Vhand 2.0 SDK Manual

DGTech Engineering Solutions
www.dg-tech.it

Release 1.1 October 2007

Table of Contents

Introduction

Developing custom applications involving the use of the DG5 Vhand 2.0 is really immediate. The provided software libraries give to the developers an integrated systems which permits the use of a multi data glove environment. Any data glove has got its own data structure which can be easily addressed, in order to develop different and sophisticated applications.

The SDK consists of 4 files, that must be included in your C++ project:

VHandManager.lib
VhandManager.dll
Project.h
DataGlove.h

These files are intended for a use inside Windows 98/2000/Xp and Vista environments. For different operative systems, please contact us at info@dg-tech.it.

By the use of the provided library the access to a data glove is reduced to few lines of code, here it is an example that creates two different data glove and start sampling them:

```
#include “Project.h”
#include “DataGlove.h”

Project *proj ;
int gloveid1, gloveid2;

int main()
{
..... software initialization

proj = new Project();
gloveid1 = proj->AddDataGlove(“dataglove1”,1,0);
gloveid2 = proj->AddDataGlove(“dataglove1”,2,1);

proj->StartSampling(1);
proj->StartSampling(2);
}

void ShowValues()
{
// this function extract values from the data glove structure and plot them on the screen
// it is pseudo code, so the plotting routine should be changed with the opportune ones

DataGlove *dg1 = proj->GetDataGlove(gloveid1);
DataGlove *dg2 = proj->GetDataGlove(gloveid2);

// show the finger value of dataglove1
Plot(“%f %f %f %f %f”,dg1->f[0],dg1->f[1],...)
// show the finger value of dataglove2
Plot(“%f %f %f %f %f”,dg2->f[0],dg2->f[1],...)
...
}
```

Let us analize more deeply the code already shown:

Proj = new Project() initialize the manager of the gloves. The function AddDataGlove(name,

comport, type) creates a new data glove which name is “name”, it is connected to the serial port “comport” and it is of type “type” (0 means right data glove, 1 means left data glove).

The function StartSampling(gloveid) opens the specified COM port and start sampling it, getting the data that arrives from the glove and storing them inside the DataGlove structure. In order to stop the sampling one can use the project function StopSampling(gloveid).

The DataGlove structure guarantees the access to all the needed information coming from the data glove:

```
class DataGlove
{
public:
        DataGlove();
        ~DataGlove(void);


        // data glove characteristics
        CString label;
        int id              ;
        int comport ;
        int type ;

        // data glove values
        int   fmin[5],fmax[5];      // maximum and minimum finger value (raw 0..1023)
        float f[5];                 // finger values (0.0 ... 100.0%)
        float roll,pitch ;          // rotation angles in degree (-90.. +90)
        float ax,ay,az    ;         // acceleration values in g (-2...+2);


        // don't touch these values!!!!
        bool NewDataAvailable ;
        bool Sampling ;
        bool PortOpened ;
        bool SelfCalibration;
        int counter_buffer ;
        int buffersize;    // dimensione buffer dei finger
        int buff[2000];
        int buffaccsize ; // dimensione buffer delle accelerazioni
        int buffacc[2000];

        int ResetValue(void);
        int ElaborateValues(void);
        int AddPackage(unsigned char * buffer);
};
```

The developers can freely access to the following data glove values:

f[i] reports the finger status in percentage (0.0 to 100.0)
If the glove is of right type the f[0] is the thumb while f[4] represents the small finger, viceversa if the glove is of left type then f[0] represents the small finger and f[4] the thumb.

Roll and pitch fields report the hand inclinations in degrees, from -90 to +90, while ax,ay, anz az represent the instantaneous accelerations of the hand, whose are useful to develop sophisticated gesture recognition algorithms.
The other values represents the internal status of the glove and they should not be manipulated.

Class Project

Include: Project.h

```
class Project
{
public:
        Project(void);
        ~Project(void);

        void AddDataGlove(const char *label, int comport, int type);
        bool RemoveGlove(int gloveid);
        DataGlove *GetDataGlove(int id);
        int SetGloveBufferSize(int size_finger, int size_acc, int gloveid);


        void SaveProject(void);
        int LoadProject(void);
        int GetNumDataGlove(void);
        int CloseProject(void);

        int StartSampling(int gloveid);
        int StopSampling(int gloveid);
};
```

The Project class permits the access and the storage of the different data gloves.

The function exposes by the calss can be divided in three sections:

- **global project manipulation**
- **data glove related functions**
- **sampling functions**

Global project manipulation

This section is composed by four functions:

int LoadProject();
int SaveProject();
int CloseProject();
int GetNumDataGlove();

---

int LoadProject()

Parameters: none
Return value: 1 if the project has been succesfully loaded, 0 otherwise

Description: this function load the project file, which must be previously saved by the function SaveProject(). The project file is stored in C:\\windows\vhand.ini, and it is automatically created any time the SaveProject() function is called. The same file is created by the GloveSetup software, this software can be used in order to initialize the project.

