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 V3.14.00

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**

Content

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Release notes help to learn about new features and changes of g.USBamp C API and tools when upgrading to a newer version of the software.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

- gUSBampSyncDemo improved: trigger channel can be selected for acquisition easily now.
- Added C# .NET demo source code for acquiring data from multiple synchronized amplifiers.

Changes

- Improved description on how to include both 32-bit and 64-bit libraries in the demo project.

Related documents

gUSBamp30_InstructionsForUse.pdf / gUSBamp30_InstructionsForUse_CE.pdf – a detailed hardware description of the device (sockets, labeling …); comes with the device

gUSBampDriver.pdf – a detailed description of the driver installation and start-up of the device; comes with the device driver

The intended function of the equipment

Measuring, recording and analysis of electrical activity of the brain (EEG) and/or through the attachment of multiple electrodes at various locations to aid in monitoring and diagnosis as routinely found in clinical settings for the EEG.

The device must not be used for patient monitoring. The device must not be used for the determination of brain death. Additional examinations are needed for diagnosis and no diagnosis may be done only based on using this 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 or notebook workstation or an embedded computer running Microsoft Windows.

The table below lists optimal requirements:

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 2010 is necessary

Software	**Version**
Windows	Windows 7 Professional English Win32 or Win64
Microsoft Visual Studio	2010 SP1
Acrobat Reader	11.0.04
g.USBamp driver	3.14.00

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\Demo
```

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).

Synchronization of multiple g.USBamps: the gUSBampSyncDemo project

Two Visual Studio 2010 demo projects come with your g.USBamp C-API installation, one for C++ and one for C# .NET. They are intended to demonstrate the usage of the g.USBamp C-API for one or more synchronized g.USBamp amplifiers.

The Visual Studio 2010 project file of the C++ demo project should be located under:

```
C:\Program Files\gtec\gUSBampCAPI\Demo\C++ (native)\gUSBampSyncDemo.vcxproj
```

The Visual Studio 2010 project file of the C# .NET demo project should be located under:

```
C:\Program Files\gtec\gUSBampCAPI\Demo\C# (.NET)\gUSBampSyncDemoCS.csproj
```

On execution, the demo projects will open the specified g.USBamp devices, initialize them and record data for a specific amount of seconds to a binary file on the hard disk. Before you compile and execute the demo projects please read the following section to configure them 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 has to be connected to the SYNC IN of the first slave device, the SYNC OUT of the first slave device must then be connected to the SYNC IN of the second slave device and so on. Ensure that all devices are switched on.

Modifying the C++ source code parameters

In 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 BOOL TRIGGER = FALSE;
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(s) and stores them 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 seconds 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 be sufficient.

_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. It is very important that you set the **SLAVE_SERIAL_SIZE** parameter to the number of the contained slave devices in this array. Data will be acquired from the devices in the specified ordering, having the master device at the end.

SLAVE_SERIALS_SIZE

This value specifies the number of contained slave device serial numbers in the **_slaveSerials[]** array. This value must not be greater 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 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, 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.

TRIGGER

Specifies whether the synchronously sampled digital trigger line inputs should be included in data acquisition. If so, the bit-encoded value of all those trigger inputs of a specific device are appended in an additional channel for that device. See the `GT_EnableTriggerLine` function for details.

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 two 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 six 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, **_channelsToAcquire[]** contains elements {2, 5, 6} and there are two devices (one master and one slave), channels 2, 5 and 6 of the master device will be recorded as well as channel 2, 5 and 6 of the slave device which results in a total of six recorded channels.

_mode

This is the mode the device(s) 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` functions 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. 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.

Modifying the C# .NET source code parameters

The `Program.cs` file is the main entry point of the C# .NET demo project. Serial numbers of the amplifiers used for data acquisition can be specified in the following line within the `Main`() method:

```
//create device configurations
Dictionary<string, DeviceConfiguration> devices =
      CreateDefaultDeviceConfigurations("UB-0000.00.00", "UB-2009.07.12");
```

The first amplifier is used as master, all remaining as slaves. In the example above, the amplifier with serial number `UB-0000.00.00` is used as master, and synchronized with amplifier `UB-2009.07.12` which is used as slave.

In the `CreateDefaultDeviceConfigurations` method, the configuration of each of the specified devices can be modified. The `DeviceConfiguration` structure and its members are documented in `DataAcquisitionUnit.cs`.

Compiling the source code

To be able to compile the source code, ensure that all required files are in their appropriate location as described in the section *Application Program Interface (API)* below.

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 executable only on a 64-bit operating system with the 64-bit g.USBamp driver installed.

Executing the project

Ensure that the g.USBamp driver is installed before executing the demos. On executing the compiled `gUSBampSyncDemo.exe` (C++) or `gUSBampSyncDemoCS.exe` (C# .NET), 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(s) can be opened and initialized successfully the application starts to receive data for the specified amount of seconds. These data will be written to the file

