USBwiz™
Library Manual

GHI Electronics, LLC

www.ghielectronics.com

Updated – September 6, 2006

Table of Contents

1.The Library 6
1.1. Getting Started 6
1.2. The Library TYPEs 6
1.3. The Library INCLUDEs “Reducing the code size” 6
1.3.1 How To Use Them With The Pic Examples? 7
1.3.2 _INCLUDE_FAT_SYSTEM_SUPPORT_ 7
1.3.3 _INCLUDE_EXTENDED_FAT_SYSTEM_SUPPORT_ 8
1.3.4 _INCLUDE_READWRITE_WRAPPERS_SUPPORT_ 8
1.3.5 _INCLUDE_TIME_WORK_SUPPORT_ 8
1.3.6 _INCLUDE_STORAGE_SECTOR_SUPPORT_ 9
1.3.7 _INCLUDE_HID_SUPPORT_ 9
1.3.8 _INCLUDE_PRINTER_SUPPORT_ 9
1.3.9 _INCLUDE_SERIAL_DEVICES_SUPPORT_ 10
1.3.10 _INCLUDE_EXTRA_COMMANDS_SUPPORT_ 10
1.3.11 _INCLUDE_OLD_STORAGE_FUNCTIONS_WRAPPERS_SUPPORT_ 10
1.4. The Library Functions 11
1.4.1 The Return Value 11
1.4.2 General Functions 11
1.4.2.1 int16 GHI_GetLibraryVersion(void) 11
1.4.2.2 int8 GHI_GetResult(void) 11
1.4.2.3 int8 GHI_GetVersion(int8* major,int8 * BCDminor) 11
1.4.2.4 int8 GHI_SetUSBwizBaudRate(int32 bauderate) 12
1.4.3 More functionality 12
1.4.3.1 int8 GHI_UpdateFirmware(int8 device) 12
1.4.3.2 int8 GHI_ReadDeviceInfo(int8 device, DEVICE_INFO* info) 12
1.4.3.3 int8 GHI_SoftwareReset(void) 13
1.4.4 FAT File System Functions 13
1.4.4.1 int8 GHI_MountFATFileSystem(int8 device) 13
1.4.4.2 int8 GHI_SwitchDevice(int8 device) 13
1.4.4.3 int8 GHI_ChangeDirectory(int8* filename) 14
1.4.4.4 int8 GHI_MakeDirectory(int8* filename) 14
1.4.4.5 int8 GHI_OpenFile(int8 filehandle, int8 *filename, int8 openmode) 14
1.4.4.6 int8 GHI_SendWriteCommand(int8 filehandle, int32 desireddatasize) 14
1.4.4.7 int8 GHI_SendReadCommand(int8 filehandle, int32 desireddatasize, int8 filler) 15
1.4.4.8 int8 GHI_GetReadAndWriteResult(int32 * actualdatasize) 15
1.4.4.9 int8 GHI_CloseFile(int8 filehande) 16
1.4.4.10 int8 GHI_DeleteFile(int8 *filename) 16
1.4.5 The Extended FAT File System Functions 16
1.4.5.1 void GHI_StartMediaStatistics (void) 16
1.4.5.2 int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free) 17
1.4.5.3 void GHI_StartQformat(void) 17
1.4.5.4 int8 GHI_InitGetFile(void) 17

1.4.5.5 int8 GHI_GetNextFile(int8 * file_name, int8 * file_ext, int8 *attributes, int32 * size)..... 17
1.4.5.6 int8 GHI_RemoveDirectory(int8* filename)..... 17
1.4.5.7 int8 GHI_SendReadWriteFileCommand(int8 readhandle, int8 writehandle, int32 size)..... 18
1.4.5.8 int8 GHI_SendShadowWriteTwoFiles(int8 firstfilehandle, int8 secondfilehandle, int32 desireddatasize)..... 18
1.4.5.9 int8 GHI_GetShadowWriteTwoFileResults(int8 *fresult1, int8 *fresult2, int32 *writtendata1, int32 *writtendata2)..... 18
1.4.5.10 int8 GHI_SendShadowWriteThreeFiles(int8 firstfilehandle, int8 secondfilehandle, int8 thirdfilehandle, int32 desireddatasize)..... 19
1.4.5.11 int8 GHI_GetShadowWriteThreeFileResults(int8 *fresult1, int8 *fresult2, int8 *fresult3, int32 *writtendata1, int32 *writtendata2, int32 *writtendata3)..... 19
1.4.5.12 int8 GHI_SeekFile(int8 filehandle, int32 newposition)..... 20
1.4.5.13 int8 GHI_GetPointerPosition(int8 filehandle, int32 *sector, int16* positioninsector)..... 20
1.4.5.14 int8 GHI_SplitFile(int8 sourcehandle, int8 desthandle1, int8 desthandle2, int32 splitposition, int32 * actualdestsize1, int32 * actualdestsize2) 21
1.4.5.15 int8 GHI_FlushFile(int8 filehandle)..... 21
1.4.5.16 int8 GHI_GetFileInfo(int8 *filename, int32 *size, int8 *attributes, int32 *TimeDate)..... 21
1.4.5.17 int8 GHI_RenameFile(int8 *filename,int8 *newname)..... 22
1.4.5.18 int8 GHI_SetFileSize(int8 filehandle,int32 newsize)..... 22
1.4.6 Write/ Read wrappers..... 22
1.4.6.1 int8 GHI_ReadFile(int8 handle, int8 *buffer, int32 size, int8 filler, int32 *actualdatasize)..... 22
1.4.6.2 int8 GHI_WriteFile(int8 handle, int8 *buffer, int32 size, int32*actualdatasize)..... 23
1.4.7 Real Time Clock work functions..... 23
1.4.7.1 int8 GHI_InitializeTime(int8 backup)..... 23
1.4.7.2 int8 GHI_SetTime(int32 time)..... 24
1.4.7.3 int8 GHI_GetTime(int32 * time)..... 24
1.4.7.4 int8 GHI_GetFormattedTime(int8 *buffer)..... 24
1.4.7.5 int32 GetFATTimeStructure(int32 year, int32 month, int32 day, int32 hours, int32 minutes, int32 seconds)..... 24
1.4.8 Sector storage functions..... 25
1.4.8.1 int8 GHI_RegisterSD(void)..... 25
1.4.8.2 int8 GHI_RegisterUSBMassStorage(int8 device)..... 25
1.4.8.3 int8 GHI_ReadSector(int32 sectornum)..... 26
1.4.8.4 int8 GHI_WriteSector(int32 sectornum)..... 26
1.4.8.5 int8 GHI_SwitchDevice(int8 device)..... 26
1.4.9 Using USB HID Devices (Mouse, keyboard, joystick...)..... 26
1.4.9.1 int8 GHI_RegisterHID(int8 device)..... 27

1.4.9.2 int8 GHI_PrepareHIDReport(int8 device, int8 requested_size, int8 * actual_reportsize)..... 27
1.4.10 Using USB Printers..... 27
1.4.10.1 int8 GHI_RegisterPrinter(int8 device)..... 28
1.4.10.2 int8 GHI_ResetPrinter(int8 device)..... 28
1.4.10.3 int8 GHI_GetPrinterStatus(int8 device, int8 * status)..... 28
1.4.10.4 int8 GHI_PrinterPrint(int8 device, int8 size)..... 28
1.4.11 Using Serial Devices..... 28
1.4.11.1 int8 GHI_RegisterSerialDevice(int8 device, int8 type, int32 baudrate) 29
1.4.11.2 int8 GHI_SerialWrite(int8 device, int8 size)..... 29
1.4.11.3 int8 GHI_SerialRead(int8 device, int8 *sent_data_size)..... 29
1.4.12 Old Functions..... 29
1.4.12.1 int8 GHI_BL_LoadFirmware(int8 drive)..... 30
1.4.12.2 int8 GHI_GetResults(void)..... 30
1.4.12.3 int8 GHI_SetARMBaudRate(int32 bauderate)..... 30
1.4.12.4 int8 GHI_AttachStorageMedia(int8 device, int8 deviceorder, int8 LUN)..... 30
1.4.12.5 int8 GHI_ReadSectorFromCurrentFileSystem(int32 sectornum)..... 30
1.4.12.6 int8 GHI_WriteSectorFromCurrnetFileDydtem(int32 sectornum)..... 31
1.4.12.7 int8 GHI_MountFileSystem(int8 filesystemorder, int8 device, int8 deviceorder)..... 31
1.4.12.8 int8 GHI_SwitchFileSystem(int8 filesystemorder);..... 31
1.4.12.9 int8 GHI_EnumurateUSBDevicetoRootHub(int8 usbport, int8 usbdevicehandle);..... 31
1.4.12.10 int8 GHI_ReleaseUSBDeviceHandle(int8 usbdevicehandle)..... 31
1.4.12.11 int8 GHI_RegisterMassStorageDevice(int8 usbdevicehandle, int8 massstoragedevicehandle, int8 *lastLUN)..... 31
1.4.12.12 int8 InitializeFATMedia(int8 DriveLetter)..... 32
1.4.12.13 int8 SwitchToFATMedia(int8 DriveLetter)..... 32
1.4.12.14 int8 GHI_GetResultsQformat(void)..... 32
2.Driver Functions..... 33
void GHI_Sleep(int16 ms);..... 33
int8 GHI_OpenInterface(void);..... 33
int8 GHI_CloseInterface(void);..... 34
int8 GHI_SetBaudRate(int32 baud);..... 34
int8 GHI_GetC(void);..... 34
void GHI_PutC(int8 ch);..... 34
void GHI_PutS(int8 * str);..... 35
int8 GHI_DataIsReady(void);..... 35
void GHI_ToggleWakePin(void);..... 35
3.Examples..... 36
3.1.Simple file write/read process..... 36
3.2.Read/Write Wrappers 37
3.3.Enumerating Files..... 38
3.4.Read and write simultaneously..... 39