---

int SaveProject()

Parameters: none
Return value: 1 if the project has been succesfully saved, 0 otherwise

Description: this function save all the data glove parameters stored in project, the information are stored in C:\\windows\vhand.ini.

---

void CloseProject()

Parameters: none
Return value: none

Description: free the memory, This funciton should be called when the application terminates.

---

int GetNumDataGlove()

Parameters: none
Return value: the number of data glove managed by the project

Description: this function returns the number of dataglove structure managed by the Project class, it should be called after the LoadProject() function or after the AddDataGlove(...) function.

Data glove related functions

int AddDataGlove(const char *label, int comport, int type)

Parameters:

label : mnemonic name of the data glove
comport : serial port connected to the glove
type: 0 = right data glove, 1= left data glove

Return value: -1 if the glove cannot be added, the unique id of the glove otherwise. The id is automatically chosen by Project.

Description: this function adds a dataglove structure to the project. The dataglove name, the associated serial port and the type of the glove must be provided.

---

int RemoveDataGlove(int gloveid)

Parameters:

gloveid = the unique dataglove identifier

Return value: -1 if the glove cannot be added, the unique id of the glove otherwise

Description: this function remove the data glove with id = gloveid from the Project data glove array..

Int SetGloveBufferSize(int size_finger, int size_acc, int gloveid)

Parameters:

gloveid = the unique dataglove identifier

Return value: -1 if the glove cannot be added, the unique id of the glove otherwise

Description: this function modifies the size of the buffer of the data glove associated to the id = gloveid. The data coming from the data glove are stored inside a buffer and the values exposes to the user represents a mean of the last size_finger (or size_acc) values. Size_finger modifies the size of the buffer that stores the data representing the finger position, while size_acc represents the size of the buffer that stores the information coming from the accelerometer.
Increasing the buffers will stabilize the values and decrease the promptness of the system, viceversa decreasing the buffer will increase the promptness of the measures decreasing the stability.

DataGlove *GetDataGlove(int gloveid)

Parameters:

gloveid = the unique dataglove identifier

Return value: NULL is there is no glove with the associated id = gloveid, otherwise a pointer to the DataGlove class;

Description: this function returns a pointer to the DataGlove class.

Sampling Function

int StartSampling(int gloveid)

Parameters:
gloveid = the unique dataglove identifier

Return value: -1 if the glove cannot be found, 1 otherwise

Description: this function starts the data glove sampling process, a serial communication is initialized and the packages coming from the data glove are processed, the DataGlove structure is updated anytime a new package arrives.

Int StopSampling(int gloveid)

Parameters:
gloveid = the unique dataglove identifier

Return value: -1 if the glove cannot be found, 1 otherwise

Description: this function terminates the sampling process associated to the glove with id = gloveid.

Class DataGlove

Include: DataGlove.h

```
class DataGlove
{
public:
	DataGlove();
	~DataGlove(void);

	CString label;
	int id			;
	int comport ;
	int type    ;

	int   fmin[5],fmax[5];
	float f[5];
	float roll,pitch ;
	float ax,ay,az    ;


	bool NewDataAvailable ;
	bool Sampling ;
	bool PortOpened ;
	bool SelfCalibration;
	int counter_buffer ;
	int buffersize;   // dimensione buffer dei finger
	int buff[2000];
	int buffaccsize ; // dimensione buffer delle accelerazioni
	int buffacc[2000];

	int ResetValue(void);
	int ElaborateValues(void);
	int AddPackage(unsigned char * buffer);
};
```

Field Description:

label: the name associted to the glove ;

id = the unique parameter that identifies the dataglove inside the array ;

comport = the RS232 serial port used by the glove to communicate to the PC;

type = the glove type, 0 = right data glove, 1 = left data glove;

fmin[i], fmax[i] = the minimum and maximum values reached by the finger sensors, these value are updated by the GloveSetup software by selecting the Start Calibration function ;

f[i] = the value, in percentage, of the finger. A value of 100.0 represent the maximum finger flexion, 0.0 represent no flexion on the finger. If the glove is of right type then f[0] is the thumb finger while f[4] is the small one, if the glove is of left type then f[0] represents the small finger while f[4] is the thumb;

roll = inclination angle of the hand around the main axis in degree (from -90.0 to +90.0) ;

pitch = inclination angle around the perpendicular axis in degree (from -90.0 to +90.0) ;

ax, ay, az = instantaneous hand acceleration in g (from -2.0 to +2.0);

Transmission Protocol

For advanced use a developer can develop its own library, the data coming from the data glove are packagerd in a simple way. Here it is the transmission protocol used by Vhand data glove:

RS232 Serial Port Setting:

Baud Rate: 115200 BPS
Data Bit: 8
Stop Bit: 1
Parity: NONE

- start transmission (PC to datagllve): send 's' to the glove;
- (Data glove to PC): the glove transmits the package continuously;
- stop transmission (PC to dataglove): send 'e' to the glove;

