PQ Labs

Multi-Touch Screen

SDK Reference

PQ Labs

PQ Multi-Touch SDK Reference

Revisions

v1.1 2009-06-16

v1.2 2009-07-22

Attention that client of version 1.2 is not compatible with that of v1.1, you need to recompile the applications. And the new client with version 1.2 require new multi-touch server with version 1.10 at least.

v1.3 2010-08-04

Add two interfaces: SetRawDataResolution, SetOnGetDeviceInfo.

Mofidy the function "ConnectServer" to support specifying the server socket port.

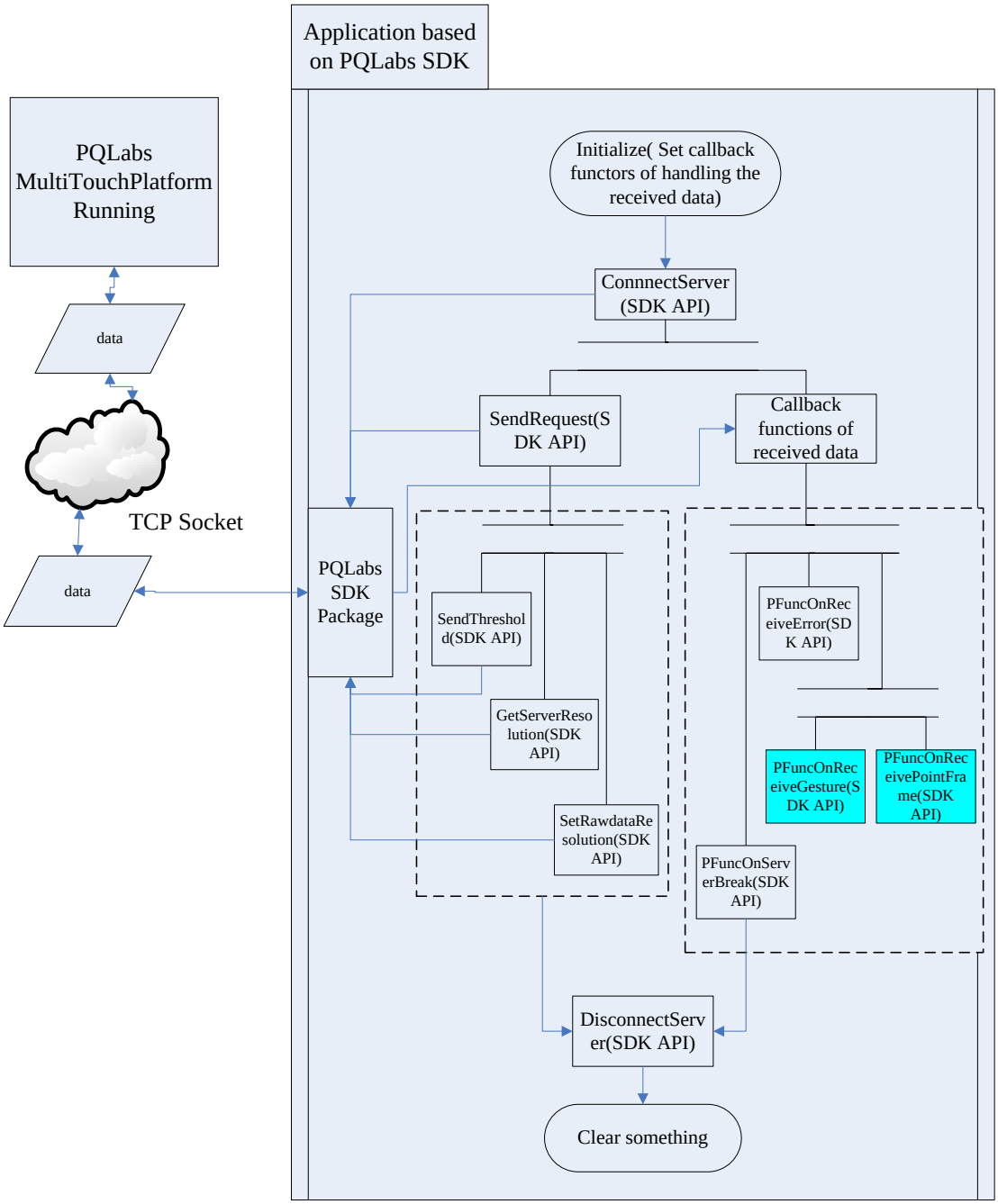
Require multitouch platform v4.1004 or later.

V1.3.6 2010-08-16

Fix bug that: the sample code demo may crash when invoking "DisconnectServer".

Contents

Flow Sheet



Functions

PQ Multi-Touch Platform support secondary development. The APIs below enable developers to create custom touching solutions conveniently.

ConnectServer

```
int ConnectServer(
    const char * ip = "127.0.0.1",
    int port = PQMT_DEFAULT_CLIENT_PORT
    );
```

Connect the multi-touch server.

Parameters:

ip

The ip address of the multi-touch server, "127.0.0.1" as default this function will connect to local machine.

port

The port of the server. PQMT_DEFAULT_CLIENT_PORT for default.