3.5.Write to multiple files simultaneously " Shadow writing"....40
3.6.Real Time Clock....41
3.6.1 Using the FAT Time Structure directly....41
3.6.2 Another way to use the FAT Time Structure....43
3.7.HIDs (Mouse, keyboard, ...)....43
3.8.Printers....44
3.9.Serial Devices....48
Appendix A:....50
FAT Time Structure....50
FAT Attribute Structure....50
Appendix B: Error Codes....52

1. The Library

1.1. Getting Started

When you download USBwiz_lib.zip file you will get a complete MPLAB project for MCC18 compiler. To get the library going to your system, you need to rewrite or modify GHI_inter.c. Use the file PIC_example.c as a starting point for you application. The final 'C' file is USBwiz_lib.c. You shouldn't need to make any changes on this file.

1.2. The Library TYPEs

First thing you need to know is that we define our own variable types. These types are in "types.h" This file also includes the type of project options. Take a look at it.

Type	Limits	Byte count
int8	0 to 255	1
int16	0 to 65535	2
int32	0 to 4,294,967,295	4

1.3. The Library INCLUDEs "Reducing the code size"

USBwiz has a lot of functionality, and so is the library. Depending on the product requirements, you might not use all this functionality. Furthermore, if the product memory is relatively small to include all of the library and the user code, you might want to exclude the library functions that are not used.

For example, if the product communicates with HID and Serial Devices, you can conveniently exclude the storage driver from the library to reduce the code size. This is easily done using the INCLUDE defines at the top of "USBwiz_lib.h"

list of the available support

```
#define         _INCLUDE_EXTRA_COMMANDS_SUPPORT_

#define         _INCLUDE_FAT_SYSTEM_SUPPORT_
#ifdef  _INCLUDE_FAT_SYSTEM_SUPPORT_
        #define         _INCLUDE_EXTENDED_FAT_SYSTEM_SUPPORT_
        #define         _INCLUDE_READWRITE_WRAPPERS_SUPPORT_
#endif
```

```
#define _INCLUDE_TIME_WORK_SUPPORT_

#define          _INCLUDE_STORAGE_SECTOR_SUPPORT_

#define _INCLUDE_HID_SUPPORT_
#define _INCLUDE_PRINTER_SUPPORT_
#define _INCLUDE_SERIAL_DEVICES_SUPPORT_
// Do not use unless necessary
// #define
_INCLUDE_OLD_STORAGE_FUNCTIONS_WRAPPERS_SUPPORT_
```

In the previous example, you can easily exclude the FAT support by commenting the following line: // #define _INCLUDE_FAT_SYSTEM_SUPPORT_

1.3.1 How To Use Them With The Pic Examples?

When the library is downloaded you get a complete MPLAB project for MCC18 compiler. To reduce the code size on the pic, you can also use the INCLUDEs defines; When commenting a INCLUDE, you exclude the library functions as well as the corresponding example.
Note: There is some dependency in the examples. If you exclude the FAT system, the printer example won't compile. Because the printer example reads a file from a storage media.
So if you want to use the examples and exclude the FAT SUPPORT, you have to exclude the PRINTER SUPPORT to let the code compile.
NOTE: The file system tests require that you include any extended INCLUDE SUPPORT for FAT system and TIME WORK SUPPORT as well.

1.3.2 _INCLUDE_FAT_SYSTEM_SUPPORT_

If defined the following functions are available:

```
int8 GHI_MountFATFileSystem(int8 device);
int8 GHI_SwitchDevice(int8 device);
int8 GHI_ChangeDirectory(int8* filename);
int8 GHI_MakeDirectory(int8* filename);
int8 GHI_OpenFile(int8 filehandle, int8 *filename, int8 openmode);
int8 GHI_SendReadCommand(int8 filehandle, int32 desireddatasize, int8 filler);
int8 GHI_SendWriteCommand(int8 filehandle, int32 desireddatasize);
int8 GHI_GetReadAndWriteResult(int32 * actualdatasize);
int8 GHI_CloseFile(int8 filehande);
int8 GHI_DeleteFile(int8 *filename);
```

1.3.3 _INCLUDE_EXTENDED_FAT_SYSTEM_SUPPORT_

If defined the following functions are available:

```
void GHI_StartMediaStatistics (void);
int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free);
void GHI_StartQformat(void);
int8 GHI_InitGetFile(void);
int8 GHI_GetNextFile(int8 * file_name, int8 * file_ext, int8 * attributes, int32 *
size);
int8 GHI_RemoveDirectory(int8* filename);
int8 GHI_SendReadWriteFileCommand(int8 readhandle, int8 writehandle, int32
size);
int8 GHI_SendShadowWriteTwoFiles(int8 firstfilehandle, int8 secondfilehandle,
int32    desireddatasize);
int8 GHI_GetShadowWriteTwoFileResults(int8 *fresult1, int8 *fresult2, int32
        *writtendata1, int32 *writtendata2);
int8 GHI_SendShadowWriteThreeFiles(int8 firstfilehandle, int8 secondfilehandle,
int8      thirdfilehandle, int32 desireddatasize);
int8 GHI_GetShadowWriteThreeFileResults(int8 *fresult1, int8 *fresult2, int8
        *fresult3, int32 *writtendata1, int32 *writtendata2, int32 *writtendata3);
int8 GHI_SeekFile(int8 filehandle, int32 newposition);
int8 GHI_GetPointerPosition(int8 filehandle, int32 *sector, int16
*positioninsector);
int8 GHI_SplitFile(int8 sourcehandle, int8 desthandle1,  int8 desthandle2, int32
        splitposition, int32 * actualdestsize1, int32 * actualdestsize2);
int8 GHI_FlushFile(int8 filehandle);
int8 GHI_GetFileInfo(int8 *filename, int32 *size, int8 *attributes, int32
*TimeDate);
int8 GHI_RenameFile(int8 *filename,int8 *newname);
int8 GHI_SetFileSize(int8 filehandle,int32 newsize);
```

1.3.4 _INCLUDE_READWRITE_WRAPPERS_SUPPORT_

If defined the following functions are available:

```
int8 GHI_ReadFile(int8 handle, int8 *buffer, int32 size, int8 filler, int32
        *actualdatasize);
int8 GHI_WriteFile(int8 handle, int8 *buffer, int32 size, int32 *actualdatasize);
```

1.3.5 _INCLUDE_TIME_WORK_SUPPORT_

If defined the following functions and structures are available:

```
int8 GHI_InitializeTime(int8 backup);
int8 GHI_SetTime(int32 time);
int8 GHI_GetTime(int32 * time);
int8 GHI_GetFormattedTime(int8 *buffer);
typedef struct
{
        int32 seconds2:5;       // seconds divided by 2
        int32 minutes:6;
        int32 hours:5;
        int32 day:5;
        int32 month:4;
        int32 years_since_1980:7;
}FAT_Time_Structure;
int32 GHI_GetFATTimeStructure(int32 year, int32 month, int32 day, int32 hours,
int32   minutes, int32 seconds);
```

1.3.6 _INCLUDE_STORAGE_SECTOR_SUPPORT_

If defined the following functions are available:

```
int8 GHI_RegisterSD(void);
int8 GHI_RegisterUSBMassStorage(int8 device);
int8 GHI_ReadSector(int32 sectornum);
int8 GHI_WriteSector(int32 sectornum);
int8 GHI_SwitchDevice(int8 device);
```

1.3.7 _INCLUDE_HID_SUPPORT_

If defined the following functions are available:

```
int8 GHI_RegisterHID(int8 device);
int8 GHI_PrepareHIDReport(int8 device, int8 requested_size, int8 *
actual_reportsize);
```

1.3.8 _INCLUDE_PRINTER_SUPPORT_

If defined the following functions are available:

```
int8 GHI_RegisterPrinter(int8 device);
int8 GHI_ResetPrinter(int8 device);
int8 GHI_GetPrinterStatus(int8 device, int8 * status);
int8 GHI_PrinterPrint(int8 device, int8 size);
```