Package structure:

The glove continuously transmits the PC the following 20 byte package:

```
1 - header  = 0x20
2 - header  = 0x0A
3 - lenght = 0x14 (20 byte)
4 - acc axis x_l
5 - acc axis x_h
6 - acc axis y_l
7 - acc axis y_h
8 - acc axis z_l
9 - acc axis z_h
10- bend 0_l
11 - bend 0_h
12 - bend 1_l
13- bend 1_h
14- bend 2_l
15- bend 2_h
16 - bend 3_l
17 - bend 3_h
18 - bend 4_l
19 - bend 4_h
20 - crc
```

CRC represents the XOR of the first 19 bytes.
Bend value are from 0 to 1023 so only 10 bit are used.

Getting the Data: Visual C++ Example

```
#include "stdafx.h"
#include "windows.h"


int _tmain(int argc, _TCHAR* argv[])
{
	DCB dcb;
	BOOL fSuccess;
	COMMTIMEOUTS CommTimeouts;
	HANDLE hCom;
	int i;
	FILE *fp;
	char pathname[]="TestBT.txt";
	int TimePakIn=0;
	int TimePakfin=0;
	int Time=0;
	int TimeByte1Fin=0;
	int TimeByteIn=0;
	int TimeByteFin=0;
	int ax,ay,az,f1,f2,f3,f4,f5;

	LPCWSTR pcCommPort = argv[1];
	hCom = CreateFile( pcCommPort,
		GENERIC_READ | GENERIC_WRITE ,
		0,		// must be opened with exclusive-access
		NULL, // no security attributes
		OPEN_EXISTING, // must use OPEN_EXISTING
		0,		// not overlapped I/O
		NULL  // hTemplate must be NULL for comm devices
		);

	if (hCom == INVALID_HANDLE_VALUE)
	{
		// handle the error
		printf("cannot open serial port!");
		return 1;
	}

	// Build on the current configuration, and skip setting the size
	// of the input and output buffers with SetupComm.
	fSuccess = GetCommState(hCom, &dcb);
	if (!fSuccess)
	{
		printf("cannot open serial port!");
		return 1;
	}

	// Fill in DCB: 115,200 bps, 8 data bits, no parity, and 1 stop bit.
	dcb.BaudRate = CBR_115200;	// set the baud rate (sembra che non cambi niente)
	dcb.ByteSize = 8;		// data size, xmit, and rcv
	dcb.Parity = NOPARITY;		// no parity bit
	dcb.StopBits = ONESTOPBIT;	// one stop bit

	fSuccess = SetCommState(hCom, &dcb);

	if (!fSuccess)
	{
```

```
		printf("cannot open serial port");
		return 1;
	}

	CommTimeouts.ReadIntervalTimeout=100;
	CommTimeouts.ReadTotalTimeoutMultiplier=0;
	CommTimeouts.ReadTotalTimeoutConstant=4000;
	CommTimeouts.WriteTotalTimeoutConstant=0;
	CommTimeouts.WriteTotalTimeoutMultiplier=0;
	SetCommTimeouts(hCom,&CommTimeouts);

	printf("Porta Aperta Correttamente!");


	DWORD dwBytesTransferred;
	DWORD dwCommModemStatus;


	// START SAMPLING
	WriteFile(hCom,"s",1,&dwBytesTransferred,0);

	BYTE Byte[40];
	Byte[0]=0;

	for(;;)
	{
		TimePakIn=GetTickCount();
		ReadFile (hCom, Byte, 1, &dwBytesTransferred, 0);
		if (Byte[0]==0x20)
		{
			ReadFile (hCom, Byte, 1, &dwBytesTransferred, 0);
			if (Byte[0]==0x0a)
			{

				ReadFile (hCom, &Byte[0], 1, &dwBytesTransferred, 0);
				if (Byte[0]==0x14)
				{
				fprintf(fp, "Header found\n");
				ReadFile (hCom, Byte, 17, &dwBytesTransferred, 0);
				ax = Byte[0]+((Byte[1]&0x7f)<<8);
				if (Byte[1]&0x80)
					ax = -ax ;
				ay = Byte[2]+((Byte[3]<<8));
				az = Byte[4]+((Byte[5]<<8));
				f1 = Byte[6]+((Byte[7]&0x03)<<8);
				f2 = Byte[8]+((Byte[9]&0x03)<<8);
				f3 = Byte[10]+((Byte[11]&0x03)<<8);
				f4 = Byte[12]+((Byte[13]&0x03)<<8);
				f5 = Byte[14]+((Byte[15]&0x03)<<8);
				printf("ax:%5d ay:%5d az:%5d f1:%4d f2:%4d f3:%4d f4:%4d
					f5:%4d\n",ax,ay,az,f1,f2,f3,f4,f5);
				}
			}
		}
	}
	// STOP SAMPLING
	WriteFile(hCom,"e",1,&dwBytesTransferred,0);
	return 0;
}
```