

## UTPAPI CONTENTS

GETEXISTCONTROLLERS FUNCTION .....	4
GETCONTROLLERNAME FUNCTION .....	5
OPENCONTROLLER FUNCTION .....	6
CLOSECONTROLLER FUNCTION.....	7
DISABLETOUCHREPORT FUNCTION .....	8
GETREPORTSTATUS FUNCTION.....	9
SETROTATION FUNCTION.....	10
GETROTATION FUNCTION.....	11
ENABLEUPSOUND FUNCTION.....	12
ENABLE DOWNSOUND FUNCTION.....	13
GETUPSOUNDSTATUS FUNCTION .....	14
GETDOWNSOUNDSTATUS FUNCTION.....	15
CLOSECOMPORT FUNCTION .....	16
REOPENCOMPORT FUNCTION .....	17
CHANGECOMPORT FUNCTION .....	18
CURRENTCOMPORT FUNCTION .....	20
WRITEBUFFER FUNCTION.....	21
READBUFFER FUNCTION .....	22
READLENGTH FUNCTION.....	23
FLUSHREADBUFFER FUNCTION .....	24
SETCALPOINT FUNCTION.....	25
CALCURRENTPOINTREADY FUNCTION .....	26
ISPENUP FUNCTION .....	28
STORECALIBRATIONDATA FUNCTION.....	29
SETDOUBLECLICKSPEED FUNCTION.....	30
SETDOUBLECLICKAREA FUNCTION .....	31
GETDOUBLECLICKSPEED FUNCTION.....	32
GETDOUBLECLICKAREA FUNCTION.....	33
SETREPORTMODE FUNCTION .....	34

<b>GETREPORTMODE FUNCTION .....</b>	<b>35</b>
<b>SETRIGHTCLICKTIME FUNCTION .....</b>	<b>36</b>
<b>GETRIGHTCLICKTIME FUNCTION .....</b>	<b>37</b>
<b>ENABLERIGHTCLICK FUNCTION .....</b>	<b>38</b>
<b>GETRIGHTCLICKENABLED FUNCTION .....</b>	<b>39</b>
<b>GETDRIVERVERSION FUNCTION.....</b>	<b>40</b>
<b>GETDRIVERINTERFACE FUNCTION.....</b>	<b>41</b>
<b>ASKTOSETDISPLAYNUMBER FUNCTION .....</b>	<b>42</b>
<b>QUERYDISPLAYBETOUCHED FUNCTION .....</b>	<b>43</b>
<b>GETDISPLAYNUMBER FUNCTION .....</b>	<b>44</b>
<b>GETEXISTDISPLAYS FUNCTION.....</b>	<b>45</b>
<b>GETDISPLAYNAME FUNCTION.....</b>	<b>46</b>
<b>GETDISPLAYAREA FUNCTION .....</b>	<b>47</b>
<b>GETMEASURETIME FUNCTION.....</b>	<b>48</b>
<b>SETMEASURETIME FUNCTION .....</b>	<b>49</b>
<b>GETMEASUREDELAY FUNCTION.....</b>	<b>50</b>
<b>SETMEASUREDELAY FUNCTION.....</b>	<b>51</b>
<b>GETINFOCUS FUNCTION.....</b>	<b>52</b>
<b>SETINFOCUS FUNCTION.....</b>	<b>53</b>
<b>GETOUTFOCUS FUNCTION.....</b>	<b>54</b>
<b>SETINFOCUS FUNCTION.....</b>	<b>55</b>
<b>SETCALIBRATIONPOINT FUNCTION.....</b>	<b>56</b>
<b>GETCALIBRATIONDATA FUNCTION.....</b>	<b>57</b>
<b>SAVECALIBRATIONDATA FUNCTION .....</b>	<b>58</b>
<b>SET_SOUND_TYPE FUNCTION .....</b>	<b>60</b>
<b>GET_SOUND_TYPE FUNCTION.....</b>	<b>61</b>
<b>GET_MANUAL_RIGHTCLICK_STATUS FUNCTION .....</b>	<b>62</b>
<b>SET_MRIGHTCLICK_APP FUNCTION.....</b>	<b>62</b>
<b>GETTOUCHSENSITIVITY FUNCTION .....</b>	<b>63</b>
<b>SETTOUCHSENSITIVITY FUNCTION .....</b>	<b>64</b>
<b>EXAMPLE PROGRAM #1 .....</b>	<b>65</b>

<b>EXAMPLE PROGRAM #2 .....</b>	<b>67</b>
<b>EXAMPLE PROGRAM #3 .....</b>	<b>69</b>
<b>EXAMPLE PROGRAM #4 .....</b>	<b>71</b>
<b>EXAMPLE PROGRAM #5 .....</b>	<b>72</b>
<b>EXAMPLE PROGRAM #6 .....</b>	<b>74</b>
<b>EXAMPLE PROGRAM #7 .....</b>	<b>75</b>
<b>EXAMPLE PROGRAM #8 .....</b>	<b>77</b>
<b>EXAMPLE PROGRAM #9 .....</b>	<b>87</b>
<b>EXAMPLE PROGRAM #10 .....</b>	<b>89</b>
<b>EXAMPLE PROGRAM #11.....</b>	<b>90</b>
<b>EXAMPLE PROGRAM #12 .....</b>	<b>92</b>
<b>EXAMPLE PROGRAM #13 .....</b>	<b>93</b>
<b>EXAMPLE PROGRAM #14 .....</b>	<b>102</b>
<b>EXAMPLE PROGRAM #15 .....</b>	<b>103</b>
<b>EXAMPLE PROGRAM #16 .....</b>	<b>104</b>
<b>EXAMPLE PROGRAM #17 .....</b>	<b>106</b>
<b>EXAMPLE PROGRAM #18 .....</b>	<b>107</b>
<b>EXAMPLE PROGRAM #19 .....</b>	<b>117</b>
<b>EXAMPLE PROGRAM #20 .....</b>	<b>118</b>

## **GetExistControllers Function**

The **GetExistControllers** function gets the number of existing controllers on the local computer. And builds the device list to support to select a device to interact.

### **Syntax**

```
BYTE GetExistControllers (  
    VOID  
);
```

### **Parameters**

*NULL*

### **Return Value**

If the function succeeds, the return value will be 1~9. It is dependent on how many controllers there are on your computer.

### **Remarks**

This function will search all devices and get their device name to build a device list. So, if you want to interact with someone device, you need call this function first.

### **Example**

See Example Program #1 (1)

## **GetControllerName Function**

The **GetControllerName** function gives programmer the name of device from the index of device list.

### **Syntax**

```
BOOL   GetControllerName(  
    BYTE ControllerNum,  
    CHAR *devName,  
    BYTE Len  
);
```

### **Parameters**

ControllerNum  
[in]  
Input the index to get device name. This value should be between 1 and 9.

devName  
[out]  
Your need support a buffer to copy the device name. The size of device name will not exceed 20 characters.

Len  
[in]  
Specifies the size of devName buffer.

### **Return Value**

If the function succeeds, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

If the function succeeds, the **devName** buffer will be the device name by **ControllerNum**. Otherwise, it will be NULL. You can use devName to get device handle to interact with it.

## Example

See Example Program #1 (2)

## OpenController Function

The **OpenController** function gets the handle of device from its name.

## Syntax

```
HANDLE OpenController (  
    CHAR *pdevicename  
);
```

## Parameters

pdevicename  
[in]  
Input the device name that you want to use.