1.3.9 _INCLUDE_SERIAL_DEVICES_SUPPORT_

If defined the following functions are available:

```
int8 GHI_RegisterSerialDevice(int8 device, int8 type, int32 baudrate);
int8 GHI_SerialWrite(int8 device, int8 size);
int8 GHI_SerialRead(int8 device, int8 *sent_data_size);
```

1.3.10 _INCLUDE_EXTRA_COMMANDS_SUPPORT_

If defined the following functions and structures are available:

```
int8 GHI_UpdateFirmware(int8 device);

        typedef struct
{
        int16   vendor_id;
        int16   product_id;
        int8    number_of_interfaces;
        int8    interface1_class;
        int8    interface1_subclass;
        int8    interface1_protocol;
}DEVICE_INFO;

int8 GHI_ReadDeviceInfo(int8 device, DEVICE_INFO* info);
int8 GHI_SoftwareReset(void);
```

1.3.11 _INCLUDE_OLD_STORAGE_FUNCTIONS_WRAPPERS_SUPPORT_

If defined the following functions are available:

```
int8 GHI_BL_LoadFirmware(int8 drive);
int8 GHI_GetResults(void);
int8 GHI_SetARMBaudRate(int32 bauderate);
int8 GHI_AttachStorageMedia(int8 device, int8 deviceorder, int8 LUN);
int8 GHI_ReadSectorFromCurrentFileSystem(int32 sectornum);
int8 GHI_WriteSectorFromCurrnetFileDydtem(int32 sectornum);
int8 GHI_MountFileSystem(int8 filesystemorder, int8 device, int8 deviceorder);
int8 GHI_SwitchFileSystem(int8 filesystemorder);
```

```
int8 GHI_EnumurateUSBDevicetoRootHub(int8 usbport, int8 usbdevicehandle);
int8 GHI_ReleaseUSBDeviceHandle(int8 usbdevicehandle);
int8 GHI_RegisterMassStorageDevice(int8 usbdevicehandle, int8
                massstoragedevicehandle, int8 *lastLUN);
int8 InitializeFATMedia(int8 DriveLetter);
int8 SwitchToFATMedia(int8 DriveLetter);
int8 GHI_GetResultsQformat(void);
```

1.4. The Library Functions

1.4.1 The Return Value

Most of the functions returns an int8. This is a USBwiz Error Code, unless otherwise indicated. Error codes are listed in ErrorCode.h.
Any other return values will be explained in the functions documentation below.

1.4.2 General Functions

1.4.2.1 int16 GHI_GetLibraryVersion(void)

Gets the library version number.

return: 2 bytes number in BCD format.
ex: if the return value is 0x205 the version number is 2.05

1.4.2.2 int8 GHI_GetResult(void)

Gets any error codes sent by USBwiz. It must be called after using some functions. Refer to the functions documentation below.

1.4.2.3 int8 GHI_GetVersion(int8* major,int8 * BCDminor)

Obtaining the latest version is very useful and very important. Always keep USBwiz updated with latest firmware.

major: A pointer to return the major release number
BCDminor: This is a BCD number that represent the minor release.

For example: 0x12 = version x.12
0x32 = version x.32
0xDS= invalid value and will never happen!

1.4.2.4 int8 GHI_SetUSBwizBaudRate(int32 bauderate)

To set the baudrate:
1. Change USBwiz baudrate using this function.
2. If successful the user should change the interface baudrate.
3. Then GHI_GetResult() must be called to get any errors.

baudrate: The new baudrate

1.4.3 More functionality

These functions adds more functionality for the user.

Make sure _INCLUDE_EXTRA_COMMANDS_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use these functions.

1.4.3.1 int8 GHI_UpdateFirmware(int8 device)

This function updates the firmware from the specified media.

device: This can be SD_DEVICE, USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.3.2 int8 GHI_ReadDeviceInfo(int8 device, DEVICE_INFO* info)

This function reads a USB device information.
Note: Only the first interface is stored in the info structure.

device: This can be SD_DEVICE, USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
info: This is a pointer of a structure to hold the information of a USB device. It is defined as follows:

```
typedef struct
{
```

```
        int16   vendor_id;
        int16   product_id;
        int8    number_of_interfaces;
        int8    interface1_class;           // only get the first interface
        int8    interface1_subclass;
        int8    interface1_protocol;
}DEVICE_INFO;
```

1.4.3.3 int8 GHI_SoftwareReset(void)

Resets USBwiz firmware.

1.4.4 FAT File System Functions

Before you can handle files on a specified media, The FAT file system must be mounted first. This can be accomplished using GHI_MountFATFileSystem(). Then you can immediately use the other functions.

When you use another media and mount the file system on it, you will be working with that media by default. You can switch between the different medias using GHI_SwitchDevice().

Make sure _INCLUDE_FAT_SYSTEM_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.4.1 int8 GHI_MountFATFileSystem(int8 device)

Before manipulating files on the media, this function must be called

device: This can be SD_DEVICE, USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.4.2 int8 GHI_SwitchDevice(int8 device)

Tells USBwiz to switch between different medias when connected to different ports.
This function should be called every time you are switching to another media for getting information or any file handles. You only need to call it once if using the same media.

Note: After using GHI_MountFATFileSystem(int8 device);, you are by default using that device and don't need to switch to it!

device: This can be SD_DEVICE, USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.4.3 int8 GHI_ChangeDirectory(int8* filename)

Changes the current access to a pre-existed directory (folder) on the connected media.

filename: A null terminated string with the directory name.

1.4.4.4 int8 GHI_MakeDirectory(int8* filename)

Creates a new directory (folder) on the connected media. The name must be unique over the current directory span.

filename: A null terminated string with the directory name.

1.4.4.5 int8 GHI_OpenFile(int8 filehandle, int8 *filename, int8 openmode)

filehandle: USBwiz supports 4 file handles. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3.

filename: The name has to comply with 8.3 standard (ex:FILE1234.TXT)
openmode: Three modes for opening files:

- FILE_READ_MODE to open an existing file for read.
- FILE_WRITE_MODE to open new file for read and if the file exists it will delete it first
- FILE_APPEND_MODE to open a file that existed and append new data to it.

1.4.4.6 int8 GHI_SendWriteCommand(int8 filehandle, int32 desireddatasize)

Writing data to files happen in multiple stages.
1.Open a file for write or append
2.Send write request GHI_SendWriteCommand()

3.If previous command passed, send your data. You can't stop USBwiz at this point and you have to send all your data. This is why we recommend smaller data blocks.
4.When finished, USBwiz will return the results. Use GHI_GetReadAndWriteResult() to query the write results.

filehandle: USBwiz supports 4 file handles. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

desireddatasize: How many bytes are needed.

1.4.4.7 int8 GHI_SendReadCommand(int8 filehandle, int32 desireddatasize, int8 filler)

Similar to writing data, reading data from files happen in multiple stages.
1.Open a file for read
2. Send read request (GHI_SendReadCommand)
3. If previous command passed, USBwiz will return the data from the file. It will send "desireddatasize" bytes
4. When finished, USBwiz will return the results. Use GHI_GetReadAndWriteResult() to query the write results.
If USBwiz failed to read the data from the file it will send back the bytes as "filler". For example, if USBwiz said it can read 10 bytes but it was able to read only 8, it will send 8 bytes actual data from a file and 2 filler bytes.

filehandle: USBwiz supports 4 file handles. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

desireddatasize: How many bytes are needed.
filler: The filler can be any value of your choice.

1.4.4.8 int8 GHI_GetReadAndWriteResult(int32 * actualdatasize)

After sending a write or read request and finish processing the data, USBwiz will try its best to process all the data. In some case, the file write or read could fail, if the media is full for example. This function tells you how many bytes the read or write command was able to process.

After writing, some data maybe buffered inside USBwiz and you have to flush the file before 100% of the data exist in the card.

actualdatasize: A pointer to int32 that returns how many bytes were actually written/read. In most cases the returned value is the same as "datasize" requested for write or read.

1.4.4.9 int8 GHI_CloseFile(int8 filehande)

Flushes all buffered data to the media and closes the handle.

filehandle: USBwiz supports 4 file handles. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

1.4.4.10 int8 GHI_DeleteFile(int8 *filename)

Delete a file from the media. The file must exist and it shouldn't be opened by any handle. USBwiz doesn't check if the file is opened by a handle.

filename: A null terminated string with the file name.

1.4.5 The Extended FAT File System Functions

These functions can be used with regular FAT functions. They just add more functionality.

Make sure _INCLUDE_EXTENDED_FAT_SYSTEM_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.5.1 void GHI_StartMediaStatistics (void)

Depending on the media size, this function could take couple seconds to finish. This function will obtain how many sectors the media has and how many free ones. Must call GHI_GetResultMediaStatistics() to obtain these values after calling the first function.