```
receivedData.bin
```

in the same directory the corresponding executable was executed from (if you run the program from the Visual Studio IDE, the working directory is set to the project directory and the file will therefore be written to the directory that contains the Visual Studio project file).

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) representing the measured values from the devices in microvolts. If trigger lines were included in acquisition, each device returns one channel in addition to the selected analog channels. 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.

In the C++ application, 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 device 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.

In the C# application, channel 1 of each scan represents the first channel of the first device specified in the `CreateDefaultDeviceConfigurations` method, which is configured as master device. The channels of the slave devices follow in the specified ordering.The last value of each scan is the value of the last channel of the last specified slave device.

The following MATLAB sample code demonstrates reading data from the `receivedData.bin` file which contains data of 24 channels recorded with two devices (twelve 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 available data):

```
fid = fopen('receivedData.bin', 'rb');
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. If trigger lines were selected for acquisition in this example, the number of recorded channels would be 26 (1 additional channel for each of the two devices).

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 file `gUSBamp.h` to your Microsoft Visual Studio project directory.
- Copy the 32-bit version of `gUSBamp.lib` to your Microsoft Visual Studio project directory and rename it to `gUSBamp_x86.lib`.
- Copy the 64-bit version of `gUSBamp.lib` to your Microsoft Visual Studio project directory and rename it to `gUSBamp_x64.lib`.
- Be sure that the g.USBamp driver appropriate for your platform is installed as described in the section: *Installation from a CD*, *Deployment*. It provides the `gUSBamp.dll` on a global system search path so that your application will find it on execution. Only for debugging, you can alternatively copy the `gUSBamp.dll` and the four associated filter files (`*.bin`) appropriate for the platform you compile for and execute on to the output directory of your Microsoft Visual Studio project (but NEVER ship these files together with your application).
- Add the following lines to your code to link the API libraries with your project:

```
#ifdef _WIN64
      #pragma comment(lib, "gUSBamp_x64.lib")
#else
      #pragma comment(lib, "gUSBamp_x86.lib")
#endif
```

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

HANDLE 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 released using GT_CloseDevice() after using the device or when the application is closed.

Input:

iPortNumber	INT	The current index of the USB device. (When an additional USB device is connected to or a device is removed from the PC the indices of the devices may change (operating system related)).

Output:

hDevice	HANDLE	A handle of the device or `NULL` if opening fails.

HANDLE 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 released using GT_CloseDevice() after using the device or when the application is closed.

Input:

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

Output:

hDevice	HANDLE	A handle of the device or `NULL` if opening fails.

BOOL 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	The handle of the device.	

Output:

Status	BOOL	status=0	Not able to close the device.
		status=1	Device closed successfully.

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

Extracts 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 when 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 returns false.

Input:

hDevice	HANDLE	The handle of the device.
*pData	BYTE	A pointer to a buffer that receives data from the driver.
dwSzBuffer	DWORD	The buffer size which 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	A pointer to an 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 time. See the demo code for a reference implementation.

See also GT_SetBufferSize().

BOOL status = GT_SetBufferSize(HANDLE hDevice, WORD wBufferSize);

Sets the buffer size of the driver.

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

Input:

hDevice	HANDLE	The handle of the device
wBufferSize	WORD	The 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.

BOOL status = GT_SetSampleRate(HANDLE hDevice, WORD wSampleRate);

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

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

Input:

hDevice	HANDLE	The handle of the device.
wSampleRate	WORD	The sampling rate.

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

The possible sampling rates and the corresponding oversampling ratios are shown in the following table. The third column recommends a buffer size to use in GT_SetBufferSize().

Sampling Rate [Hz]	Oversampling 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().

BOOL status = GT_Start(HANDLE hDevice);

Starts 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 returns 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	The handle of the device.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_Stop(HANDLE hDevice);

Stops data acquisition.

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

Input:

hDevice	HANDLE	The handle of the device	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_SetChannels(HANDLE hDevice, UCHAR *ucChannels, …

UCHAR ucSizeChannels);

Defines the channels to record.

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

Input:

hDevice	HANDLE	The handle of the device.
*ucChannels	UCHAR	Defines the channels to acquire. Examples: [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	The length of the ucChannels array, i.e. the total number of acquired channels (1-16).

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_ResetTransfer(HANDLE hDevice);

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

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

Input:

hDevice	HANDLE	The handle of the device.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

Digital I/O

BOOL status = GT_SetDigitalOut(HANDLE hDevice, UCHAR ucNumber, UCHAR ucValue);

Sets the digital outputs for g.USBamp version 2.0.