## Return Value

If the function succeeds, the return value will be none zero value. Otherwise, it is 0.

## Remarks

If the function succeeds, you can use this handle to access the device with associate funcutions. The device name comes from **GetControllerName** function.

## Example

See Example Program #1 (3)

## CloseController Function

The **CloseController** function closes the handler from OpenController function.

## Syntax

```
BOOL CloseController(  
    HANDLE Handler  
);
```

## Parameters

Handler  
[in]  
Input the device handle before you have opened.

## Return Value

If the function succeeds, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

When you don't want to access this device any more, you need to call this function to release the handle.

## Example

See Example Program #1 (6)

## DisableTouchReport Function

The **DisableTouchReport** function is used to disable/enable report touch event to OS.

## Syntax

```
BOOL DisableTouchReport (  
    HANDLE deviceHandle,  
    BYTE On  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

On

[in]

If **On** is 0, it will enable to report event. If 1, it will disable to report event and save its report data into buffer. After reboot your computer, it will report event again. If 2, disable report event forever.



## Return Value

If the function succeeds, the return value will TRUE. Otherwise, it is FALSE.

## Remarks

If you want to get report data, you need to disable report and call **ReadBuffer** function to get report data.

## Example

See Example Program #1 (4)

## GetReportStatus Function

The **GetReportStatus** function is used to dedicate the status of reporting touch event to OS.

## Syntax

```
BOOL GetReportStatus(  
    HANDLE deviceHandle,  
    );
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

If do report, the return value is FALSE. Otherwise, it is TRUE.

## Remarks

This function is used to query whether has been disabled or not.

## Example

See Example Program #1 (5)

## SetRotation Function

The **SetRotation** function is used to match the orientation of your screen.

## Syntax

```
BOOL SetRotation(  
    HANDLE deviceHandle,  
    BYTE Rotation  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

Rotation

[in]

It depends on your screen. 0: no rotation. 1: 90 degree. 2: 180 degree. 3: 270 degree

## Return Value

If the function succeeds, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

Some kind of screen can be set to rotate a degree to fulfill user's application. So, you can set this parameter to match the screen's orientation.

## Example

Example Program #2 (1)

## GetRotation Function

The **GetRotation** function is used to get the coordinate orientation of the controller.

## Syntax

```
BYTE GetRotation(  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

If the function succeeds, the return value will be the orientation.  
0: 0 degree, 1: 90 degree, 2: 180 degree, and 3: 270 degree.  
Otherwise, it will be 0.

## Remarks

You can use this function to query current coordinate orientation of the device.

## Example

Example Program #2 (2)

## EnableUpSound Function

The **EnableUpSound** function is used to enable/disable generate a sound when user lift or pen up the touch panel.

## Syntax

```
BOOL EnableUpSound (  
    HANDLE deviceHandle,  
    BOOL Enable);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

Enable

[in]

TRUE is to generate sound and FALSE not to generate.

## Return Value

If the function succeeds, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

You can use this function to enable/disable to generate a sound when user lifts up.

## Example

Example Program #3 (1)

## Enable DownSound Function

The **EnableDownSound** function is used to enable/disable to generate a sound when user touch touch panel down first.

## Syntax

```
BOOL EnableDownSound (  
    HANDLE deviceHandle,  
    BOOL Enable);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

Enable

[in]

TRUE is to generate sound and FALSE not to generate.

## Return Value

If the function succeeds, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

You can use this function to enable/disable to generate a sound when user touches down for the first times.

## Example

Example Program #3 (2)

## GetUpSoundStatus Function

The **GetUpSoundStatus** function is used to get the status if generate a sound when user lift finger or pen up.

## Syntax

```
BOOL GetUpSoundStatus (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

If it is enabled to generate up-sound, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

You can get the current setting whether to generate a sound when lift.

## Example

Example Program #3 (3)

## GetDownSoundStatus Function

The **GetDownSoundStatus** function is used to get the status if generate a sound when user first touch down.

## Syntax

```
BOOL GetDownSoundStatus(  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

If it is enabled to generate down sound, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

You can use this function to get the status whether to generate a sound when user touches the panel down for first times.

## Example

Example Program #3 (4)

## CloseCOMPort Function

The **CloseCOMPort** function only works on the RS232 interface. It can release serial port access that touch panel occupies it.

## Syntax

```
BOOL CloseCOMPort (  
    HANDLE deviceHandle,  
    );
```



## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value will be TRUE when success to release the serial port. Otherwise, it will be FALSE.

## Remarks

You can use this function to release the serial port but the device driver still exists. If you want to occupy it again, use **ReopenCOMPort** function.

## Example

Example Program #4

## ReopenCOMPort Function

The **ReopenCOMPort** function only works on the RS232 interface. It can get the access right again.

## Syntax

```
BOOL ReopenCOMPort (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value will be TRUE when success to open the serial port. Otherwise, it will be FALSE.

## Remarks

You can use this function to open the serial port before we call **CloseCOMPort** function to release it.

## Example

Example Program #5

## ChangeCOMPort Function

The **ChangeCOMPort** function only be used in the RS232 interface. It can assign our device driver to open a serial port with specified serial number and baud rate.

## Syntax

```
BOOL ChangeCOMPort(  
    HANDLE deviceHandle,  
    DWORD thePort,  
    DWORD Baudrate
```

```
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

thePort

[in]

Specified the port number, for example, COM1=1, COM2=2,..., COMn=n.

Baudrate

[in]

Set the value as 9600.

## Return Value

The return value will be TRUE when success to open the serial port. Otherwise, it will be FALSE.

## Remarks

You can use this function to open the specific-serial port..

## Example

Example Program #6 (1)

## **CurrentCOMPort Function**

The **CurrentCOMPort** function only is used to query the device driver which serial port is occupied now.

### **Syntax**

```
BYTE CurrentCOMPort (  
    HANDLE deviceHandle,  
    );
```

### **Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

### **Return Value**

The return value will be the number of serial port. Such as, COM1=1, COM2=2, ..., COMn=n.

### **Remarks**

You can use this function to get the serial port number that occupied by device driver. This function works only in serial interface.

### **Example**

See Example Program #6 (2)

See Example Program #12 (3)

## **WriteBuffer Function**

The WriteBuffer function is used to send a command to the device.

### **Syntax**

```
BOOL WriteBuffer (  
    HANDLE deviceHandle,  
    UCHAR *CmdBuf,  
    DWORD Len  
);
```

### **Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

CmdBuf  
[in]  
It contains the command that you want to send.

Len  
[in]  
Command length

### **Return Value**

If the function succeeds, return value will TRUE. Otherwise, it will be FALSE.

## Remarks

You can use this function to send a command to the device. Please, refer to URTC reference guide about the supported commands.

## Example

Example Program #7 (2)

## ReadBuffer Function

The ReadBuffer function is used to receive response data from the device.

## Syntax

```
DWORD ReadBuffer (  
    HANDLE deviceHandle,  
    UCHAR *RespBuf,  
    DWORD Len  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

RespBuf  
[out]  
Receiving buffer.

Len  
[in]  
Specify receiving buffer length

## Return Value

Return value is the size of real response data.

## Remarks

You can use this function to receive response data before sending a command to device driver. Please, refer to URTC reference guide for the responded data.

## Example

Example Program #7 (4)

## ReadLength Function

The ReadLength function is used to check how many data have received in the device driver buffer.

## Syntax

```
DWORD ReadLength (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

Return value is the size that the device driver has received data.

## Remarks

You can use this function to check how many response data have been received.