1.4.5.2 int8 GHI_GetResultMediaStatistics(int32 * size, int32 * free)

After sending GHI_StartMediaStatistics, use this function to obtain the size of the media and the free space size.

size: A pointer to return the media size in sectors.
free: A pointer to return the free space in sectors.

1.4.5.3 void GHI_StartQformat(void)

Tells USBwiz to format the media. Formatting could take a few seconds to finish. When done GHI_GetResult() must be called which returns any errors.

1.4.5.4 int8 GHI_InitGetFile(void)

This function must be called once before using GetNextFile() to go to the first entry in the current directory.

1.4.5.5 int8 GHI_GetNextFile(int8 * file_name, int8 * file_ext, int8 *attributes, int32 * size)

In some applications, it would be useful to obtain a list of available files. Use GHI_InitGetFile() once and then keep pooling GHI_GetNextFile() until you have read all directories. You can access files in between GHI_GetNextFile() calls.

file_name: A non-null terminated string size 8.
file_ext: A non- null terminated string size 3 with file extension.
attributes: File Attributes are one byte Standard Attribute Structure in FAT system.
size : The file size in bytes.

Note: ERROR_END_OF_DIR_LIST value is returned if all of files are read.

1.4.5.6 int8 GHI_RemoveDirectory(int8* filename)

The directory should be empty before deleting it.

filename: Null terminated string with the file name.

1.4.5.7 int8 GHI_SendReadWriteFileCommand(int8 readhandle, int8 writehandle, int32 size)

A very effective way to read a file and write it to another file, even if the files are on different medias. If successful, USBwiz will return the results. Use GHI_GetReadAndWriteResult() to query the write results.

readhandle: A handle of the file that is open in read mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

writehandle: A handle of the file that is open in write/append mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

size: The size of write/read data in bytes

1.4.5.8 int8 GHI_SendShadowWriteTwoFiles(int8 firstfilehandle, int8 secondfilehandle, int32 desireddatasize)

Very effective way to write the same data to two files simultaneously. Even if the files are on different medias. If successful, the user should send all the data . Then GHI_GetShadowWriteTwoFileResults() must be called to query for the write results.

firstfilehandle: First file handle that is open for write. This value can be any valid file handle.
secondfilehandle: Second file handle that is open for write. This value can be any valid file handle.
desireddatasize: Size of data to write to the files.

1.4.5.9 int8 GHI_GetShadowWriteTwoFileResults(int8 *fresult1, int8 *fresult2, int32 *writtendata1, int32 *writtendata2)

After sending a write request and finish processing the data, USBwiz will try its best to process all the data. In some case, the file write could fail, if the media is full for example. This function tells you how many bytes the write command was able to process.

After writing, some data maybe buffered inside USBwiz and you have to flush the file before 100% of the data exist in the card.

fresult1: A pointer to hold error code for writing the first file.
fresult2: A pointer to hold error code for writing the second file.
writtendata1: A pointer to int32 that returns how many bytes were actually written to the first file. In most cases the returned value is the same as "datasize" requested for the write process.
writtendata2: A pointer to int32 that returns how many bytes were actually written to the second file. In most cases the returned value is the same as "datasize" requested for the write process.

1.4.5.10 int8 GHI_SendShadowWriteThreeFiles(int8 firstfilehandle, int8 secondfilehandle, int8 thirdfilehandle, int32 desireddatasize)

Very effective way to write the same data to three files simultaneously. Even if the files are on different medias. If successful, the user should send all the data . Then GHI_GetShadowWriteThreeFileResults() must be called to query for the write results.

firstfilehandle: First file handle that is open for write. This value can be any valid file handle.
secondfilehandle: Second file handle that is open for write. This value can be any valid file handle.
thirdfilehandle: Third file handle that is open for write. This value can be any valid file handle.
desireddatasize: Size of data to write to the files.

1.4.5.11 int8 GHI_GetShadowWriteThreeFileResults(int8 *fresult1, int8 *fresult2, int8 *fresult3, int32 *writtendata1, int32 *writtendata2, int32 *writtendata3)

After sending a write request and finish processing the data, USBwiz will try its best to process all the data. In some case, the file write could fail, if the media is full for example. This function tells you how many bytes the write command was able to process.
After writing, some data maybe buffered inside USBwiz and you have to flush the file before 100% of the data exist in the card.

fresult1: A pointer to hold error code for writing the first file.
fresult2: A pointer to hold error code for writing the second file.
fresult3: A pointer to hold error code for writing the third file.

writtendata1: A pointer to int32 that returns how many bytes were actually written to the first file. In most cases the returned value is the same as "datasize" requested for the write process.
writtendata2: A pointer to int32 that returns how many bytes were actually written to the second file. In most cases the returned value is the same as "datasize" requested for the write process.
writtendata3: A pointer to int32 that returns how many bytes were actually written to the third file. In most cases the returned value is the same as "datasize" requested for the write process.

1.4.5.12 int8 GHI_SeekFile(int8 filehandle, int32 newposition)

Sets the file pointer to new position. The file must be opened in read mode.

filehandle: A handle of the file that is open in read mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

newposition: new position.

1.4.5.13 int8 GHI_GetPointerPosition(int8 filehandle, int32 *sector, int16* positioninsector)

This function gets the current position in a file as a sector number and position inside the sector.

filehandle: A handle of the file that is open in read mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

sector: A pointer that holds the current sector number.
positioninsector: a pointer that holds the current position inside the sector.

1.4.5.14 int8 GHI_SplitFile(int8 sourcehandle, int8 desthandle1, int8 desthandle2, int32 splitposition, int32 * actualdestsize1, int32 * actualdestsize2)

Splits a file into 2 new files. The files can be opened on different drives or the same
drive. It requires one file to be open for read and 2 other files to be open for write. Files handles will be automatically closed after successful executing of the function.

sourcehandle: A valid file handle that is opened for read. This file will get split.
desthandle1: A valid file handle that is opened for write. This is the first destination file.
desthandle2: A valid file handle that is opened for write. This is the second destination file.
splitposition: The offset inside the source file at which the first file ends and the second file starts.
actualdestsize1: The actual size of data written to the first file.
actualdestsize2: The actual size of data written to the second file.

1.4.5.15 int8 GHI_FlushFile(int8 filehandle)

Flushes all buffered data inside USBwiz to the media.

filehandle: A handle of the file. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

1.4.5.16 int8 GHI_GetFileInfo(int8 *filename, int32 *size, int8 *attributes, int32 *TimeDate)

Use this function to find a specific file with a known name. This will also return found directories. Use Attributes to determine the type of found entry.

filename: A null terminated string with the file name.
size: This pointer stores the file size.
attributes: File Attributes are one byte Standard Attribute Structure in FAT system.
TimeDate: 32-Bit variable that holds the time as a standard FAT Time Structure.

1.4.5.17 int8 GHI_RenameFile(int8 *filename,int8 *newname)

This function rename an existing file to a new name.

filename: An null terminated string with the file name.
newname: The new name of the file.

1.4.5.18 int8 GHI_SetFileSize(int8 filehandle,int32 newsize)

This command sets file size to a certain size less than its actual size and omits the rest of the data. The file must be opened in Read-Mode.
The file handle will be automatically closed after successful executing of this function.

filehandle: A handle of the file that is open in read mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

newsize: The new size of the file.

1.4.6 Write/ Read wrappers

These are optional wrappers to do all the steps of reading or writing through a given buffer.

Make sure _INCLUDE_READWRITE_WRAPPERS_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.6.1 int8 GHI_ReadFile(int8 handle, int8 *buffer, int32 size, int8 filler, int32 *actualdatasize)

Reads data from a file into a given buffer.

handle: A handle of the file that is open in read mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

buffer: This should be provided by the user to hold the data and must be at least of size 'size'.
size: Size of data to read.
filler: The filler can be any value of your choice.
actualdatasize: A pointer to int32 that holds how many bytes were actually read. In most cases the returned value is the same as "size" in the read function.

1.4.6.2 int8 GHI_WriteFile(int8 handle, int8 *buffer, int32 size, int32*actualdatasize)

Writes data to a file from a given buffer.

handle: A handle of the file that is open in write mode. This value can be any of the following:

- FILE_HANDLE_0
- FILE_HANDLE_1
- FILE_HANDLE_2
- FILE_HANDLE_3

buffer: This should be provided by the user to hold the data and must be at least of size 'size'.
size: Size of data to write.
actualdatasize: A pointer to int32 that holds how many bytes were actually written. In most cases the returned value is the same as "size" in the write function.

1.4.7 Real Time Clock work functions

When you need to use RTC you only need to initialize it once. Then you can Set the time and read it.

Make sure _INCLUDE_TIME_WORK_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.7.1 int8 GHI_InitializeTime(int8 backup)

The Real Time Clock inside USBwiz is needed to set the correct dates on the saved files. There are 2 options for the RTC. It can be run using the same power source and oscillator as the processor core or it can run off 32Khz clock and a backup battery. Use this function when you need to switch between the 2 options.

backup: A flag that is when it is true, USBwiz will run the RTC on backup battery and when it is false it will run the RTC using the same power source and oscillator as core processor.