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

Input:

hDevice	HANDLE	The handle of the device.
ucNumber	UCHAR	The digital output id to set (1 or 2).
ucValue	UCHAR	The value of the digital output id to set (TRUE or FALSE).

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL 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 returns false.
The function returns false for g.USBamp version 2.0.

Input:

hDevice	HANDLE	The handle of the device.
dout	DigitalOUT	The DigitalOUT 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.

BOOL status = GT_GetDigitalIO(HANDLE hDevice, PDigitalIO pDIO);

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

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

Input:

hDevice	HANDLE	The handle of the device.
pDIO	PDigitalIO	A pointer to the DigitalIO 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.

BOOL status = GT_GetDigitalOut(HANDLE hDevice, PDigitalOUT pDOUT);

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

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

Input:

hDevice	HANDLE	The handle of the device.
pDOUT	PDigitalOUT	A pointer to DigitalOUT 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.

BOOL status = GT_EnableTriggerLine(HANDLE hDevice, BOOL bEnable);

Enables or disables the digital trigger line.

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

Input:

hDevice	HANDLE	The handle of the device.
bEnable	BOOL	TRUE to enable the digital trigger line, FALSE to disable it.

Output:

wtatus	BOOL	status=0	Command not successful.
		status=1	Command successful.

Note: If enabled, the trigger lines are sampled synchronously to the analog channels. 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. If e.g. input 0 to 3 are HIGH the value is 15…

Filter

BOOL status = GT_GetFilterSpec(FILT *FilterSpec);

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

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

Input:

*FilterSpec	FILT	A pointer to a filter structure FILT with the following members:	
	FLOAT	fu	The lower cutoff frequency of the filter.
	FLOAT	fo	The upper cutoff frequency of the filter.
	FLOAT	fs	The sampling rate of the filter.
	FLOAT	type	The filter type, which can be one of the following: 2 - CHEBYSHEV 1 - BUTTERWORTH
	FLOAT	order	The filter order.

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_GetNumberOfFilter(int* nof);

Reads in the total number of available filter settings.

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

Input:

*nof	INT	The number of available bandpass filters.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

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

Sets the digital bandpass filter coefficients for a specific channel.

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

Input:

hDevice	HANDLE	The handle of the device.
ucChannel	UCHAR	The channel number to set the filter for (can be 1 to 16).
index	INT	The filter id to set for the specified channel. 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.

BOOL status = GT_GetNotchSpec(FILT *FilterSpec);

Reads in the available notch filter settings. Use GT_GetNumberOfNotch() to determine the required size of the *FilterSpec array. Filter description data is copied into *FilterSpec.

The function returns true if the call succeeded, otherwise it returns false

Input:

*FilterSpec	FILT	A pointer to a filter structure FILT with the following members:	
	FLOAT	fu	The lower cutoff frequency of filter.
	FLOAT	fo	The upper cutoff frequency of filter.
	FLOAT	fs	The sampling rate of the filter.
	FLOAT	type	The filter type, which can be one of the following: 1 - CHEBYSHEV 2 - BUTTERWORTH
	FLOAT	order	The filter order.

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_GetNumberOfNotch(int* nof);

Reads in the total number of available filter settings.

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

Input:

*nof	INT	The number of available notch filters.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL 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 returns false.

Input:

hDevice	HANDLE	The handle of the device.
ucChannel	UCHAR	The channel number to set the filter for (can be 1 to 16).
index	INT	The filter id to set for the specified channel. 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

BOOL status = GT_SetMode(HANDLE hDevice, UCHAR ucMode);

Sets the operation mode of the device.

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

Input:

hDevice	HANDLE	The handle of the device.	
ucMode	UCHAR	Defines the operating mode of the amplifier.	
	M_NORMAL		Acquires data from the 16 input channels.
	M_CALIBRATE		Calibrates the input channels. Applies a calibration signal onto all input channels.
	M_IMPEDANCE		Measures the electrode impedance.
	M_COUNTER		Applies a counter on channel 16 if selected for acquisition (overrun at 1e6).

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_GetMode(HANDLE hDevice, UCHAR* ucMode);

Gets the operation mode of the device.

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

Input:

hDevice	HANDLE	The handle of the device.
ucMode	UCHAR*	Receives the operating mode of the amplifier.
	M_NORMAL	Acquires data from the 16 input channels.
	M_CALIBRATE	Calibrates the input channels. Applies a calibration signal onto all input channels.
	M_IMPEDANCE	Measure the electrode impedance.
	M_COUNTER	Applies a counter on channel 16 if selected for acquisition (overrun at 1e6).