## Example

Example Program #7 (3)

## FlushReadBuffer Function

The FlushReadBuffer function is used to clear all data in the receiving buffer.

## Syntax

```
BOOL FlushReadBuffer (  
    HANDLE deviceHandle,  
    );
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.



## Return Value

If the function succeeds, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

You can use this function to clear all data on the receiving buffer before sending a command.

## Example

Example Program #7 (1)

## SetCalPoint Function

The SetCalPoint function is used to set which point to collect the calibration data and enter into the linearity mode.

## Syntax

```
BOOL SetCalPoint(  
    HANDLE deviceHandle,  
    BYTE    PointNum,  
    BYTE    Samples,  
    BYTE    CalMode  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

PointNum

[in]

Input the linearity point number that we want to collect. If CalMode is 9, the value should be between 0 and 8. If CalMode is 5, the value should be between 0 and 4.

Samples

[in]

How many samples we want to take.

CalMode

[in]

The linearity mode is either 5 or 9.

## Return Value

If the function succeeds, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

You can use this function to set current linearity mode, 5 or 9, and select which one point to be collected.

## Example

See Example Program #8 (3)

## CalCurrentPointReady Function

The CalCurrentPointReady function is used to query if device driver receive enough data to fill linearity.

## Syntax

```
BOOL CalCurrentPointReady (  
    HANDLE deviceHandle,  
    PULONG AvgPnt  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

AvgPnt  
[out]  
Return the average linearity data.

## Return Value

If the device driver had been ready to collect, the return value will be TRUE. Otherwise, it will be FALSE.

## Remarks

You can use this function to query if it finishes to collect this point data. Otherwise, you need to continue to poll with this function.

## Example

See Example Program #8 (4)

## **IsPenUp Function**

The IsPenUp function is used to query if user had lift pen or finger.

### **Syntax**

```
BOOL IsPenUp (  
    HANDLE deviceHandle,  
);
```

### **Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

AvgPnt  
[out]  
Return the average linearity data.

### **Return Value**

If user lift the pen or finger, the return value will be TRUE.  
Otherwise, it is FALSE.

### **Remarks**

You need to sure that user had lift and then we can start the next point to collect data.

### **Example**

See Example Program #8 (5)

## **StoreCalibrationData Function**

The StoreCalibrationData function is used to save linearity data into touch controller.

### **Syntax**

```
BOOL StoreCalibrationData (  
    HANDLE deviceHandle,  
    );
```

### **Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

### **Return Value**

If it success to save data, the return value will be TRUE.  
Otherwise, it will be FALSE.

### **Remarks**

This function can be used to restore linearity data into the controller.

### **Example**

See Example Program #8 (6)

## **SetDoubleClickSpeed Function**

The SetDoubleClickSpeed function is used to set double click speed into OS parameter.

### **Syntax**

```
BOOL SetDoubleClickSpeed(  
    HANDLE deviceHandle,  
    BYTE Speed  
);
```

### **Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

Speed  
[in]  
Input double click speed to determine to generate double click event. This value is between 1 and 10.

### **Return Value**

If it is successful, the return value will be TRUE. Otherwise, it is FALSE.

### **Remarks**

This speed value is used to determine how long it is between the two clicks.

## Example

See Example Program #9 (1)

## SetDoubleClickArea Function

The SetDoubleClickArea function is used to set double click speed into OS parameter.

## Syntax

```
BOOL SetDoubleClickArea (  
    HANDLE deviceHandle,  
    BYTE Area  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

Area

[in]

Input double click area to determine to generate double click event. This value is between 1 and 6.

## Return Value

If it is successful, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

This area value is used to determine to generate between two clicks.

## Example

See Example Program #9 (2)

## GetDoubleClickSpeed Function

The GetDoubleClickSpeed function is used to get the current double click speed setting.

## Syntax

```
BYTE GetDoubleClickArea (  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value will be the speed level. It will be between 1 and 10.



## Remarks

The function is used to get the current double click speed setting.

## Example

See Example Program #9 (3)

## GetDoubleClickArea Function

The GetDoubleClickSpeed function is used to get the current double click speed setting.

## Syntax

```
BYTE GetDoubleClickArea (  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value will be the area range. It will be between 1 and 6.

## Remarks

The function is used to get the current double click area setting.

## Example

See Example Program #9 (4)

## SetReportMode Function

The SetReportMode function is used to set the report mode.

## Syntax

```
BOOL SetReportMode (  
    HANDLE deviceHandle,  
    BYTE    Mode  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

Mode

[in]

Report mode, 1 = Desktop, 2 = Drawing, and 3 = Button.

## Return Value

If this function is successful, the return value will be TRUE. Otherwise, It will be FALSE.

## Remarks

The function is used to set the reporting style to OS. They are Desktop, Drawing, and Button modes.

## Example

See Example Program #8 (2)

## GetReportMode Function

The GetReportMode function is used to get the current report mode.

## Syntax

```
BYTE GetReportMode (  
    HANDLE deviceHandle,  
    );
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value is between 1 and 3.

## Remarks

The function is used to get the current report mode.

## Example

See Example Program #8 (1)

## SetRightClickTime Function

The SetRightClickTime function is used to set how long to generate a right click event if user touch down and no moving.

## Syntax

```
BOOL SetRightClickTime (  
    HANDLE deviceHandle,  
    LONG    Time  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

Time

[in]

It is the time if user touches down and no move.  
This value is between 400 and 1272 by the step of

**Return Value**

If this function is successful, the return value will be TRUE.  
Otherwise, It will be FALSE.

**Remarks**

This function is used to set the time to generate right click.

**Example**

See Example Program #10 (1)

**GetRightClickTime Function**

The GetRightClickTime function is used to get current the time setting for generating a right click event.

**Syntax**

```
LONG SetRightClickTime (
    HANDLE deviceHandle,
);
```

**Parameters**

deviceHandle  
[in]  
Input the device handle before you have opened with  
**OpenController** function.

## Return Value

The return value is the time to generate right click event.

## Remarks

This function is used to get the current right click time setting.

## Example

See Example Program #10 (2)

## EnableRightClick Function

The EnableRightClick function is used to enable/disable generating right click event.

## Syntax

```
BOOL EnableRightClick (  
    HANDLE deviceHandle,  
    BYTE    Enable  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

Enable  
[in]

If 0, it don't generate right click event. If 1, it like to generate right click event.

## Return Value

If the function is successful to set, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

This function is used to determine whether to generate when device driver receive a right click command from a controller.

## Example

See Example Program #11 (1)

## GetRightClickEnabled Function

The GetRightClickEnabled function is used to get the status whether to generate right click event.

## Syntax

```
BOOL GetRightClickEnabled (  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with

**OpenController** function.

## Return Value

If it need to generate right click, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

This function is used to get the current status whether to generate right click event.

## Example

See Example Program #11 (2)

## GetDriverVersion Function

The GetDriverVersion function is used to get the current version of device driver.

## Syntax

```
BOOL GetDriverVersion (  
    HANDLE deviceHandle,  
    PCHAR version,  
    BYTE Len  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.



version

[out]

Pass the version buffer to copy driver version into this buffer. This buffer size should be more than 20 bytes.

Len

[in]

The size of version buffer.

## Return Value

If this function is successful, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

This function is used to tell user the current version of device driver.

## Example

See Example Program #12 (1)

## GetDriverInterface Function

The GetDriverInterface function is used to get the controller's interface, either USB or RS232.