1.4.7.2 int8 GHI_SetTime(int32 time)

This function sets the time in USBwiz.

time: holds the time value to be set. The data is in the same format used by FAT file system and is defined in FAT Time structure.

1.4.7.3 int8 GHI_GetTime(int32 * time)

If you need to obtain the time, to save to a file for example, use this function.

time: A pointer to hold in the time value. The data is in the same format used by FAT file system and is defined in FAT Time structure.

1.4.7.4 int8 GHI_GetFormattedTime(int8 *buffer)

To get the time in a formatted way that is readable for the human eyes.

buffer: A null-terminated buffer that will hold the time. It must be at least 22 bytes.

Note: The formatted buffer uses zero's to pad.
Here is a typical formatted buffer: 05/12/2009 – 02:50:34

Note: This function is a little slower than GHI_GetTime(time).

1.4.7.5 int32 GetFATTimeStructure(int32 year, int32 month, int32 day, int32 hours, int32 minutes, int32 seconds)

This function takes the time as parameters and returns a 32-Bit variable that holds the FAT Time Structure.

The arguments just express the preferred time.

Note: You could use the FAT Time Structure directly. It's defined in the library for convenience, see below, but since some compilers don't support unaligned bits we implement this function to get the structure.

```
typedef struct
{
        int32 seconds2:5;       //  0…30  , seconds divided by 2
        int32 minutes:6;        //  0…59
        int32 hours:5;          //  0…23
        int32 day:5;            //  1…31
        int32 month:4;          //  1…12
        int32 years_since_1980:7;     // Years since 1980
}FAT_Time_Structure;
```

Note2: Unlike the structure, the function takes the regular seconds and years counts.

For further explanation you can refer to the examples section.

return: 32-Bit variable that holds the time as a FAT Time Structure.

1.4.8 Sector storage functions

These functions handle directly the sectors on the storage media and is intended for advanced users. No FAT System is required.
If used on a FAT formatted media, re-formatting the media might be need to access the files.
Before handling the sectors, registering the media is required. Then you can read or write directly to the sectors.

Make sure _INCLUDE_SECTOR_STORAGE_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.8.1 int8 GHI_RegisterSD(void)

Only used when handling sectors. This initializes an SD card before you can read or write to the sectors.

1.4.8.2 int8 GHI_RegisterUSBMassStorage(int8 device)

Only used when handling sectors. This initializes a USB Mass Storage before you can read or write to the sectors.

decive: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.8.3 int8 GHI_ReadSector(int32 sectornum)

After initializing the media, you can read the from it using this function. It reads a sector of size 512 bytes. After calling the function the user must read 512 bytes from USBwiz. Then GHI_GetResult() must be called to get any error information.

sectornum: The sector number to read the data from.

1.4.8.4 int8 GHI_WriteSector(int32 sectornum)

After initializing the media, you can write data to it using this function. It writes to sector of size 512 bytes. After calling the function the user must send 512 bytes to USBwiz. Then GHI_GetResult() must be called to get any error information.

sectornum: The sector number to write the data to.

1.4.8.5 int8 GHI_SwitchDevice(int8 device)

Tells USBwiz to switch between different medias when connected to different ports.
This function should be called every time you are switching to another media for getting information or any file handles. You only need to call it once if using the same media.

Note: After registering the media , you are by default using that device and don't need to switch to it!

device: This can be SD_DEVICE, USB_DEVICE_PORT_0 or
USB_DEVICE_PORT_1.

1.4.9 Using USB HID Devices (Mouse, keyboard, joystick...)

These functions will communicate with HID devices through USBwiz. Before using them, registering the device is required by calling GHI_RegisterHID(). Then you can get the data the HID sends by calling GHI_PrepareHIDReport() which gets how many bytes the device will send and asks USBwiz to get the data. USBwiz will send a hex string containing the data. The user must read the data string from USBwiz and call GHI_GetResult() which returns any error codes.

For example: If the report size is 4 bytes, USBwiz will sends a hex data string after calling GHI_PrepareHIDReport(). ex: 01F11203

Were 01 is the first byte, F1 is the second byte and so on.

Make sure _INCLUDE_HID_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.9.1 int8 GHI_RegisterHID(int8 device)

This function registers the HID.
* You only need to call this function once.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.9.2 int8 GHI_PrepareHIDReport(int8 device, int8 requested_size, int8 * actual_reportsize)

Tells USBwiz to get any data from the HID. You can request how many bytes you want or read all available data.
If successful, the user must read the sent data as mentioned earlier and then get any further errors by calling GHI_GetResult().

*If the return error code is HID_HAS_NO_DATA, then the HID has no data to send and you do not need to call any more functions.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
requested_size: This is how many bytes the user wants to read. Set this argument to 0 to read all available bytes.
actual_reportsize: A pointer that holds the size of data which will be read. If the requested_size is 0. This argument will equal how many bytes are available.

1.4.10 Using USB Printers

To communicate with a printer you need to register it first! And then you can send it any data, read it status or reset it.

Make sure _INCLUDE_PRINTER_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.10.1 int8 GHI_RegisterPrinter(int8 device)

Must be called before using the printer to initialize it.
* You only need to call it once.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.10.2 int8 GHI_ResetPrinter(int8 device)

Just resets the printer.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.

1.4.10.3 int8 GHI_GetPrinterStatus(int8 device, int8 * status)

Gets the status of the printer and stores it.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
status: A pointer that stores the printer status.

1.4.10.4 int8 GHI_PrinterPrint(int8 device, int8 size)

Tells USBwiz to send data of size “size” to the printer to print.
After calling this function, the user must send all the data to USBwiz and then call GHI_GetResult() to get any errors.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
size: Size of data to be sent. (max: 255 bytes)

1.4.11 Using Serial Devices

To Communicate with a Serial device, first the device should be registered. Then you can write or read from it.

Make sure_INCLUDE_SERIAL_DEVICES_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.11.1 int8 GHI_RegisterSerialDevice(int8 device, int8 type, int32 baudrate)

Registers a serial device. You only need to call this function once per device.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
type: The type of the serial device this can be any of the following:

- SERIAL_CDC_DEVICE
- SERIAL_SILABS_DEVICE
- SERIAL_PROLIFIC1_DEVICE
- SERIAL_PROLIFIC2_DEVICE
- SERIAL_FTDI_DEVICE

baudrate: Used to set the device's baudrate.

1.4.11.2 int8 GHI_SerialWrite(int8 device, int8 size)

After registering the device, this functions will tell USBwiz to send data of size "size"
to the serial device. Send all the data after using this function and then call GHI_GetResult() to get any error codes.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
size: The size of data to be sent. (Max 255 bytes)

1.4.11.3 int8 GHI_SerialRead(int8 device, int8 *sent_data_size)

After registering the device, this function will tell USBwiz to read any data if available from the Serial device. The user must read all the data and then call GHI_GetResult() to get any error codes.

device: This can be USB_DEVICE_PORT_0 or USB_DEVICE_PORT_1.
sent_data_size: Size of data USBwiz will send in bytes.

1.4.12 Old Functions

The following functions are included for people who already have them used in their software. And:
* They should not be used in any new projects.

* Updating the current product to use the new interface is highly recommended.

Make sure _INCLUDE_OLD_STORAGE_FUNCTIONS_WRAPPERS_SUPPORT_ is defined at the top of USBwiz_lib.h to be able to use the following functions.

1.4.12.1 int8 GHI_BL_LoadFirmware(int8 drive)

Updates the firmware from a media.

drive: Can be any of the following:
'A' or 'a' : For SD_DEVICE
'B' or 'b' : For USB_DEVICE_PORT_0
'C' or 'c' : For USB_DEVICE_PORT_1

1.4.12.2 int8 GHI_GetResults(void)

Same as GHI_GetResult().

1.4.12.3 int8 GHI_SetARMBaudRate(int32 bauderate)

Same as GHI_SetUSBwizBaudRate().

1.4.12.4 int8 GHI_AttachStorageMedia(int8 device, int8 deviceorder, int8 LUN)

Registers a storage device. Used to read and write to sectors.
* You don't need this if you are using FAT system.

device: This is 'C' or 'S' for SD cards and 'U' for USB mass storage.
deviceorder: Either 0 for USB port 0, or 1 for USB port 1.
LUN: Currently ignored.

1.4.12.5 int8 GHI_ReadSectorFromCurrentFileSystem(int32 sectornum)

Same as GHI_ReadSector().

1.4.12.6 int8 GHI_WriteSectorFromCurrnetFileDydtem(int32 sectornum)

Same as GHI_WriteSector().