Return Values:

PQMTE_SUCCESS indicates connecting successfully, otherwise it will return a error code, which equal to the windows socket error code, see Winsock Error Codes from msdn.

Remarks:

When connect server successfully, function will send a default request RQST_RAWDATA_ALL.

Requirements:

See also:

DisconnectServer, Client Request Type.

DisconnectServer

```
int DisconnectServer();
```

Disconnect from the multi-touch server.

Return Values:

PQMTE_SUCCESS indicates disconnecting successfully, otherwise it will return a error code, which equal to the windows socket error code, see Winsock Error Codes from msdn.

See also:

ConnectServer.

SendRequest

```
int SendRequest(
    const TouchClientRequest & request
    );
```

After connect the multi-touch server successfully, send your request to the server. The request tell the server which service you'd like to enjoy.

Parameters:

request

Request information to send to the server. See TouchClientRequest.

Return Values:

PQMTE_SUCCESS indicates send successfully, otherwise it will return a error code, which maybe the socket error code or the window error code, see Winsock Error Codes from msdn.

Remarks:

If you don't send any request to the server, server will treat you as the default client which sends a default request to server. You can send a combination of request type in a request, such as RQST_RAWDATA_INSIDE | RQST_GESTURE_INSIDE. See TouchClientRequest about the combination.

See also:

ConnnectServer, TouchClientRequest.

SendThreshold

```
int SendThreshold(
    int move_threshold
    );
```

Defaultly the server will send out the moving touch points to client with a tolerance so that some "still" touch points, which is closed to its last move position, will be filtered. If you need all the touch points or you want to filter more "still" touch points, you can send specified move threshold to the server, the move threshold is in pixel.

Parameters:

move_threshold

It is the move threshold that will filter some points not move(the move_dis < threshold), it is in pixel(the pixel in the coordinate of server), 0 for highest sensitivity in server;

Return Values:

PQMTE_SUCCESS indicates send successfully, otherwise it will return a error code, which maybe the socket error code or the window error code, see Winsock Error Codes from msdn.

Remarks:

The send threshold can only work for RQST_RAWDATA_INSIDE request type;

See also:

ConnnectServer, TouchClientRequest.

SetOnReceivePointFrame

```
PFuncOnReceiveData SetOnReceiveData(
    PFuncOnReceivePointFrame pf_on_rcv_point_frame,
    void * call_back_object
    );
```

Set the function that you want to execute while receiving the touch point frame.

Parameters:

pfunc_on_rcv_point_frame

The function pointer you want to execute while receiving the touch points. See PFuncOnReceivePointFrame.

call_back_object

The object passed to pfunc_on_receive_data, generally it is used for passing an object pointer.

Return Values:

The old functor.

See also:

PFuncOnPointFrame.

SetOnReceiveGesture

```
PFuncOnReceiveData SetOnReceiveGesture(
    PFuncOnReceiveGesture pf_on_rcv_gesture,
    void * call_back_object
    );
```

Set the function that you want to execute while receiving the touch point frame.

Parameters:

pfunc_on_rcv_gesture

The function pointer you want to execute while receiving the touch gesture. See PFuncOnReceiveGesture.

call_back_object

A pointer passed to pfunc_on_receive_data, generally it is used for passing an object pointer.

Return Values:

The old functor.

See also:

PFuncOnReceiveGesture

SetOnServerBreak

```
PFuncOnServerBreak SetOnServerBreak(
    PFuncOnServerBreak pfunc_on_svr_break,
    void * call_back_object
    );
```

Set the function that you want to execute while receive the message that the server interrupt the connection.

Parameters:

pfunc_on_svr_break

The call back function pointer.

call_back_object

A pointer passed to pfunc_on_receive_data, generally it is used for passing an object

pointer.

Return Values:

The old function pointer.

See also:

PFuncOnServerBreak.

SetOnReceiveError

```
PFuncOnReceiveError SetOnReceiveError(
    PFuncOnReceiveError pfunc_on_rcv_error,
    void * call_back_object
    );
```

Set the function that you want to execute while some errors occur during the receive process.

Parameters:

pfunc_on_rcv_error

The call back function pointer.

call_back_object

A pointer passed to pfunc_on_receive_data, generally it is used for passing an object pointer.

Return Values:

The old function pointer.

See also:

PFuncOnReceiveError.

SetRawDataResolution

```
int SetRawDataResolution(
    int max_x,
    int max_y
    );
```

Set the resolution of the raw data(touch points). It's the resolution that the received touch points of this client based on.

Parameters:

max_x

resolution of the x axis.

max_y

resolution of the y axis.

Return Values:

PQMTE_SUCCESS for success, or windows socket error code for fail.

Requirements:

The server should be at least PQ MultitouchPlatform v3.1004.

SetOnGetDeviceInfo

```
PFuncOnGetDeviceInfo
SetOnGetDeviceInfo(
    PFuncOnGetDeviceInfo        pf_on_get_device_info,
    void * call_back_object
    );
```

The device information will be sent after the client SendRequest to server, or when the device plug in/out. Set the function that you want to execute while the client receive the device information here.