## Syntax

```
INT GetDriverInterface(  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

If the return value is 1, the interface is RS232. If return value is 2, the interface is USB.

## Remarks

This function is used to tell user the interface is RS232 or USB.

## Example

See Example Program #12 (2)

## AskToSetDisplayNumber Function

The AskToSetDisplayNumber function is used to ask all the device drivers on the computer enter matching monitor mode.

## Syntax

```
BOOL AskToSetDisplayNumber(  
    BYTE Numth  
);
```

## Parameters

Numth

[in]

To matching Numth display on the local computer.

## Return Value

If this function is successful, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

This function is used to ask all device drivers enter to matching the Numth screen. If user touches someone screen, the relative controller that locates on this screen will sense the touch and tell device driver manager to save Numth value into this controller.

## Example

See Example Program #13 (2)

## QueryDisplayBeTouched Function

The QueryDisplayBeTouched function is used to query whether someone screen is touched. If do, finish this order setting.

## Syntax

```
BOOL QueryDisplayBeTouched (  
    VOID  
);
```

## Parameters

NULL

## Return Value

If the return value is TRUE, finish to set this order. Otherwise, programmer needs to continue to poll until TRUE.

## Remarks

This function is used to poll whether there is a touch event happening. If there is, find which one be touched and save Numth into the controller.

## Example

See Example Program #13 (5)

## GetDisplayNumber Function

The GetDisplayNumber function is used to query which screen does the controller locate.

## Syntax

```
BYTE  GetDisplayNumber(  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

This return value is the display number that the controller locates on. The return value should be between 0 and 3.

## Remarks

This function supports to get the display number that the controller locates. It can be used in multimonitor utility.

## Example

See Example Program #12 (4)

## GetExistDisplays Function

The GetExistDisplays function is used to query how many screens there are on your computer.

## Syntax

```
BYTE GetExistDisplays (  
    VOID  
);
```

## Parameters

NULL

## Return Value

This return value is the screen number on your computer.

## Remarks

This function enumerates all displays on the computer and builds a display list. You can use the index of display list to get display name.

## Example

See Example Program #13 (5)

## GetDisplayName Function

The GetDisplayName function gets the display name from windows OS.

## Syntax

```
BOOL  GetDisplayName (
    BYTE  DisplayNum,
    CHAR  *devName,
    BYTE  Len
);
```

## Parameters

DisplayNum  
[in]  
The index of display list.

devName  
[out]  
This buffer is used to store display name.

Len  
[in]  
You need to give the size of devName buffer.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

This function returns the relative display name. And programmer can use the name to get the relative display information with GetDisplayArea function.

## Example

See Example Program #13 (3)

## GetDisplayArea Function

The GetDisplayArea function gets the display name from windows OS.

## Syntax

```
BOOL  GetDisplayArea(  
    CHAR  *devName,  
    RECT  *rect  
);
```

## Parameters

devName  
[in]  
Specify the display to get display information.

rect

[out]  
Return the work area of the display.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

This function supports programmer to get monitor's work area that maps to windows coordinate.

## Example

See Example Program #13 (4)

## GetMeasureTime Function

The GetMeasureTime function gets the time that spend to report a coordinate data.

## Syntax

```
UINT GetMeasureTime(  
    HANDLE deviceHandle  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with  
**OpenController** function.



## Return Value

The return value is the time to report a coordinate data.

## Remarks

This function supports programmer to get the report rate. For example, If the return is 8, the report rate is 125 pps.

## Example

See Example Program #14 (2)

## SetMeasureTime Function

The SetMeasureTime function sets the time that spends to report a coordinate data.

## Syntax

```
BOOL SetMeasureTime(  
    HANDLE deviceHandle,  
    UINT    Time  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

Time  
[in]  
Spends the time to report a coordinate.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

If you set the report rate is 125 pieces per second. The time argument will be 8 ms.

## Example

See Example Program #14 (1)

## GetMeasureDelay Function

The GetMeasureDelay function gets the current setting time that spends to wait the touch sensor being stable.

## Syntax

```
UINT GetMeasureDelay (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with  
**OpenController** function.

## Return Value

The return value is the current delay time to measure.

## Remarks

This function gets the current setting time that spends to wait the touch sensor being stable.

## Example

See Example Program #15 (2)

## SetMeasureDelay Function

The SetMeasureDelay function sets the waiting time that spends to wait the touch sensor being stable.

## Syntax

```
BOOL SetMeasureDelay (  
    HANDLE deviceHandle,  
    UINT MeasureDelay  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

MeasureDelay  
[in]

Delay the quantity of MeasureDelay instruction cycle time.

## Return Value

If this function is successful, the return value is TRUE. Otherwise, it is FALSE.

## Remarks

The SetMeasureDelay function sets the waiting time that spends to wait the touch sensor being stable. The value will match with the measuring time.

## Example

See Example Program #15 (1)

## GetInfocus Function

The GetInfocus function gets the current in-focus area.

## Syntax

```
UINT GetInfocus (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value is the current in-focus value.

## Remarks

The `GetInfocus` function gets the current in-focus value.

## Example

See Example Program #16 (2)

## SetInfocus Function

The `SetInfocus` function sets the in-focus area to optimum your touch solution.

## Syntax

```
BOOL SetInfocus (  
    HANDLE deviceHandle,  
    UINT Infocus  
);
```

## Parameters

`deviceHandle`  
[in]  
Input the device handle before you have opened with **OpenController** function.

`Infocus`  
[in]

In-focus value.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

The SetInfocus function sets the in-focus area to optimum your touch solution.

## Example

See Example Program #16 (1)

## GetOutfocus Function

The GetOutfocus function gets the current out-focus area.

## Syntax

```
UINT GetOutfocus (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with  
**OpenController** function.

## Return Value

The return value is the current out-focus value.

## Remarks

The GetOutfocus function gets the current out-focus value.

## Example

See Example Program #17 (2)

## SetInfocus Function

The SetOutfocus function sets the out-focus area to optimum your touch solution.

## Syntax

```
BOOL SetOutfocus (  
    HANDLE deviceHandle,  
    UINT Outfocus  
);
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with **OpenController** function.

Outfocus  
[in]  
Outfocus value.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

The SetOutfocus function sets the outfocus value to optimum your touch solution.

## Example

See Example Program #17 (1)

## SetCalibrationPoint Function

The SetCalibrationPoint function is used to set which point to collect the 3pts calibration data.

## Syntax

```
BOOL SetCalibrationPoint (  
    HANDLE deviceHandle,  
    BYTE   PointNum,  
    );
```

## Parameters

deviceHandle  
[in]  
Input the device handle before you have opened with  
**OpenController** function.



PointNum

[in]

Input the 3pts calibration point number that we want to collect. The value should be between 0 and 2. If the value sets as 0xFF, it means to cancel 3pts calibration mode.

## Return Value

If the function succeeds, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

You can use this function to set current point that the collecting data to represent it.

## Example

See Example Program #18 (1)

## GetCalibrationData Function

The **CalCurrentPointReady** function is used to query if device driver receive enough data to fill calibration need.

## Syntax

```
BOOL GetCalibrationData (  
    HANDLE deviceHandle,  
    PULONG AvgPnt  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

AvgPnt

[out]

Return the average calibration data to represent the point.

## Return Value

If the device driver is ready to collect, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

You can use this function to query whether it finishes to collect this point data. If it is ready, the AvgPnt argument will return the average data.

## Example

See Example Program #18 (2)

## SaveCalibrationData Function

The SaveCalibrationData function is used to save calibration data into device driver.