1.4.12.7 int8 GHI_MountFileSystem(int8 filesystemorder, int8 device, int8 deviceorder)

Associates a file system number with a device.

filesystemorder: Can be 0, 1 or 2.
device: This is 'C' or 'S' for SD cards and 'U' for USB mass storage.
deviceorder: Either 0 for USB port 0, or 1 for USB port 1.

1.4.12.8 int8 GHI_SwitchFileSystem(int8 filesystemorder);

Switches the current file system.

filesystemorder: Can be 0, 1 or 2.

1.4.12.9 int8 GHI_EnumurateUSBDevicetoRootHub(int8 usbport, int8 usbdevicehandle);

You don't need to use this function anymore. It only returns ERROR_NO_ERROR.

1.4.12.10 int8 GHI_ReleaseUSBDeviceHandle(int8 usbdevicehandle)

You don't need to use this function anymore. It only returns ERROR_NO_ERROR.

1.4.12.11 int8 GHI_RegisterMassStorageDevice(int8 usbdevicehandle, int8 massstoragedevicehandle, int8 *lastLUN)

You don't need to use this function anymore. It only returns ERROR_NO_ERROR.

1.4.12.12 int8 InitializeFATMedia(int8 DriveLetter)

Initializes a specified device with FAT file system.

DriveLetter: Can be any of the following:
'A' or 'a' : For SD_DEVICE
'B' or 'b' : For USB_DEVICE_PORT_0
'C' or 'c' : For USB_DEVICE_PORT_1

1.4.12.13 int8 SwitchToFATMedia(int8 DriveLetter)

Switches the current device to another one.

DriveLetter: Can be any of the following:
'A' or 'a' : For SD_DEVICE
'B' or 'b' : For USB_DEVICE_PORT_0
'C' or 'c' : For USB_DEVICE_PORT_1

1.4.12.14 int8 GHI_GetResultsQformat(void)

It behaves like GHI_GetResult().

2. Driver Functions

USBwiz_lib is written to work on any processor but there will be a need to implement a few driver functions that will help USBwiz_lib in using your processor. In our example code, the driver functions fro UART, SPI and I2C for PIC are provided. We also included GHI_inter_user.c as a template to add your own driver functions. The library should compile for any compiler and architecture but the library doesn't know anything about your processor nor the interface you want to use. For example, When the library wants to send a byte to USBwiz, it will call GHI_PutC function and then it is the use responsibility to implement the function.

void GHI_Sleep(int16 ms);

In some situations, USBwiz_lib will require some delay on some task. All delays happen through GHI_Sleep. This function will return after x milliseconds. This function doesn't have to be accurate at all and can be implemented using simple loops. You can test if your function is working right by simply toggling an LED and use 500 ms for delay.

```
Void BlinEveryOneSecond(void)
{
        LED=!LED; // toggle an LED
        // 500 ms low + 500 ms high = switch on every 1 second
        GHI_Sleep(500);
}
```

ms: how many millisecond to loop

int8 GHI_OpenInterface(void);

The library doesn't need this function but you will use it at power up to initialize the interface. The interface can be UART, SPI or I2C.

Note when we say interface from now on we will be referring to UART, SPI or I2C.

Return: 0 if okay or error code otherwise.

int8 GHI_CloseInterface(void);

Will close the interface. In most cases this won't be necessary.

Return: 0 if okay or error code otherwise.

int8 GHI_SetBaudRate(int32 baud);

This is needed only in the case of the interface is UART. It is a good practice to switch the baud rate to a faster one. USBwiz powers up with 9600 baud. This is very slow to what you can set the baud rate to.

baud: The baud is the rate of how many bit per seconds will be transfered.

Return: 0 if okay or error code otherwise.

int8 GHI_GetC(void);

When there is data ready in the interface receive buffer, GHI_GetC will fetch it and return it immediately. If there is no data ready, GHI_GetC will wait for data to be available. You can some timeout and return 0 if there is no data for a long time to prevent code lockups.

The implementation can simply pool the interface or it can read a FIFO that is filled by the interface's interrupt routine.

Return: a character (byte) from the interface when ready.

void GHI_PutC(int8 ch);

If the interface is ready to transmit data, GHI_PutC will place a byte in it's transmit buffer. GHI_PutC will wait for the interface to become ready before it sends anything.

ch: A character (byte) to be transferred to the interface when ready.

void GHI_PutS(int8 * str);

Very similar to GHI_PutS but GHI_PutS will traqnsfer a null terminated string to the interface. Be careful when you use a processor that has RAM and ROM pointers, a PIC for example. GHI_PutS will work with ram pointers only. Most processor use the same pointer for RAM and ROM, including your PC's processor.

str: a null terminated string to be transmitted to the interface.

Null terminated means that the least byte of the string must be zero.

int8 GHI_DataIsReady(void);

It is a good practice to check that there is some data in the receive buffer before using GHI_GetC so we will not lockup the code.

Return: 1 if data is ready and 0 if there is no data ready.

void GHI_ToggleWakePin(void);

USBwiz_lib doesn't know how to toggle the wake pin on your system. Implement this function to do so if you need to use the sleep mode.

3. Examples

Note1: The following code require proper set-up in the program
Note2: For the sake of simplicity, the code doesn't check for error codes. It's highly recommended that you do.

3.1.Simple file write/read process

```
//…….

int8 file_name[] = "FILE.EXT", buffer[3];
int32 size;

// Using Mass storage port 0
GHI_MountFATFileSystem(USB_DEVICE_PORT_0);

// Open a file for write using the 0 handle
GHI_OpenFile(FILE_HANDLE_0, file_name, FILE_WRITE_MODE);

// write 2 bytes to handle 0
GHI_SendWriteCommand(FILE_HANDLE_0, 2);

// sending 2 bytes
GHI_PutC('V');
GHI_PutC('A');

// after this the size should be 2 in most cases
GHI_GetReadAndWriteResults(&size);

// writing done, let's read it
// release the handle 0
GHI_CloseFile(FILE_HANDLE_0);

// re-open the same file for read
GHI_OpenFile(FILE_HANDLE_0, file_name, FILE_READ_MODE);
```

```
// read 3 bytes using '*' as a filler
GHI_SendReadCommand(FILE_HANDLE_0, 3, '*');

// read three bytes
GHI_GetC(&buffer[0]);
GHI_GetC(&buffer[1]);
GHI_GetC(&buffer[2]);
/*
/*  Now the buffer should look like this:
/*  buffer[0] = 'V'
/*  buffer[1] = 'A'
/*  buffer[2] = '*'
*/
// after this the size should be 2 indicating reading 2 bytes successfully
GHI_GetReadAndWriteResults(&size);

// release the handle
GHI_CloseFile(FILE_HANDLE_0);
```

3.2.Read/Write Wrappers

```
// ....

int8 buffer[3], file_name[] = "FILE.TXT";
int32 size;

// Using Mass storage port 0
GHI_MountFATFileSystem(USB_DEVICE_PORT_0);

// open a file
GHI_OpenFile(FILE_HANDLE_0, file_name, FILE_WRITE_MODE);

// file buffer
buffer[0] = 'A'; buffer[1] = 'B';
```

```
// write 2 bytes
GHI_WriteFile(FILE_HANDLE_0, buffer, 2, &size);
// now size should be 2 "actual written bytes"

// close file
GHI_CloseFile(FILE_HANDLE_0);

// re-open for write
GHI_OpenFile(FILE_HANDLE_0, file_name, FILE_READ_MODE);

// read 3 bytes using '#' as a filler
GHI_ReadFile(FILE_HANDLE_0, buffer, 3, '#', &size);
// now size should be 2 "actual read bytes"

/* The buffer should be now
/* buffer[0] == 'A';
/* buffer[1] == 'B';
/* buffer[2] == '#';
*/

// close file
GHI_CloseFile(FILE_HANDLE_0);
```

3.3.Enumerating Files

```
// …

int8 file_name[16], file_ext[4], attributes;
int32 size;

// assuming using an SD card
GHI_MountFATFileSystem(SD_DEVICE);

// Start the list
```

```
GHI_InitGetFile();

// loop until the end
While(  GHI_GetNextFile(file_name, file_ext, &attribute, &size) !=

        ERROR_END_OF_DIR_LIST   )
{
          // just print the file name
        printf("%s\n", file_name);
}
```

3.4.Read and write simultaneously

```
// ...

int8 lread_file[] = "READFROM.TXT", write_file[] = "WRITETO.TXT";
int32 size;

// assuming using usb port 0 to read from and usb port 1 to write to

GHI_MountFATFileSystem(USB_DEVICE_PORT_0);
GHI_MountFATFileSystem(USB_DEVICE_PORT_1);

// switch to the first file system on USB port 0
GHI_SwitchDevice(USB_DEVICE_PORT_0);

// assuming this file already exist
GHI_OpenFile(FILE_HANDLE_0, read_file, FILE_READ_MODE);

// switch to the second file system on USB port 1
GHI_SwitchDevice(USB_DEVICE_PORT_1);

// open for write
GHI_OpenFile(FILE_HANDLE_1, write_file, FILE_WRITE_MODE);
```

```
// read and write 10 bytes
GHI_SendReadWriteFileCommand(FILE_HANDLE_0, FILE_HANDLE_1, 10);

// get results. After this call, size should be 10
GHI_GetReadAndWriteResult(&size);

// end
GHI_CloseFile(FILE_HANDLE_0);
GHI_CloseFile(FILE_HANDLE_1);
```

Note: You could mount the first file system and get a file handle, then mount the second file system and get the other file handle without the need to switch devices.