Parameters:

pf_on_get_device_info

The call back function pointer.

call_back_object

A pointer passed to pfunc_on_get_device_info, generally it is used for passing an object pointer.

Return Values:

PQMTE_SUCCESS for success, or windows socket error code for fail.

Requirements:

The server should be at least PQ MultitouchPlatform v3.1004.

See also:

PFuncOnGetDeviceInfo.

GetServerResolution

```
int GetServerResolution(
    PFuncOnGetServerResolution pFnCallback,
    void * call_back_object
    );
```

Get the display resolution of multi-touch server. Generally it needn't to get the resolution information of server but when the client is working at another PC different with server, the resolution is needed for the client calculating the point position and width/height.

Parameters:

pFnCallback

The call back function pointer when receiving the resolution infomation.

call_back_object

A pointer passed to pfunc_on_receive_data, generally it is used for passing an object pointer.

Return Values:

Name of the touch gesture.

See also:

PFuncOnGetServerResolution

GetGestureName

```
const char * GetGestureName(
    const TouchGesture & tg
    );
```

Get the touch gesture name of the touch gesture.

Parameters:

tg

The touch gesture.

Return Values:

Name of the touch gesture.

Remarks:

Requirements:

See also:

Struct Definitions

TouchPoint

Syntax:

```
struct TouchPoint
{
    unsigned short  point_event;
    unsigned short  id;
    int             x;
    int             y;
    unsigned short  dx;
    unsigned short  dy;
};
```

The raw touch point received from the server.

DataMembers:

point_event

Indicates current action or event of the touch point, It is one of the values in table: TouchPoint Type.

id

Use id to distinguish different points on the screen.

x

Specifies the x-coordinate of the center position of the point. In pixels.

y

Specifies the y-coordinate of the center position of the point. In pixels.

dx

Specifies the x-width of the touch point. In pixels.

dy

Specifies the y-width of the touch point. In pixels.

Remarks:

See also:

TouchPoint Event Type

TouchGesture

Syntax:

```
struct TouchGesture
{
    // type
    unsigned short   type;
    // param size
    unsigned short   param_size;
    // params
    double           params[MAX_TG_PARAM_SIZE];
};
```

DataMembers:

type

Provides a code to distinguish different gestures. It is one of the values in table: TouchGesture Type.

param_size

Specifies size of the params that the gesture contains.

params

Contains the values of the gesture params.

Remarks:

See also:

TouchGesture Type

TouchClientRequest

Syntax:

```
struct TouchClientRequest
{
    // request type
    int              type;
    // an sole id of your application, it's reserved at present
    GUID             app_id;
```

```
    // for RQST_TRANSLATOR_CONFIG, it is the name of gesture translator which will be queried from the server configure tools; otherwise, it is a reserved param.
    char            param[128];
};
```

Contain the request information that you want to tell the server. It is send by the function SendRequest.

DataMembers:

type

Specifies what service you want the server to provide.It can be combination of the values in table Client Request Type.

app_id

An license key id of your application. You should send this key id to tell the server activate the receive right of your application. It's reserved at present.

Param

If type is RQST_TRANSLATOR_CONFIG, it is the name of gesture translator which will be queired from the server configure tools. Otherwise, it is reserved.

Remarks:

See also:

SendRequest, Client Request Type.

TouchDeviceInfo

Syntax:

```
struct PQMT_CLIENT_API TouchDeviceInfo{
    int    screen_width;    //the physical touchable width of touch screen device
    int    screen_height;//the physical touchable height of touch screen device
    char   serial_number[128];    //the serial number of touch screen device
};
```

Informaion of the touch device.

DataMembers:

screen_width

The physical touchable width of the touch screen. In millimeter.

screen_height

The physical touchable height of the touch screen. In millimeter.

serial_number

An unique id of the touch device.

See also:

SetOnGetDeviceInfo.

Macro Definitions

Error Type

Type	Descriptions
PQMTE_SUCCESS	OK.
PQMTE_RCV_INVALIDATE_DATA	the data received is invalidate,may be the client receive the data from other application but pq multi-touch server.
PQMTE_SERVER_VERSION_OLD	the pq multi-touch server is too old for this version of client.
Others	Socket error code or windows error code.

TouchPoint Event Type

Type	Descriptions
TP_DOWN	It is the first time that the touch point occur. It is called "down" event.
TP_MOVE	The touch point is move on the screen after "down".
TP_UP	The touch point leave the screen, called "up" event.

TouchGesture Type

Remarks

If the param value indicates coordinates, it is in pixels. All the parameters are positive.

Here, we take finger for example as the object that you use to touch the screen.

"Ditto" here means " the same as above".

Here, "finger down" always means "finger touch onto the screen".