## Syntax

```
BOOL SaveCalibrationData (  
    HANDLE deviceHandle,  
    PLONG pLong,  
    BYTE cbSize  
);
```

## Parameters

deviceHandle	[in] Input the device handle before you have opened with <b>OpenController</b> function.
pLong	[in] It is the pointer to point to the calibration data array.
cbSize	[in] It is the size of the calibration data array. This value should be 24.

## Return Value

If it success to save data, the return value will be TRUE.  
Otherwise, it is FALSE.

## Remarks

This function can be used to restore calibration data into device driver.

## Example

See Example Program #18 (4)

## Set\_Sound\_Type Function

The Set\_Sound\_Type function is used to set the sound type when user touch down or lift up.

## Syntax

```
BOOL Set_Sound_Type(  
    BYTE type  
);
```

## Parameters

type

[in]  
If 1, the sound type is buzzer. If 2, the sound type is audio type.

## Return Value

If this function is successful, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

This function can be used to set to generate either buzzer or audio sound.

## Example

See Example Program #19 (1)

## Get\_Sound\_Type Function

The Get\_Sound\_Type function is used to get the current sound type.

## Syntax

```
BYTE Get_Sound_Type(  
    VOID  
);
```

## Parameters

NULL

## Return Value

The return value will be 1 or 2.

## Remarks

This function can be used to get the current sound type.

## Example

See Example Program #19 (2)

## **Get\_Manual\_RightClick\_Status Function**

The Get\_Manual\_RightClick\_Status function is used to get whether Rbutton.exe is launched or not.

### **Syntax**

```
BOOL Get_Manual_RightClick_Status (  
    VOID  
);
```

### **Parameters**

NULL

### **Return Value**

If Rbutton.exe is launched, the return value will be TRUE.  
Otherwise, it is FALSE.

### **Remarks**

This function can be used to detect whether the Rbutton.exe have been launched.

## **Set\_MRightClick\_App Function**

The Set\_MRightClick\_App function is used to launch or close Rbutton.exe.

### **Syntax**

```
BOOL Set_MRightClick_App (  
    BOOL On_Off
```

```
);
```

## Parameters

On\_Off

[in]

If TRUE, launch the RButton.exe. If FALSE, close the Rbutton.exe.

## Return Value

If this function is successful, the return value will be TRUE. Otherwise, it is FALSE.

## Remarks

This function is used to launch or close the Rbutton.exe. The Rbutton.exe is a utility to inform device driver to generate a right click for the next touch down and up.

## GetTouchSensitivity Function

The GetTouchSensitivity function gets the current touch sensitivity level.

## Syntax

```
BYTE GetTouchSensitivity (  
    HANDLE deviceHandle,  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.

## Return Value

The return value should be between 0 and 5 to show the current level.

## Remarks

This function is used to get current touch sensitivity level.

## Example

See Example Program #20 (2)

## SetTouchSensitivity Function

The SetTouchSensitivity function sets touch force level.

## Syntax

```
BOOL SetTouchSensitivity (  
    HANDLE deviceHandle,  
    BYTE Level  
);
```

## Parameters

deviceHandle

[in]

Input the device handle before you have opened with **OpenController** function.



Level

[in]

0: Low, 1: Normal, 2: More, 3: Very, 4: More Very,  
5: Most. The default is 0.

## Return Value

If this function is successful, the return value is TRUE.  
Otherwise, it is FALSE.

## Remarks

This function sets the touch sensitivity level value to set the touch force. The default value is 0.

## Example

See Example Program #20 (1)

## Example Program #1

```
/* This example is showing how to use GetExistControllers, GetControllerName, OpenController,
CloseController, DisableTouchReport, .*/
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;

    //Get existing controller in the computer
    tcNum = GetExistControllers(); //(1)

    if (tcNum == 0) {
```

```

printf("No controller in the computer. <Press any key>\n");
getch();
goto Exit;
}
else{
printf("*****   Devices *****\n");
for (tcIdx=0; tcIdx<tcNum; tcIdx++)
{
    // get controller's name and use it to get device handle.
    GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME); //(2)
    printf("\t %d) %s\n", tcIdx+1, DeviceName);
}
printf("*****   Devices *****\n\n");
printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
scanf("%d", &SelectedItem);
}
// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName); //(3)
if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Select the type of report: \n");
printf("\t 0) Report event. \n");
printf("\t 1) Temporarily disable report event until next boot. \n");
printf("\t 2) Disable report event forever. \n");
printf("\t Select the type of report<0, 1, 2>: \n");
scanf("%d", &SelectedItem);
status = DisableTouchReport (hController, (BYTE) SelectedItem); //(4)

status = GetReportStatus(hController); //(5)
switch (status){
    case TRUE:
        printf("Current is enabled to report.\n");
        break;
    case FALSE:
        printf("Current is disabled to report.\n");

```

```

        break;
    }
Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController); //(6)
    return;

}

```

## **Example Program #2**

```

/* This example is showing how to use SetRotation, GetRotation. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    BYTE   Orientation;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }

    // get the selected controller name from the index of device list.

```

```

GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

```

```

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

```

```

printf("\t Select the type of report: \n");
printf("\t 0) Don't rotate. \n");
printf("\t 1) Rotate 90 degree. \n");
printf("\t 2) Rotate 180 degree. \n");
printf("\t 3) Rotate 270 degree. \n");

```

```

printf("\t Select the type of rotation<0, 1, 2, 3>: \n");
scanf("%d", &SelectedItem);
status = SetRotation( hController, (BYTE) SelectedItem); //(1)

```

```

Orientation = GetRotation( hController); //(2)

```

```

Switch (Orientation){
    case 0:
        printf("\tCurrent is no rotation\n");
        break;
    case 1:
        printf("\t Current rotates 90 degree.\n");
        Break;
    case 2:
        printf("\t Current rotates 180 degree.\n");
        Break;
    case 3:
        printf("\t Current rotate 270 degree.\n");
        Break;
}

```

Exit:

```

if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);

```

```

return;

}

```

### **Example Program #3**

```

/* This example is showing how to use EnableUpSound, EnableDownSound,
   GetUpSoundStatus, and GetUpSoundStatus.*/
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   status;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.
    hController=OpenController(DeviceName);

```

```

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t 0) Disable to generate a up sound.\n");
printf("\t 1) Enable to generate a up sound.\n");
scanf("%d", &SelectedItem);
switch (SelectedItem){
    case 0:
        status = EnableUpSound (hController, FALSE); //(1)
        break;
    case 1:
        status = EnableUpSound (hController, TRUE);
        break;
}

printf("\t 0) Disable to generate a down sound.\n");
printf("\t 1) Enable to generate a sown sound.\n");
scanf("%d", &SelectedItem);
switch (SelectedItem){
    case 0:
        status = EnableDownSound (hController, FALSE); //(2)
        break;
    case 1:
        status = EnableDownSound (hController, TRUE);
        break;
}

status = GetUpSoundStatus (hController); //(3)
if (status==TRUE){
    printf("\t Enable to generate up sound.");
}
else{
    printf("\t Enable to generate up sound.");
}

status = GetDownSoundStatus (hController); //(4)
if (status==TRUE){
    printf("\t Enable to generate down sound.");
}

```

```

    }
    else{
        printf("\t Enable to generate down sound.");
    }
}

```

Exit:

```

    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #4**

```

/* This example is showing how to use CloseCOMPort. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   status;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    }
}

