. During acquisition call GT_GetData() for all devices before invoking WaitForSingleObject()
5. To stop the acquisition call GT_Stop() for all slave devices first. The master device has to be called last.

Calibration/DRL

Status = GT_SetDAC(HANDLE hDevice, DAC AnalogOut);

Define the calibration/DRL settings.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
AnalogOut	DAC		
	WaveShape	define the output waveshape	
	WS_SQUARE		generate square wave signal
	WS_SAWTOOTH		generate sawtooth signal
	WS_SINE		generate sine wave
	WS_DRL		generate DRL signal
	WS_NOISE		generate white noise
	Amplitude	max: 2000 (250mV), min: -2000 (-250mV)	
	Frequency	frequency in Hz	
	Offset	no offset: 2047, min: 0, max 4096	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_SetScale(HANDLE hDevice, PSCALE Scaling);

Set factor and offset values for all channels.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
Scaling	PSCALE	pointer to SCALE structure, which contains offset and scaling
	offset[i]	contains the offset of channel i in μV
	factor[i]	contains the factor for channel i

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: Values are stored in permanent memory.

Calculation: y = (x – d)*k;
y … values retrieved with GT_GetData (calculated values) in μV
x … acquired data
d … offset value in μV
k … factor

Status = GT_GetScale(HANDLE hDevice, PSCALE Scaling);

Read factor and offset values for all channels.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device	
Scaling	PSCALE	pointer to SCALE structure, which receives offset and scaling	
	offset[i]	contains the offset of channel i in µV	
	factor[i]	contains the factor for channel i	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_Calibrate(HANDLE hDevice, PSCALE Scaling);

Calculate factor and offset values for all channels.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
Scaling	PSCALE	pointer to SCALE structure, which receives offset and scaling
	offset[i]	contains the offset calculated for channel i in µV
	factor[i]	contains the factor calculated for channel i

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Note: Function call is blocking (~4s).
Scaling and offset values in permanent memory reset to 1 and 0.
Use GT_SetScale to write new values to storage.
Use GT_GetScale to verify stored values.

Function call modifies g.USBamp configuration. You need to configure the device after this call to meet your requirements.

See also: GT_GetScale, GT_SetScale

Error handling

Status = GT_GetLastError(WORD *wErrorCode, CHAR *pLastError);

Get the last error message.

The function returns true if the call succeeded otherwise it will return false.

Input:

*wErrorCode	WORD	error id	
*pLastError	CHAR	error message (max. 1024 characters)	

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

General

Version = GT_GetDriverVersion(void);

Return the g.USBamp driver version.

Output:

Version	REAL	g.USBamp driver version

HWVersion = GT_GetHWVersion(HANDLE hDevice);

Return the hardware version of the device.

Output:

HWVersion	FLOAT	g.USBamp hardware version number (2.0 or 3.0)

Status = GT_GetSerial(HANDLE hDevice, LPSTR lpstrSerial,UINT uiSize);

Get the serial number from an opened device.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
lpstrSerial	LPSTR	pointer to character array to receive the serial string
uiSize	UINT	size of the array pointed to by lpstrSerial, should be 16

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

Status = GT_GetImpedance(HANDLE hDevice, UCHAR Channel, double* Impedance);

Measure the electrode impedance for specified channel.

The function returns true if the call succeeded otherwise it will return false.

Input:

hDevice	HANDLE	handle of the device
Channel	UCHAR	channel number (1 … 20)
Impedance	double*	pointer to double which receives the value of electrode impedance on specified channel

Output:

Status	BOOL	Status=0	command not successful
		Status=1	command successful

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 you must enter the following channel numbers:

Channel number	Electrode
1 ..16	1 ..16
17	REFA
18	REFB
19	REFC
20	REFD

Remark: Function call modifies g.USBamp configuration. You need to configure the device after this call to meet your requirements.

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