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

Connect Ground and Reference

BOOL status = GT_SetGround(HANDLE hDevice, GND CommonGround);

Connects or disconnects the grounds of the 4 groups (A, B, C, D).

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

Input:

hDevice	HANDLE	The handle of the device.	
CommonGround	GND	A GND structure with the following members:	
	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.

BOOL status = GT_GetGround(HANDLE hDevice, GND* CommonGround);

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

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

Input:

hDevice	HANDLE	The handle of the device.	
CommonGround	GND*	A pointer to a GND structure with the following members:	
	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.

BOOL status = GT_SetReference(HANDLE hDevice, REF CommonReference);

Connects or disconnects the references of the 4 groups (A, B, C, D).

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

Input:

hDevice	HANDLE	The handle of the device.	
CommonReference	REF	A REF structure with the following members:	
	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.

BOOL status = GT_GetReference(HANDLE hDevice, REF* CommonReference);

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

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

Input:

hDevice	HANDLE	The handle of the device.	
CommonReference	REF*	A pointer to a REF structure with the following members:	
	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

BOOL status = GT_SetBipolar(HANDLE hDevice, BIPOLAR bipoChannel);

Defines the channels for bipolar derivation.

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

Input:

hDevice	HANDLE	The handle of the device.	
bipoChannel	BIPOLAR	A BIPOLAR structure with the following members:	
	Channel1	UCHAR	Defines 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

BOOL status = GT_SetDRLChannel(HANDLE hDevice, CHANNEL drlChannel);

Defines the channels for Driven Right Leg (DRL) calculation.

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

Input:

hDevice	HANDLE	The handle of the device.	
drlChannel	CHANNEL	A CHANNEL structure with the following members:	
	Channel1	UCHAR	Set to 1 if the channel should be used for driven right leg calculation, otherwise set 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

BOOL status = GT_EnableSC(HANDLE hDevice, BOOL bEnable);

Enables or disables 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 returns false.

Input:

hDevice	HANDLE	The 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

BOOL status = GT_SetSlave(HANDLE hDevice, BOOL bSlave);

Sets the amplifier to slave/master mode.

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

Input:

hDevice	HANDLE	The 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 at last.

Calibration/DRL

BOOL status = GT_SetDAC(HANDLE hDevice, DAC AnalogOut);

Defines the calibration/DRL settings.

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

Input:

hDevice	HANDLE	The handle of the device.	
AnalogOut	DAC	A DAC structure with the following members:	
	WaveShape	Defines the output waveshape and can be one of the following values:	
		WS_SQUARE	Generates square wave signal.
		WS_SAWTOOTH	Generates sawtooth signal.
		WS_SINE	Generates sine wave.
		WS_DRL	Generates DRL signal.
		WS_NOISE	Generates white noise.
	Amplitude	max: 2000 (250mV), min: -2000 (-250mV).	
	Frequency	The frequency in Hz.	
	Offset	No offset: 2047, min: 0, max 4096.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

BOOL status = GT_SetScale(HANDLE hDevice, PSCALE Scaling);

Sets factor and offset values for all channels.

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

Input:

hDevice	HANDLE	The handle of the device.
Scaling	PSCALE	A pointer to a 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

BOOL status = GT_GetScale(HANDLE hDevice, PSCALE Scaling);

Reads factor and offset values for all channels.

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

Input:

hDevice	HANDLE	The handle of the device.
Scaling	PSCALE	A pointer to a 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.

BOOL status = GT_Calibrate(HANDLE hDevice, PSCALE Scaling);

Calculates factor and offset values for all channels.

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

Input:

hDevice	HANDLE	The handle of the device.
Scaling	PSCALE	A pointer to a 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

BOOL status = GT_GetLastError(WORD *wErrorCode, CHAR *pLastError);

Retrieves the last error message.

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

Input:

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

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

General

FLOAT version = GT_GetDriverVersion(void);

Returns the g.USBamp driver version.

Output:

Version	FLOAT	g.USBamp driver version.

FLOAT hwVersion = GT_GetHWVersion(HANDLE hDevice);

Returns the hardware version of the device.

Output:

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

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

Retrieves the serial number from an opened device.

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

Input:

hDevice	HANDLE	The handle of the device.	
lpstrSerial	LPSTR	A pointer to a pre-allocated character array to receive the serial string.	
uiSize	UINT	The size of the lptstrSerial array. Should be 16.	

Output:

status	BOOL	status=0	Command not successful.
		status=1	Command successful.

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

Measures the electrode impedance for the specified channel.

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

Input:

hDevice	HANDLE	The handle of the device.
Channel	UCHAR	The channel number to measure (1 … 20).
Impedance	double*	A pointer to a variable that receives the specified electrode's impedance value in ohm.

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 have to 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.


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