```

```

        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.
    hController=OpenController(DeviceName);
    if (hController==INVALID_HANDLE_VALUE){
        printf("Can't open the controller.<Press any key>");
        getch();
        goto Exit;
    }

    status = CloseCOMPort (hController);
    if (status == TRUE) {
        printf("\t Success to close serial port.\n");
    }
    else{
        printf("\t Fail to close serial port.\n");
    }
}

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #5**

```

/* This example is showing how to use ReopenCOMPort. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   status;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

```



```

if (tcNum == 0) {
    printf("No controller in the computer. <Press any key>\n");
    getch();
    goto Exit;
}
else{
    printf("*****   Devices *****\n");
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
    {
        // get controller's name and use it to get device handle.
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
    }
    printf("*****   Devices *****\n\n");
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    scanf("%d", &SelectedItem);
}
// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);
if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

status = ReopenCOMPort (hController);
if (status == TRUE) {
    printf("\t Success to open serial port again.\n");
}
else{
    printf("\t Fail to close serial port again.\n");
}

Exit:
if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
return;

}

```

## **Example Program #6**

```
/* This example is showing how to use ChangeCOMPort, and CurrentCOMPort.*/
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   status;
    BYTE   bPort;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.
    hController=OpenController(DeviceName);

    if (hController==INVALID_HANDLE_VALUE){
        printf("Can't open the controller.<Press any key>");
        getch();
        goto Exit;
    }
}
```

```

    }

    // open com1 with the baudrate being 9600
    status = ChangeCOMPort ( hController, 1, 9600); //(1)
    if (status == TRUE) {
        printf("\t Success to open serial port.\n");
    }
    else{
        printf("\t Fail to close serial port.\n");
    }

    // get the used serial port number.
    bPort = CurrentCOMPort( hController ); //(2)
    printf("\t COM%d is occupied by the device driver.\n", hPort);

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #7**

```

/* This example is showing how to use FlushReadBuffer, WriteBuffer, ReadLength,
and ReadBuffer. */
void main(void)
{
    int        SelectedItem;
    BYTE       tcNum, tcIdx;
    CHAR       DeviceName[MAX_DEVICE_NAME];
    UCHAR      CmdBuf[5], RespBuf[128]; //Max return buffer is 128
    BOOL       Status;
    DWORD      rLen;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
    }
}

```

```

    getch();
    goto Exit;
}
else{
    printf("*****   Devices *****\n");
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
    {
        // get controller's name and use it to get device handle.
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
    }
    printf("*****   Devices *****\n\n");
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    scanf("%d", &SelectedItem);
}
// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}
// Send R command
CmdBuf[0]=0x01;
CmdBuf[1]='R';
CmdBuf[2]=0x0D;
//Clear receiveing buffer
FlushReadBuffer (hController); //(1)

Status = WriteBuffer (hController, CmdBuf, 3); //(2)
If (Status==TRUE){
    Sleep(50); //Wait for device driver to prepare return data.
    rLen = ReadLength( hController ); //(3)
    if (rLen>0){
        rLen = ReadBuffer( hController, RespBuf, rLen); //(4)
        printf("\tResponse Data:\n\t");
        for(l=0; l<rLen; l++) printf("%02X ", RespBuf[l]);
        printf("\n");
    }
}

```

```

    }
    else{
        printf("\tNo Response Data.\n");
    }
}
else{
    printf("\tError to send command\n");
}

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #8**

```

/* This example is showing how to use GetReportMode , SetReportMode , ReadLength, and
ReadBuffer to do 5 or 9 pts linearity. */
//

#include "stdafx.h"
#include "utpapi.h"
#include "resource.h"

#define MAX_LOADSTRING    100
#define MAX_NAME          20
#define MAX_TIMES         3
#define SAMPLES           60
#define POINTS_9          9
#define POINTS_5          5

// Global Variables:
HINSTANCE hInst;                // current instance
TCHAR szTitle[MAX_LOADSTRING]; // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING]; // The
title bar text
BYTE CurrPoint = 0;
BYTE CaliMode, oldMode;
HANDLE hDevice = INVALID_HANDLE_VALUE;

```

```
BOOL SendCommand(HANDLE hHandle, UCHAR *Cmd, BYTE Len);
```

```
// Forward declarations of functions included in this code module:
```

```
ATOM MyRegisterClass(HINSTANCE hInstance);
```

```
BOOL InitInstance(HINSTANCE, int);
```

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
void DrawGridLine(HDC hdc);
```

```
void DrawCaliedPnt(HDC hdc, int Num);
```

```
void DrawCaliPnt(HDC hdc, BYTE Num, int Mode);
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
```

```
                    HINSTANCE hPrevInstance,
```

```
                    LPSTR lpCmdLine,
```

```
                    int nCmdShow)
```

```
{
```

```
    // TODO: Place code here.
```

```
    MSG msg;
```

```
    HACCEL hAccelTable;
```

```
    BYTE ExistDev=0;
```

```
    BOOL status;
```

```
    CHAR devName[MAX_NAME];
```

```
    UCHAR CmdBuf[12];
```

```
    // Initialize global strings
```

```
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
    LoadString(hInstance, IDC_CALIEXAMPLE, szWindowClass, MAX_LOADSTRING);
```

```
    ExistDev=GetExistControllers();
```

```
    if (!ExistDev) {
```

```
        MessageBox(NULL, "No device exist.", "Error!!", MB_OK);
```

```
        return FALSE;
```

```
    }
```

```
    else{
```

```
        status=GetControllerName(1, devName, MAX_NAME); //get the first controller's name
```

```
        if (!status) return FALSE;
```

```
        hDevice=OpenController(devName);
```

```
        if (hDevice==INVALID_HANDLE_VALUE) {
```

```
            MessageBox(NULL, "Error to open the first controller.", "Error!!", MB_OK);
```

```
            return FALSE;
```

```

    }
}

//Enter MS Mode
oldMode = GetReportMode(hDevice); //(1)
SetReportMode(SetReportMode(hDevice, 2);
//Enter Calibration Mode
CmdBuf[0]=0x01;
CmdBuf[1]='C';
CmdBuf[2]='R';
CmdBuf[3]=0x0D;
status = SendCommand( hDevice, CmdBuf, 4);

if (!status){ //Error to enter calibration mode.
    if (hDevice!=INVALID_HANDLE_VALUE) {
        SetReportMode(SetReportMode(hDevice, oldMode);
        CloseController(hDevice);
    }
    return FALSE;
}

MyRegisterClass(hInstance);

// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
    if (hDevice!=INVALID_HANDLE_VALUE)
        CloseController(hDevice);
    return FALSE;
}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CALIEXAMPLE);

// Main message loop:
while (GetMessage(&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

```

```

    }

    if (hDevice!=INVALID_HANDLE_VALUE) {
        SetReportMode(SetReportMode(hDevice, oldMode);
        CloseController(hDevice);
    }
    return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//      This function and its usage is only necessary if you want this code
//      to be compatible with Win32 systems prior to the 'RegisterClassEx'
//      function that was added to Windows 95. It is important to call this function
//      so that the application will get 'well formed' small icons associated
//      with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = (WNDPROC)WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_CALIEXAMPLE);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH) GetStockObject(GRAY_BRUSH);
    wcex.lpszMenuName   = NULL;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