Type	Descriptions	Paras size	Params
TG_TOUCH_START	Indicates that the first touch come, you can initialize something here.	0	
TG_DOWN	A single finger touch on the screen.	2	The center position of the touch finger. Params[0]: x-coordinates Params[1]: y-coordinates

TG_MOVE	The touch point is moving after TG_DOWN.	2	Ditto
TG_UP	The touch point will leave the screen.	2	Ditto.
TG_CLICK	A single finger click on the screen.	2	Ditto.
TG_DB_CLICK	A single finger double click on the screen.	2	Ditto.
TG_BIG_DOWN	A single fist or something with "big" touch area touching on the screen.	2	Ditto.
TG_BIG_MOVE	The fist is moving after TG_BIG_DOWN.	2	Ditto.
TG_BIG_UP	The touching fist will leave the screen.	2	Ditto.
TG_MOVE_RIGHT	The single touching finger move to right with a distance.	2	Ditto.
TG_MOVE_UP	Ditto. But move to up.	2	Ditto.
TG_MOVE_LEFT	Ditto. But move to left.	2	Ditto.
TG_MOVE_DOWN	Ditto. But move to down.	2	Ditto.
TG_SECOND_DOWN	After a single finger touch down on the screen, the second finger comes and touch on the screen.	4	(params[0],parmas[1]) indicate the center position of the second finger, params[0] is x-coordinate and params[1] is y. (params[2],params[3]) indicate the center position of the first finger.
TG_SECOND_UP	The second finger will leave the screen after TG_SECOND_DOWN.	4	Ditto.
TG_SECOND_CLICK	The second finger click.	4	Ditto.
TG_SECOND_DB_CLICK	The second finger double click.	4	Ditto.
TG_SPLIT_START	There are two fingers on the screen and the two fingers are moving to contrary direction. This gesture indicates that the split gesture	4	The params indicate the positions of the two touching finger. (params[0], params[1]) for one and (params[2], params[3]) for the other.

	comes.		
TG_SPLIT_APART	The two split fingers is split apart with their distance increasing.	6	params[0] indicate the delta distance in pixels. params[1] indicate the delta ratio to the last distance of the two fingers.(params[2],params[3]) and (params[4],params[5]) is the positions of the two points.
TG_SPLIT_CLOSE	Contrary to TG_SPLIT_APART.	6	Ditto.
TG_SPLIT_END	The Split gesture will end. The reason may be one of the bellows: 1. one of the two fingers touch up. 2. the two fingers now are doing some other gestures, known or unknown. 3. The third finger have touched down.	4	Same with TG_SPLIT_START.
TG_ROTATE_START	There are two fingers touching the screen and one is anchored at the focus position while the other is rotating around the focus point. TG_ROTATE_START indicates that the rotate gesture is coming.	4	Like that of the Split Gesture, the 4 parameters indicate the positions of the two fingers. But here, the first finger which indicated by (params[0], params[1]) is the anchor finger while the second finger ,(params[2], param[3]), is the round finger.
TG_ROTATE_ANTICLOCK	The rounding finger is moving anticlockwise.	5	Params[0] indicates the rotate angle in radians.(params[1],params[2]) and (params[3],params[4]) indicate the positons of the two touching fingers.(params[1],params[2]) is the anchor finger.
TG_ROTATE_CLOCK	Contrary to above.	5	Ditto.

TG_ROTATE_END	The rotate gesture will end. The reason may be one of the bellows: 1. one the two fingers touch up. 2. the two fingers now are doing some other gestures, known or unknown. 3. The third finger have touched down.	4	Same with TG_ROTATE_START
TG_NEAR_PARREL_DOWN	There are two fingers touching on the screen and they touch the screen in the same time and they are in a short distance to each other. It is different from TG_SECOND_DOWN. The latter is that the second finger is down after a while, short or long, since the first finger touching on the screen.	6	(params[0],params[1]) indicate the middle position of the two fingers. (params[2],params[3]) indicate the first position of the two fingers. (params[4],params[5]) indicate the second position of the two fingers.
TG_NEAR_PARREL_MOVE	The two finger moves and remains their short distance.	6	Ditto.
TG_NEAR_PARREL_UP	One of the two fingers touch up or there comes the third or more fingers.	6	Ditto.
TG_NEAR_PARREL_CLICK	Two fingers click in the “same” time on the screen and there the TG_NEAR_PARREL_DOWN and TG_NERA_PRREL_UP come before the click.	2	(params[0],params[1]) indicate the middle position of the two fingers TG_NEAR_PARREL_DOWN.
TG_NEAR_PARREL_DB_CLICK	Two TG_NEAR_PARREL_CLICK come one by one in a short	2	Ditto.

	time.		
TG_NEAR_PARREL_MOVE_RIGHT	The middle point of the two fingers move to right.	2	Ditto.
TG_NEAR_PARREL_MOVE_UP	Ditto, but move to up.	2	Ditto.
TG_NEAR_PARREL_MOVE_LEFT	Ditto, but move to left	2	Ditto.
TG_NEAR_PARREL_MOVE_DOWN	Ditto, but move to down.	2	Ditto.
TG_MULTI_DOWN	There are three or more fingers touching on the screen. Every the first time there comes more than two fingers, there comes TG_MULTI_DOWN.	2	(params[0], params[1]) indicates the center point of those fingers.
TG_MULTI_MOVE	The center point of those fingers is moving.	2	Ditto.
TG_MULTI_UP	Some fingers touch up and there are less than three fingers left on the screen.	2	Ditto.
TG_MULTI_MOVE_RIGHT	The center point of those fingers is moving to right.	2	Ditto.
TG_MULTI_MOVE_UP	Ditto, but move to up.	2	Ditto.
TG_MULTI_MOVE_LEFT	Ditto, but move to left.	2	Ditto.
TG_MULTI_MOVE_DOWN	Ditto, but move to down.	2	Ditto.
TG_TOUCH_END	There are no fingers touching the screen.The touch end. You can "clear" here.	0	
TG_NO_ACTION	It just a signal that the gesture is not implemented or something else that lead to the current gesture not used. What you need to do for this signal is just do nothing.		