3.5.Write to multiple files simultaneously " Shadow writing"

```
// ...

int8 file1_results, file2_results, file1[] = "FILE1.EXT", file2[] = "FILE2.EXT";
int32 actual_file1_size, actual_file2_size;

// first mount the file system on SD_CARD
GHI_MountFATFileSystem(SD_DEVICE);

// open two files for writing

GHI_OpenFile(FILE_HANDLE_0, file1, FILE_WRITE_MODE);
GHI_OpenFile(FILE_HANDLE_1, file2, FILE_WRITE_MODE);

// write 2 bytes
GHI_SendShadowWriteTwoFiles(FILE_HANDLE_0, FILE_HANDLE_1, 2);

// Send data
GHI_PutC('V');
```

```
GHI_PutC('P');

// get results
GHI_GetShadowWriteTwoFileResults(&file1_results, &file2_results,
                                                    &actual_file1_size,
&actual_file2_size);
/*
if successful :  file1_results ==  file2_results == ERROR_NO_ERROR;
                 actual_file1_size == actual_file1_size == 2;
*/

// close handles

GHI_CloseFile(FILE_HANDLE_0);
GHI_CloseFile(FILE_HANDLE_1);
```

3.6.Real Time Clock

NOTE: The compiler should support unaligned bits in order to use the FAT Time Structure directly. Please consult your compiler manual.
You could however use the structure indirectly using this function, see examples below!
int32 GetFATTimeStructure(int32 year, int32 month, int32 day, int32 hours, int32 minutes, int32 seconds);

3.6.1 Using the FAT Time Structure directly

```
// …

FAT_Time_Structure  time;
int32  time_var;
int8 buffer[32];

// set the time to FEB.4.2007 13:10:10
time.seconds2 = 5;                    // this should be the seconds divided by 2
```

```
time.minutes = 10;
time.hours = 13;
time.day = 4;
time.month = 2;
time.years_since_1980 = 2007-1980;

GHI_InitializeTime(0);              // initialize with '0' "NO backup battery",
use '1' if any
time_var = *(int32*)(&time);        // cast the structure to int32
GHI_SetTime(time_var);              // set the time

// Exactly after 2 years and 5 minutes

GHI_GetTime(&time_var);                         // read the time
time = *( FAT_Time_Structure  *)(&time_var)     // cast it to the structure

/*
/*  now the structure should look like this
/*  time.seconds2 □ 5               // it's seconds divided by 2
/*  time.minutes □ 15
/*  time.hours □ 13
/*  time.day □ 4
/*  time.month □ 2
/*  time.years_since_1980 □ 29
*/
// you can use this function to display the time, although it is a little slower than
the above one
GHI_GetFormattedTime(buffer);                   // get the time
printf("%s\n", buffer);                         // print it

/*
/* the output should look like this:
/* 02/04/2007 – 13:15:00
*/
```

3.6.2 Another way to use the FAT Time Structure

```
// …

int32 seconds, minutes, hours, day, month, years;
int32 time;
int8 buffer[32];

// set the time parameters
seconds = 40;
minutes = 30;
hours = 4;
day = 11;
month = 8;
years = 2006;

// Set the time into 32bit structure
time = GetFATTimeStructure(years, month, day, hours, minutes, seconds);
GHI_InitTime(0);              // initialize with '0' "NO backup battery", use '1'  if
any
GHI_SetTime(time);            // set the time

// After 1 hour, 20 minutes
GHI_GetFormattedTime(buffer);          // read the time, you also use
                                       // GHI_GetTime(&time);
        printf("%s\n", buffer);        // print it

/*
/* the output should look like this:
/* 08/11/2006 – 05:50:40
*/
```

3.7.HIDs (Mouse, keyboard, ...)

```
// ...
```

```
in8 report[16], report_size, index;

// Connecting a device (assuming a mouse) to port 0
GHI_RegisterHID(USB_DEVICE_PORT_0);

// the user moved the mouse to the right and clicked the right button
// let's read any data
GHI_PrepareHIDReport(USB_DEVICE_PORT_0, &report_size);

// usually for mouses the report_size is 4 bytes, so report_size == 4

// the report is a hex string simillar to 0100FF02
// this was 4 bytes
// read the report into the buffer
for(index = 0; index < (report_size * 2); index++)
{
        report[index] = GHI_GetC();
}

// get any errors
GHI_GetResult();

/* now the buffer might look like this
/* "the user moved the mouse to the right and clicked the right button"
/* buffer bits     ->  [0][1]  [2][3]   [4][5]  [6][7]
/* buffer values ->  [0][2]  [0][D]  [0][0]  [0][0]
*/
```

Note: For information on decoding the report, check the HID tutorial at www.ghielectronics.com

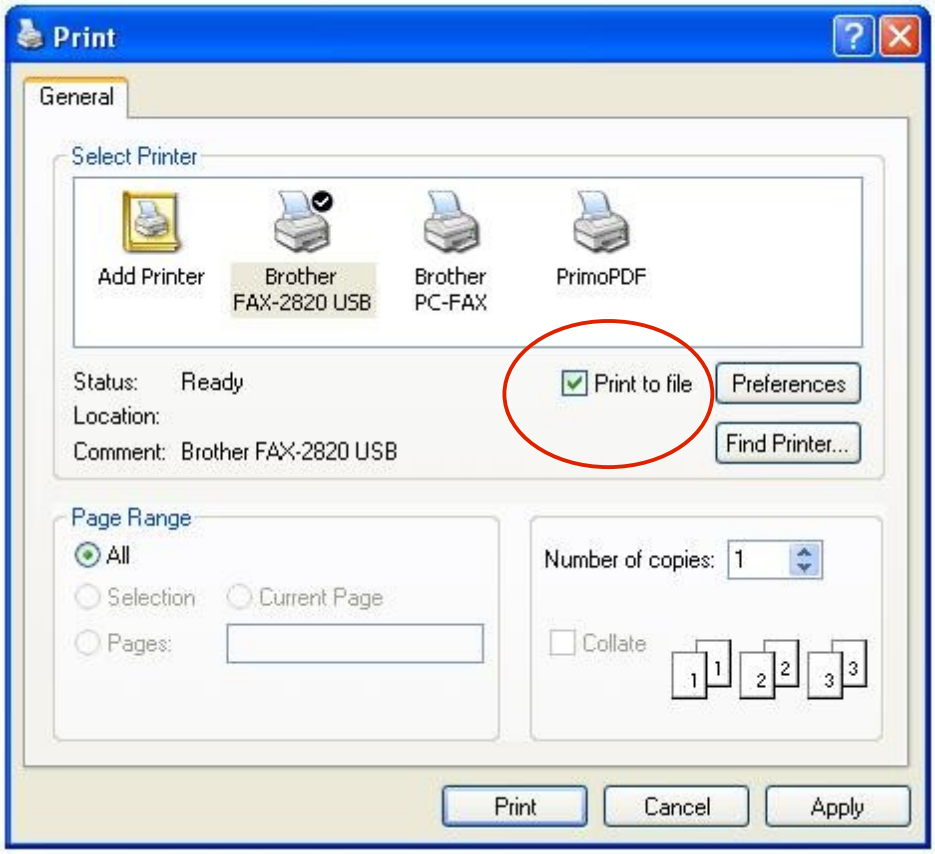
3.8.Printers

There are printer commands to talk to a specific printer and USBwiz provides the functionality to communicate with a printer. So we need to send the printer some commands through USBwiz to print!