```



```

    return RegisterClassEx(&wcex);
}

//
//  FUNCTION: InitInstance(HANDLE, int)
//
//  PURPOSE: Saves instance handle and creates main window
//
//  COMMENTS:
//
//      In this function, we save the instance handle in a global variable and
//      create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    int Screen_Height, Screen_Width;
    LONG    Value = 0;
    BOOLEAN tt=FALSE;

    hInst = hInstance; // Store instance handle in our global variable

    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

    hWnd = CreateWindow(szWindowClass, NULL, WS_POPUP,
        0, 0, Screen_Width, Screen_Height, NULL, NULL, hInstance, NULL);

    if (!hWnd)    return FALSE;

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

//

```

```

// FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
// PURPOSE: Processes messages for the main window.
//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    int wmlId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    static BOOL IsGreen=FALSE;
    BOOL ReadyStatus, UpStatus;
    ULONG AvgPnt[2];

    switch (message)
    {
        case WM_CREATE:
            CurrPoint = 0;
            CaliMode = POINTS_9; //Do Nine-points calibration
                                // If Five-points calibration, set POINTS_5
                                //start to collect the first point value
            SetCalPoint(hDevice, CurrPoint, SAMPLES, CaliMode); //(3)
            SetTimer( hWnd, 0x240, 500, NULL);
            break;
        case WM_KEYDOWN:
            wmlId = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            switch(wmlId){
                case VK_ESCAPE: //Exit the program
                {
                    UCHAR CmdBuf[8];
                    CmdBuf[0]=0x01; CmdBuf[1]='R'; CmdBuf[2]=0x0D;
                    SendCommand(hDevice, CmdBuf, 3);
                    DestroyWindow(hWnd);
                }
                break;
            }
    }
}

```

```

    }

    break;
case WM_PAINT:
    hdc = BeginPaint(hWnd, &ps);
    DrawGridLine(hdc);
    EndPaint(hWnd, &ps);
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_TIMER:

    if (wParam==0x240){
        hdc = GetDC( hWnd);

        ReadyStatus=CalCurrentPointReady(hDevice, AvgPnt); //(4)
        if (ReadyStatus==TRUE) { //is ready to collect samples.
            DrawCaliPnt(hdc, (int)CurrPoint);
            Beep(1000, 10);
            UpStatus=IsPenUp(hDevice); //(5)
            if (UpStatus==TRUE){
                Beep(2000, 20);
                CurrPoint++;
                if (CurrPoint>=CaliMode){
                    // store calibration data
                    StoreCalibrationData(hDevice); //(6)
                    DestroyWindow(hWnd);
                }
                else SetCalPoint(hDevice, CurrPoint, SAMPLES, CaliMode);
            }
        }
    }

    if (!ReadyStatus){
        if (IsGreen==FALSE){ //Show Green
            IsGreen=TRUE;
            DrawCaliPnt(hdc, CurrPoint, 1);
        }
        else{ //Show Gray
            IsGreen=FALSE;
            DrawCaliPnt(hdc, CurrPoint, 0);

```

```

        }
    }

    ReleaseDC( hWnd, hdc );
}
break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

```

void DrawCaliedPnt(HDC hdc, int Num)

```

```

{
    int Screen_Width, Screen_Height;
    HPEN hNewPen, hOldPen;
    int CXY[9][2]={ {2048, 2048}, {3968, 2048}, {3968, 128}, {2048, 128}, \
                    {128, 128}, {128, 2048}, {128, 3968}, {2048, 3968}, \
                    {3968, 3968}};

```

```

    if (CaliMode==POINTS_5) Num=2*Num;
    else if(CaliMode==POINTS_3) Num=(Num+1)*2;

```

```

    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

```

```

    hNewPen = CreatePen( PS_SOLID, 2, RGB(255, 255, 0)); //

```

```

    hOldPen = (HPEN)SelectObject (hdc, hNewPen);

```

```

    Arc( hdc, ((CXY[Num][0]*Screen_Width)>>12)-20, // x-coord of rectangle's upper-left corner
          ((CXY[Num][1]*Screen_Height)>>12)-20,
          ((CXY[Num][0]*Screen_Width)>>12)+20, // x-coord of rectangle's lower-right
corner
          ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of rectangle's lower-right
corner
          ((CXY[Num][0]*Screen_Width)>>12), // x-coord of first radial ending point
          ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of first radial ending point
          ((CXY[Num][0]*Screen_Width)>>12), // x-coord of second radial ending
point

```

```

        ((CXY[Num][1]*Screen_Height)>>12)+20 // y-coord of second radial ending
point
    );

    SelectObject (hdc, hOldPen);
    DeleteObject (hNewPen);

}

```

```

void DrawGridLine(HDC  hdc)
{
    POINT  PrePoint;
    HPEN    hNewPen, hOldPen;
    int      Screen_Width, Screen_Height;

    // Get System Screen Width, Height
    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

    hNewPen = CreatePen( PS_SOLID, 2, RGB(255, 255, 255)); //White color
    hOldPen = (HPEN)SelectObject (hdc, hNewPen);
    //
    MoveToEx( hdc, 0, (128*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width, (128*Screen_Height)>>12);

    MoveToEx( hdc, 0, (3968*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width, (3968*Screen_Height)>>12);

    MoveToEx( hdc, (128*Screen_Width)>>12, 0, &PrePoint);
    LineTo(hdc, (128*Screen_Width)>>12, Screen_Height);

    MoveToEx( hdc, (3968*Screen_Width)>>12, 0, &PrePoint);
    LineTo(hdc, (3968*Screen_Width)>>12, Screen_Height);

    MoveToEx( hdc, Screen_Width>>1, (128*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width>>1, (3968*Screen_Height)>>12);

    MoveToEx( hdc, (128*Screen_Width)>>12, Screen_Height>>1, &PrePoint);
    LineTo(hdc, (3968*Screen_Width)>>12, Screen_Height>>1);
}

```

```

        SelectObject (hdc, hOldPen);
        DeleteObject (hNewPen);

    }

void DrawCaliPnt(HDC hdc,  BYTE Num, int Mode)
{
    int Screen_Width, Screen_Height;
    HPEN  hNewPen, hOldPen;
    int CXY[9][2]={2048, 2048}, {3968, 2048}, {3968, 128}, {2048, 128}, \
                  {128, 128}, {128, 2048}, {128, 3968}, {2048, 3968}, \
                  {3968, 3968}};

    if (CaliMode==POINTS_5) Num=2*Num;
    else if(CaliMode==POINTS_3) Num=(Num+1)*2;
    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );
    if (Mode ==1) hNewPen = CreatePen( PS_SOLID, 2, RGB(0, 255, 0)); //
    else hNewPen = CreatePen( PS_SOLID, 2, RGB(200, 2000, 200)); //
    hOldPen = (HPEN)SelectObject (hdc, hNewPen);

    Arc( hdc, ((CXY[Num][0]*Screen_Width)>>12)-20, // x-coord of rectangle's upper-left corner
        ((CXY[Num][1]*Screen_Height)>>12)-20,
        ((CXY[Num][0]*Screen_Width)>>12)+20, // x-coord of rectangle's lower-right
corner
        ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of rectangle's lower-right
corner
        ((CXY[Num][0]*Screen_Width)>>12), // x-coord of first radial ending point
        ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of first radial ending point
        ((CXY[Num][0]*Screen_Width)>>12), // x-coord of second radial ending
point
        ((CXY[Num][1]*Screen_Height)>>12)+20 // y-coord of second radial ending
point
        );

    SelectObject (hdc, hOldPen);
    DeleteObject (hNewPen);