Client Request Type

Type	value	Description
RQST_RAWDATA_INSIDE_ONLY	0x0000	The Server will send the raw touch points to this client application only when the first finger is touching down in the active rectangle of the main window. This query is similar with RQST_RAWDATA_INSIDE but that it can not combine with other request.
RQST_RAWDATA_INSIDE	0x0001	The Server will send the raw touch points to this client application when the first finger is touching down in the active rectangle of the main window.
RQST_RAWDATA_ALL	0x0002	The Server will send all the raw touch points to this client application when fingers touch the screen.
RQST_GESTURE_INSIDE	0x0004	The Server will send the touch gestures to this client application when the first finger of the gesture is touching down in the active rectangle of the main window.
RQST_GESTURE_ALL	0x0008	The Server will send all the touch gestures to this client application when fingers touch the screen.
RQST_TRANSLATOR_CONFIG	0x0010	This request will tell the server to translator the gestures to the system input with the translators whose name is the content of "params"(data member of TouchClientRequest) when activating the client application. You can get the names form the MultiTouchServer Configuration Tools. This request doesn't tell the server to send touch data to the client application.

Remarks

All the requests can combine with others except RQST_RAWDATA_INSIDE_ONLY.

Function Pointers

PFuncOnReceivePointFrame

```
typedef void (*PFuncOnReceivePointFrame)(
    int frame_id,
    int time_stamp,
    int moving_point_count,
    const TouchPoint * moving_point_array,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when the touch point frame coming. The touch points unmoving won't be sent from the server for the sake of efficency. The new touch point with its pointevent being TP_DOWN and the leaving touch point with its pointevent being TP_UP will be always sent from server.

Params:

frame_id : a unique id for the current frame;

time_stamp : the time flag when the frame generated, it is in milli-seconds;

moving_point_count : the count of the moving or new/leaving points in this frame;

moving_point_array : the moving or new/leaving points data in this frame;

call_back_object : a pointer coming from the "SetOnReceivePointFrame", it is an object pointer generally;

PFuncOnReceiveGesture

```
typedef void (* PFuncOnReceiveGesture)(
    const TouchGesture & gesture,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when the touch gesture coming.

Params:

gesture : a unique id for the current frame;

call_back_object : a pointer coming from "SetOnReceiveGesture", it is an object pointer generally;

PFuncOnServerBreak

```
typedef void (*PFuncOnServerBreak)(
    void * param,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when the multi-touch server interrupt the connection.

Params:

param : reserved;

call_back_object : a pointer coming from "SetOnServerBreak", it is an object pointer generally;

PFuncOnReceiveError

```
typedef void (*PFuncOnReceiveError)(
    int error_code,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when some error occur during the touch data receiving process.

Params:

error_code: error_code of PQ multi-touch client, it is generally an error code of windows socket error code, except that "PQMTE_RCV_INVALIDATE_DATA", which means that there the data received is invalidate, the reason may be that the server version is not compatibal with the client version;

call_back_object : a pointer coming from "SetOnReceiveError", it is an object pointer generally;

PFuncOnGetServerResolution

```
typedef void (* PFuncOnGetServerResolution)(
    int max_x,
    int max_y,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when gettomg the display resolution information of multi-touch server;

Params:

max_x : the max pixels of x axis.

max_y : the max pixels of y axis.

call_back_object : a pointer coming from "GetServerResolution", it is an object pointer generally;

PFuncOnGetDeviceInfo

```
typedef void (*PFuncOnGetDeviceInfo)(
    const TouchDeviceInfo &device_info,
    void * call_back_object
    );
```

Call back funciton pointer which defines the function that you want to call when receiving the device physical information. The device information will be sent after the client sent request or while the touch device plugged in.

Params:

device_info

the physical information of the device.see TouchDeviceInfo for more details.

call_back_object : a pointer coming from "SetOnGetDeviceInfo", it is an object pointer generally;

Sample Code

The Sample Code here is written by C++. It is compiled in Visual Studio 2005.

Header Files

SDK_SampleCode.h

```
//+-------------------------------------------------------------------------
//
//    PQLabs.
//
//    Copyright (c) PQLabs.    All rights reserved.
//
//    File:              SDK_SampleCode.h
//
//    Contents:      Sample code for MultiTouch Clinet SDK APIs.
//
//      Date:          2008-12-19
//
//--------------------------------------------------------------------------
#ifndef PQ_SDK_MULTITOUCH_SAMPLE_H_
#define PQ_SDK_MULTITOUCH_SAMPLE_H_