An easy way to test a printer without the knowledge of its commands is to use "print to file" and send the file to USBwiz. To print to a file, you can follow these steps:

1. Create a txt file and type a string in it:
   "new.txt" has "This was printed using USBwiz!!"

2. Go to file -> print.
3. Select your printer and check print to a file.



4. Hit print and for the output file name type “TEST.PRT” and hit OK.
5. Now we got the file.

// ...

int8 buffer[255], actual_size, error, index;

int8 file_name = “TEST.PRT”;

// Register the printer to port 1

```
GHI_RegisterPrinter(USB_DEVICE_PORT_1);

// Reset the printer
GHI_ResetPrinter( USB_DEVICE_PORT_1 );

// Assume the media that has the file is connected to port 0
// let's open the file
GHI_MountFATFileSystem(USB_DEVICE_PORT_0);
GHI_OpenFile(FILE_HANDLE_0, file_name, FILE_READ_MODE);

// Now read the file and send to printer
// we send every 255(0xFF) bytes at a time because it is maximum you can send
at
// a time

while(1)
{
        error = GHI_ReadFile(FILE_HANDLE_0, buffer, 0xFF, '*', &actual_size)
        if(error)
                return error;

        // we read all the file
        if(actual_size == 0)
                break;

        // since there is no errors, the actual_size should be 0xFF unless we
reached
        // end of file.
        // Print
        GHI_PrinterPrint(USB_DEVICE_PORT_1, actual_size);

        // send the data
        for(index = 0; index < actual_size; index++)
                GHI_PutC(buffer[index]);
```

```
        // get any errors " the user should check"
        error = GHI_GetResult();
}

// We are done and the printer should be printing now.
```

3.9.Serial Devices

```
//...

int8 size, i, buffer[255];

// Assume a FTDI device is connected to usb port 0
// And we want to set the baudrate to 9600.

// First must initialize
GHI_RegisterSerialDevice(USB_DEVICE_PORT_0, SERIAL_FTDI_DEVICE, 9600);

// writing data. Just send two bytes.
GHI_SerialWrite(USB_DEVICE_PORT_0, 2);

// send the data
GHI_PutC('A');
GHI_PutC('B');

// Get any errors
GHI_GetResult();

// Done writing

// Assuming the deviced respoded with 'C' then 'D'

// Let's read
 GHI_SerialRead(USB_DEVICE_PORT_0, &size);
```

```
// now the size should equal 2

// read the data
for(i = 0; i < size; i++)
        buffer[i] = GHI_GetC();

// Get any errors
GHI_GetResult();

/* Now the buffer should look like:
/* buffer[0] --- 'C'
/* buffer[1] --- 'D'
*/
```

Appendix A:

FAT Time Structure

Time and Date structure is a DWORD Standard structure in FAT system.

Bits(s)	Field	Description
31..25	Year1980	Years since 1980
24..21	Month	1..12
20..16	Day	1..31
15..11	Hour	0..23
10..5	Minute	0..59
4..0	Second2	Seconds divided by 2 (0..30)

FAT Attribute Structure

File Attributes are one byte Standard Attribute Structure in FAT system.

7	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Reserved		*Archive*	*Folder*	*Volume ID*	*System*	*Hidden*	*Read Only*

Appendix B: Error Codes

Description	Value
No Error	0x00
ERROR_READ_SECTOR	0x01
ERROR_WRITE_SECTOR	0x02
ERROR_ERASE_SECTOR	0x03
ERROR_MBR_SIGNATURE_MISSMATCH	0x11
ERROR_BS_SIGNATURE_MISSMATCH	0x12
ERROR_SECTOR_SIZE_NOT_512	0x13
ERROR_FSINFO_SIGNATURE_MISSMATCH	0x14
ERROR_CLUSTER_OVER_RANGE	0x21
ERROR_CLUSTER_UNDER_RANGE	0x22
ERROR_NEXT_CLUSTER_VALUE_OVER_RANGE	0x23
ERROR_NEXT_CLUSTER_VALUE_UNDER_RANGE	0x24
ERROR_NO_FREE_CLUSTERS	0x25
ERROR_FILE_NAME_FORBIDDEN_CHAR	0x31
ERROR_FILE_NAME_DIR_NAME_OVER_8	0x32
ERROR_FILE_NAME_DIR_EXTENSION_OVER_3	0x33
ERROR_FILE_NAME_FIRST_CHAR_ZERO	0x34
ERROR_MEDIA_FULL	0x35
DIR_ENT_FOUND	0x40
DIR_ENT_NOT_FOUND	0x41
ERROR_FOLDER_IS_CORRUPTED_FIRST_CLUSTER	0x42
ERROR_FOLDER_IS_CORRUPTED_DIR_DOT_NOT_FOUND	0x43
ERROR_FOLDER_IS_CORRUPTED_DIR_DOTDOT_NOT_FOUND	0x44
ERROR_ROOT_DIRECTORY_IS_FULL	0x45
ERROR_OPEN_FOLDER_FILE	0x46
ERROR_WRTIE_TO_READ_MODE_FILE	0x47
ERROR_SEEK_REQUIER_READ_MODE	0x48
ERROR_INVALID_SEEK_POINTER	0x49
ERROR_FOLDER_NOT_EMPTY	0x4A
ERROR_IS_NOT_FOLDER	0x4B
ERROR_READ_MODE_REQUIRED	0x4C
ERROR_END_OF_DIR_LIST	0x4D
ERROR_FILE_PARAMETERS	0x4E
ERROR_INVALID_HANDLE	0x4F
ERROR_EOF	0x50
ERROR_NEW_SIZE_ZERO	0x51
ERROR_HCD_CHIP_NOT_FOUND	0x60
ERROR_HCD_PTD_COMP_CRC	0x61
ERROR_HCD_PTD_COMP_BIT_STUFFING	0x62
ERROR_HCD_PTD_COMP_DATA_TOGGLE	0x63
ERROR_HCD_PTD_COMP_STALL	0x64
ERROR_HCD_PTD_COMP_DEVICE_NO_RESPOND	0x65
ERROR_HCD_PTD_COMP_PID_CHECK_FAIL	0x66
ERROR_HCD_PTD_COMP_UNEXPECTED_PID	0x67
ERROR_HCD_PTD_COMP_DATA_OVERRUN	0x68
ERROR_HCD_PTD_COMP_DATA_UNDERRUN	0x69
ERROR_HCD_PTD_COMP_RESERVED1	0x6A
ERROR_HCD_PTD_COMP_RESERVED2	0x6B

Description	Value
ERROR_HCD_PTD_COMP_BUFFER_OVERRUN	0x6C
ERROR_HCD_PTD_COMP_BUFFER_UNDERRUN	0x6D
ERROR_HCD_INALID_CHIP_ID	0x6E
ERROR_HCD_USB_DEVICE_NOT_CONNECTED	0x6F
ERROR_PORT_NMBER_NOT_AVILABLE	0x70
ERROR_USBD_NO_ENOUGH_PIPES	0x71
ERROR_USBD_HANDLE_INUSE	0x72
ERROR_USBD_INCORRECT_DESCRIPTOR	0x73
ERROR_USBD_NONCONTROL_TRANSFER_FUNCTION	0x74
ERROR_USBD_DATA_SIZE_IS_BIG_FOR_ENDPOINT	0x75
ERROR_USBD_TIMEOUT	0x76
ERROR_USBD_CONTROL_TRANSFER_REQUIERED	0x77
ERROR_USBD_NACK	0x78
ERROR_USBD_HANDLE_CORRUPTED	0x79
ERROR_USBD_DESCRIPTOR_CORRUPTED	0x7A
ERROR_DESCRIPTOR_NOT_FOUND	0x7B
ERROR_USB_HUB)NOT_FOUND	0x7C
ERROR_BOMS_CSW_COMMAND_FAILD	0x81
ERROR_BOMS_CSW_STATUS_PHASE_ERROR	0x82
ERROR_BOMS_WORNG_LUN_NUMBER	0x83
ERROR_BOMS_WORNG_CSW_SIGNATURE	0x84
ERROR_BOMS_WORNG_TAG_MISSMATCHED	0x85
ERROR_USB_MASS_STORAGE_DEVICE_NOT_READY	0x90
ERROR_USB_MASSSTORAGE_PROTOCOL_NOT_SUPPORTED	0x91
ERROR_USB_MASSSTORAGE_SUBCLASS_NOT_SUPPORTED	0x92
ERROR_SPC_INVALID_SENSE	0x93
ERROR_SPC_NO_ASC_ASCQ	0x94
ERROR_USB_MASSSTORAGE_NOT_FOUND	0x95
ERROR_COMMANDER_BAD_COMMAND	0xA1
ERROR_COMMANDER_STR_LEN_TOO_LONG	0xA2
ERROR_COMMANDER_NAME_NOT_VALID	0xA3
ERROR_COMMANDER_NUMBER_INVALID	0xA4
ERROR_COMMANDER_WRITE_PARTIAL_FAILURE	0xA5
ERROR_COMMANDER_UNKNOWN_MEDIA_LETTER	0xA6
ERROR_COMMANDER_FAILED_TO_OPEN_MEDIA	0xA7
ERROR_COMMANDER_INCORRECT_CMD_PARAMETER	0xA8
ERROR_USB_COMMANDER_CONFIG_NOT_LOADED	0xA9
ERROR_CHECK_SUM	0xAA
ERROR_FILE_SYSTEM_NOT_MOUNTED	0xAB
ERROR_FTDI_DEVICE_NOT_REGISTERED	0xB1
ERROR_INCORRECT_VENDORID	0xB2
ERROR_INCORRECT_PRODUCTID	0xB3
ERROR_PRINTER_NOT_REGISTERED	0xB4
HID_HAS_NO_DATA	0xB5(not error)
ERROR_COMMANDER_UNKNOWN_ERROR	0xFD