#include "PQMTClient.h"
using namespace PQ_SDK_MultiTouch;
namespace PQ_SDK_MultiTouchSample
{
class Sample{
public:
        Sample();
        ~Sample();
        // Init: the entry of sample codes;
        //              demonstrate: ConnectServer, SendRequest etc;
        int Init();
private:
//////////////////////call back functions/////////////////////
        // OnReceivePointFrame: function to handle when recieve touch point frame
        //      the unmoving touch point won't be sent from server. The new touch point with its pointevent is
TP_DOWN
        //      and the leaving touch point with its pointevent will be always sent from server;
        static void OnReceivePointFrame(int frame_id,int time_stamp,int moving_point_count,const
TouchPoint * moving_point_array, void * call_back_object);
        // OnReceivePointFrame: function to handle when recieve touch gesture
```

```
        static void OnReceiveGesture(const TouchGesture & ges, void * call_back_object);
        // OnServerBreak: function to handle when server break(disconnect or network error)
        static void OnServerBreak(void * param, void * call_back_object);
        // OnReceiveError: function to handle when some errors occur on the process of receiving touch datas.
        static void OnReceiveError(int err_code,void * call_back_object);
        static void OnGetServerResolution(int x, int y, void * call_back_object);
////////////////////////call back functions end ////////////////////////
        // functions to handle TouchGestures, attention the means of the params
        void InitFuncOnTG();
        // set the call back functions while reciving touch data;
        void SetFuncsOnReceiveProc();
        // OnTouchPoint: function to handle TouchPoint
        void OnTouchPoint(const TouchPoint & tp);
        // OnTouchGesture: function to handle TouchGesture
        void OnTouchGesture(const TouchGesture & tg);
        //
        //here use function pointer table to handle the different gesture type;
        typedef void (*PFuncOnTouchGesture)(const TouchGesture & tg,void * call_object);
        static void DefaultOnTG(const TouchGesture & tg,void * call_object); // just show the gesture
        static void OnTG_TouchStart(const TouchGesture & tg,void * call_object);
        static void OnTG_Down(const TouchGesture & tg,void * call_object);
        static void OnTG_Move(const TouchGesture & tg,void * call_object);
        static void OnTG_Up(const TouchGesture & tg,void * call_object);
        //
        static void OnTG_SecondDown(const TouchGesture & tg,void * call_object);
        static void OnTG_SecondUp(const TouchGesture & tg,void * call_object);
        //
        static void OnTG_SplitStart(const TouchGesture & tg,void * call_object);
        static void OnTG_SplitApart(const TouchGesture & tg,void * call_object);
        static void OnTG_SplitClose(const TouchGesture & tg,void * call_object);
        static void OnTG_SplitEnd(const TouchGesture & tg,void * call_object);
        // OnTG_TouchEnd: to clear what need to clear;
        static void OnTG_TouchEnd(const TouchGesture & tg,void * call_object);
private:
        PFuncOnTouchGesture m_pf_on_tges[TG_TOUCH_END + 1];
// sample code end
};
}; // end of namespace
#endif // end of header
```

Stdafx.h

```
// stdafx.h : include file for standard system include files,
……
#pragma comment(lib,"PQMTClient.lib")
```

CPP Files

Main.cpp/z_TestSDK.cpp

```
// z_TestSDK.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "SDK_SampleCode.h"
#include <iostream>
using namespace std;
using namespace PQ_SDK_MultiTouchSample;

int _tmain(int argc, _TCHAR* argv[])
{
	Sample sample;
	int err_code = sample.Init();
	if(err_code != PQ_MT_SUCESS){
		cout << "press any key to exit..." << endl;
		getchar();
		return 0;
	}
	// do other things of your application;
	cout << "hello world" << endl;
	//
	// here just wait here, not let the process exit;
		// here just wait here, not let the process exit;
	char ch	= 0;
	while(ch != 'q' && ch != 'Q'){
		cout << "press \'q\' to exit" << endl;
		ch = getchar();
	}
	return 0;
}
```

SDK_SampleCode.cpp

```
//+------------------------------------------------------------------------
//
//	PQLabs.
//
//	Copyright (c) PQLabs.	All rights reserved.
//
//	File:		SDK_SampleCode.cpp
```

```
//
//    Contents:     Implementation of SDK_SampleCode.h
//
//      Date:         2008-12-19
//
//----------------------------------------------------------------------

#include "stdafx.h"
#include "SDK_SampleCode.h"

#include <iostream>
#include <set>
#include <map>
#include <cassert>
#include <functional>
using namespace std;

namespace PQ_SDK_MultiTouchSample
{

Sample::Sample()
{
        memset(m_pf_on_tges,0, sizeof(m_pf_on_tges));
}
Sample::~Sample()
{
        DisconnectServer();
}
//////////////////////////// functions ////////////////////////////////////
int Sample::Init()
{
        int err_code = PQ_MT_SUCESS;

        // initialize the handle functions of gestures;
        InitFuncOnTG();
        // set the functions on server callback
        SetFuncsOnReceiveProc();
        // connect server
        cout << " connect to server..." << endl;
        if((err_code = ConnectServer()) != PQ_MT_SUCESS){
                cout << " connect server fail, socket error code:" << err_code << endl;
                return err_code;
        }
        // send request to server
```

```
	cout << " connect success, send request." << endl;
	TouchClientRequest tcq = {0};
	tcq.type = RQST_RAWDATA_ALL | RQST_GESTURE_ALL;
	if((err_code = SendRequest(tcq)) != PQ_MT_SUCESS){
		cout << " send request fail, err_code:" << err_code << endl;
		return err_code;
	}
	//////////////you can set the move_threshold when the tcq.type is RQST_RAWDATA_INSIDE;
	//send threshold
	//int move_threshold = 1;// 1 pixel
	//if((err_code = SendThreshold(move_threshold) != PQMTE_SUCESS){
	//	cout << " send threadhold fail, error code:" << err_code << endl;
	//	return err_code;
	//}
	//
	//////////////////////
	//get server resolution
	if((err_code = GetServerResolution(OnGetServerResolution, NULL)) != PQ_MT_SUCESS){
		cout << " get server resolution fail" << endl;
		return err_code;
	};
	// start receiving
	cout << " send request success, start recv." << endl;
	return err_code;
}

void Sample:: InitFuncOnTG()
{
	// initialize the call back functions of toucha gestures;
	m_pf_on_tges[TG_TOUCH_START] = &Sample::OnTG_TouchStart;
	m_pf_on_tges[TG_DOWN] = &Sample::OnTG_Down;
	m_pf_on_tges[TG_MOVE] = &Sample::OnTG_Move;
	m_pf_on_tges[TG_UP] = &Sample::OnTG_Up;

	m_pf_on_tges[TG_SECOND_DOWN] = &Sample::OnTG_SecondDown;
	m_pf_on_tges[TG_SECOND_UP] = &Sample::OnTG_SecondUp;

	m_pf_on_tges[TG_SPLIT_START] = &Sample::OnTG_SplitStart;
	m_pf_on_tges[TG_SPLIT_APART] = &Sample::OnTG_SplitApart;
	m_pf_on_tges[TG_SPLIT_CLOSE] = &Sample::OnTG_SplitClose;
	m_pf_on_tges[TG_SPLIT_END] = &Sample::OnTG_SplitEnd;

	m_pf_on_tges[TG_TOUCH_END] = &Sample::OnTG_TouchEnd;
}
```

```
void Sample::SetFuncsOnReceiveProc()
{
	PFuncOnReceivePointFrame old_rf_func =
SetOnReceivePointFrame(&Sample::OnReceivePointFrame,this);
	PFuncOnReceiveGesture old_rg_func = SetOnReceiveGesture(&Sample::OnReceiveGesture,this);
	PFuncOnServerBreak old_svr_break = SetOnServerBreak(&Sample::OnServerBreak,NULL);
	PFuncOnReceiveError old_rcv_err_func = SetOnReceiveError(&Sample::OnReceiveError,NULL);
}

void Sample:: OnReceivePointFrame(int frame_id, int time_stamp, int moving_point_count, const
TouchPoint * moving_point_array, void * call_back_object)
{
	Sample * sample = static_cast<Sample*>(call_back_object);
	assert(sample != NULL);
	const char * tp_event[] =
	{
		"down",
		"move",
		"up",
	};

	cout << " frame_id:" << frame_id << " time:"    << time_stamp << " ms" << " moving point count:" <<
moving_point_count << endl;
	for(int i = 0; i < moving_point_count; ++ i){
		TouchPoint tp = moving_point_array[i];
		sample->OnTouchPoint(tp);
	}
}
void Sample:: OnReceiveGesture(const TouchGesture & ges, void * call_back_object)
{
	Sample * sample = static_cast<Sample*>(call_back_object);
	assert(sample != NULL);
	sample->OnTouchGesture(ges);
}
void Sample:: OnServerBreak(void * param, void * call_back_object)
{
	// when the server break, disconenct server;
	cout << "server break, disconnect here" << endl;
	DisconnectServer();
}
void Sample::OnReceiveError(int err_code, void * call_back_object)
{
	switch(err_code)
	{
```

```
        case PQMTE_RCV_INVALIDATE_DATA:
                cout << " error: receive invalidate data." << endl;
                break;
        case PQMTE_SERVER_VERSION_OLD:
                cout << " error: the multi-touch server is old for this client, please update the multi-touch server."
<< endl;
                break;
        default:
                cout << " socket error, socket error code:" << err_code << endl;
        }
}
void Sample:: OnGetServerResolution(int x, int y, void * call_back_object)
{
        cout << " server resolution:" << x << "," << y << endl;
}
// here, just record the position of point,
//      you can do mouse map like "OnTG_Down" etc;
void Sample:: OnTouchPoint(const TouchPoint & tp)
{
        switch(tp.point_event)
        {
        case TP_DOWN:
                cout << "    point " << tp.id << " come at (" << tp.x << "," << tp.y
                        << ") width:" << tp.dx << " height:" << tp.dy << endl;
                break;
        case TP_MOVE:
                cout << "    point " << tp.id << " move at (" << tp.x << "," << tp.y
                        << ") width:" << tp.dx << " height:" << tp.dy << endl;
                break;
        case TP_UP:
                cout << "    point " << tp.id << " leave at (" << tp.x << "," << tp.y
                        << ") width:" << tp.dx << " height:" << tp.dy << endl;
                break;
        }
}
void Sample:: OnTouchGesture(const TouchGesture & tg)
{
        if(TG_NO_ACTION == tg.type)
                return ;

        assert(tg.type <= TG_TOUCH_END);
        DefaultOnTG(tg,this);
        PFuncOnTouchGesture pf = m_pf_on_tges[tg.type];
        if(NULL != pf){
```

```
            pf(tg,this);
        }
}
void Sample:: OnTG_TouchStart(const TouchGesture & tg,void * call_object)
{
        assert(tg.type == TG_TOUCH_START);
        cout << "    here, the touch start, initialize something." << endl;
}
void Sample:: DefaultOnTG(const TouchGesture & tg,void * call_object) // just show the gesture
{
        cout <<"ges,name:"<< GetGestureName(tg) << " type:" << tg.type << ",param size:" << tg.param_size
<< " ";
        for(int i = 0; i < tg.param_size; ++ i)
            cout << tg.params[i] << " ";
        cout << endl;
}
void Sample:: OnTG_Down(const TouchGesture & tg,void * call_object)
{
        assert(tg.type == TG_DOWN && tg.param_size >= 2);
        cout << "    the single finger touching at :( "
            << tg.params[0] << "," << tg.params[1] << " )" << endl;
}
void Sample:: OnTG_Move(const TouchGesture & tg,void * call_object)
{
        assert(tg.type == TG_MOVE && tg.param_size >= 2);
        cout << "    the single finger moving on the screen at :( "
            << tg.params[0] << "," << tg.params[1] << " )" << endl;
}
void Sample:: OnTG_Up(const TouchGesture & tg,void * call_object)
{
        assert(tg.type == TG_UP && tg.param_size >= 2);
        cout << " the single finger is leaving the screen at :( "
            << tg.params[0] << "," << tg.params[1] << " )" << endl;
}
//
void Sample:: OnTG_SecondDown(const TouchGesture & tg,void * call_object)
{
        assert(tg.type == TG_SECOND_DOWN && tg.param_size >= 4);
        cout << "    the second finger touching at :( "
            << tg.params[0] << "," << tg.params[1] << " ),"
            << " after the first finger touched at :( "
            << tg.params[2] << "," << tg.params[3] << " )" << endl;
}
void Sample:: OnTG_SecondUp(const TouchGesture & tg,void * call_object)
```

```
{
	assert(tg.type == TG_SECOND_UP && tg.param_size >= 4);
	cout << "    the second finger is leaving at :( "
		<< tg.params[0] << "," << tg.params[1] << " ),"
		<< " while the first finger still anchored around :( "
		<< tg.params[2] << "," << tg.params[3] << " )" << endl;
}
//
void Sample:: OnTG_SplitStart(const TouchGesture & tg,void * call_object)
{
	assert(tg.type == TG_SPLIT_START && tg.param_size >= 4);
	cout << "    the two fingers is splitting with one finger at: ( "
		<< tg.params[0] << "," << tg.params[1] << " ),"
		<< " , the other at :( "
		<< tg.params[2] << "," << tg.params[3] << " )" << endl;
}

void Sample:: OnTG_SplitApart(const TouchGesture & tg,void * call_object)
{
	assert(tg.type == TG_SPLIT_APART && tg.param_size >= 1);
	cout << "    the two fingers is splitting apart with there distance incresed by "
		<< tg.params[0]
		<< " with a ratio :" << tg.params[1]
		<< endl;
}
void Sample:: OnTG_SplitClose(const TouchGesture & tg,void * call_object)
{
	assert(tg.type == TG_SPLIT_CLOSE && tg.param_size >= 1);
	cout << "    the two fingers is splitting close with there distance decresed by "
		<< tg.params[0]
		<< " with a ratio :" << tg.params[1]
		<< endl;
}
void Sample:: OnTG_SplitEnd(const TouchGesture & tg,void * call_object)
{
	assert(tg.type == TG_SPLIT_END);
	cout << "    the two splitting fingers with one finger at: ( "
		<< tg.params[0] << "," << tg.params[1] << " ),"
		<< " , the other at :( "
		<< tg.params[2] << "," << tg.params[3] << " )"
		<< " will end" << endl;
}
// OnTG_TouchEnd: to clear what need to clear
void Sample:: OnTG_TouchEnd(const TouchGesture & tg,void * call_object)
```

```
{
        assert(tg.type == TG_TOUCH_END);
        cout << "    all the fingers is leaving and there is no fingers on the screen." << endl;
}
//////////////////////////// functions ////////////////////////////////////


}
```