}

```

```

BOOL SendCommand(HANDLE hHandle, UCHAR *Cmd, BYTE Len)
{
    BYTE    i;
    UCHAR    RspBuf[12];
    DWORD    gLen, RLen;

    for (i=0; i<MAX_TIMES; i++){
        FlushReadBuffer(hHandle);
        WriteBuffer(hHandle, Cmd, (DWORD) Len);
        Sleep(100);

        gLen = ReadLength(hHandle);
        if (gLen!=3) continue;
        RLen = ReadBuffer(hHandle, RspBuf, gLen);
        if (RLen != 3) continue;
        else{
            if (RspBuf[0]==0x01 &&
                RspBuf[1]==0x30 &&
                RspBuf[2]==0x0D) return TRUE;
        }
    }

    return FALSE;
}

```

## **Example Program #9**

```

/* This example is showing how to use SetDoubleClickSpeed,
SetDoubleClickArea , GetDoubleClickSpeed , GetDoubleClickArea .*/
void main(void)
{
    int    SelectedItem;
    BYTE    tcNum, tcIdx;
    CHAR    DeviceName[MAX_DEVICE_NAME];
    BOOL    Status;
    BYTE    Speed, Area;

    //Get existing controller in the computer

```

```

tcNum = GetExistControllers();

if (tcNum == 0) {
    printf("No controller in the computer. <Press any key>\n");
    getch();
    goto Exit;
}
else{
    printf("*****   Devices   *****\n");
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
    {
        // get controller's name and use it to get device handle.
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
    }
    printf("*****   Devices   *****\n\n");
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    scanf("%d", &SelectedItem);
}

// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Input the double click speed<1~10>: ");
scanf("%d", &SelectedItem);
status = SetDoubleClickSpeed(hController, (BYTE) SelectedItem); //(1)

printf("\t Input the double click area<1~6>: ");
scanf("%d", &SelectedItem);

status = SetDoubleClickArea(hController, (BYTE) SelectedItem); //(2)

Speed = GetDoubleClickSpeed(hController); //(3)
printf("\t Current double click speed is %d\n", Speed);

```



```

        Area = GetDoubleClickArea(hController); //(4)
        printf("\t Current double click area is %d\n", Area);

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #10**

```

/* This example is showing how to use SetRightClickTime, GetRightClickTime. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    LONG   rTime;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }
}

```

```

}
// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Input the right click time<400~1272>: ");
scanf("%d", &SelectedItem);
status = SetRightClickTime(hController, LONG) SelectedItem); //(1)

rTime = GetRightClickTime(hController); //(2)
printf("\t Current right click time is %d\n", rTime);

Exit:
if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
return;

}

```

## **Example Program #11**

```

/* This example is showing how to use EnableRightClick, GetRightClickTime . */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
    }
}

```

```

    getch();
    goto Exit;
}
else{
    printf("*****   Devices *****\n");
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
    {
        // get controller's name and use it to get device handle.
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
    }
    printf("*****   Devices *****\n\n");
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    scanf("%d", &SelectedItem);
}
// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t 0) Disable right click\n ");
printf("\t 1) Enable right click\n ");
printf("\t Input <0 or 1>\n");
scanf("%d", &SelectedItem);
EnableRightClick(hController, (BYTE) SelectedItem); //(1)

Status = GetRightClickEnabled(hController); //(2)
if (Status==TRUE){
    printf("\t Right click is enabled\n ");
}
else{
    printf("\t Right click is disabled\n ");
}
}

```

Exit:

```

    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #12**

```

/* This example is showing how to use GetDriverVersion, GetDriverInterface, and
CurrentCOMPort, GetDisplayNumber.*/
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    CHAR   Version[128];
    Int     Interface;
    BYTE   bPort, ScreenNo;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.

```

```

GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

GetDriverVersion(hController, Version, 128) //(1)
Printf("\tDriver Version: %s\n", version);
Interface = GetDriverInterface(hController); //(2)
Switch(Interface){
    case 1:
        bPort = CurrentCOMPort(hController); //(3)
        printf("\tInterface is COM%d\n", bPort);
        break;
    case 2:
        printf("\tInterface is USB\n");
        break;
}
ScreenNo = GetDisplayNumber(hController); //(4)
Printf("\tLocates on Display%d\n", ScreenNo);
Exit:
if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
return;

}

```

### **Example Program #13**

```

// Monitor_Matching.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "resource.h"
#include "Utpapi.h"

```

```

#define MAX_LOADSTRING 100

#define BOLD 1
#define ITALIC 2
#define ULINE 4

// Global Variables:
HINSTANCE hInst; // current instance
int g_nCmdShow;
TCHAR szWindowClass[MAX_LOADSTRING]; // The
title bar text
BYTE Cur_Display, Total_Display, OldMode;

// Forward declarations of functions included in this code module:
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HFONT GetAFont(int fnFont, LONG Height, LONG Width);

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    // TODO: Place code here.
    MSG msg;
    BYTE Total_Ctrls, i;
    CHAR DevName[20];

    // Initialize global strings
    LoadString(hInstance, IDC_MONITOR_MATCHING, szWindowClass, MAX_LOADSTRING);

    Total_Ctrls = GetExistControllers();
    for (i=1; i<=Total_Ctrls; i++){
        HANDLE hHandle = 0;
        memset(DevName, 0, sizeof(DevName));
        GetControllerName(i, DevName, (BYTE)sizeof(DevName));
        hHandle = OpenController(DevName);
        if (hHandle!=0){
            OldMode = GetReportMode(hHandle);
            SetReportMode(hHandle, 2);
        }
    }
}

```

```

        CloseController(hHandle);
    }
}

Cur_Display = 1;
Total_Display = GetExistDisplays(); //(1)

if (Total_Display<1) return FALSE;

AskToSetDisplayNumber( Cur_Display-1 ); //based on zero (2)

MyRegisterClass(hInstance);

// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
    return FALSE;
}

// Main message loop:
while (GetMessage(&msg, NULL, 0, 0))
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}

Total_Ctrls = GetExistControllers();
for (i=1; i<=Total_Ctrls; i++){
    HANDLE hHandle=0;
    memset(DevName, 0, sizeof(DevName));
    GetControllerName( i, DevName, (BYTE)sizeof(DevName));
    hHandle = OpenController(DevName);
    if (hHandle!=0){
        SetReportMode(hHandle, OldMode);
        CloseController(hHandle);
    }
}

return msg.wParam;
}

```

```

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//      This function and its usage is only necessary if you want this code
//      to be compatible with Win32 systems prior to the 'RegisterClassEx'
//      function that was added to Windows 95. It is important to call this function
//      so that the application will get 'well formed' small icons associated
//      with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = (WNDPROC)WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_MONITOR_MATCHING);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH) GetStockObject(GRAY_BRUSH);
    wcex.lpszMenuName   = NULL;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

    return RegisterClassEx(&wcex);
}

//
//  FUNCTION: InitInstance(HANDLE, int)
//
//  PURPOSE: Saves instance handle and creates main window
//
//  COMMENTS:

```



```

//
//      In this function, we save the instance handle in a global variable and
//      create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND    hWnd;
    int      Screen_Height, Screen_Width;
    CHAR     devName[50];
    RECT     rect;

    hInst = hInstance; // Store instance handle in our global variable

    memset(devName, 0, sizeof(devName));
    GetDisplayName(Cur_Display, devName, (BYTE) sizeof(devName)); //(3)
    GetDisplayArea(devName, &rect); //4

    Screen_Width = (rect.right-rect.left);
    Screen_Height = (rect.bottom-rect.top);

    hWnd = CreateWindow( szWindowClass, NULL, WS_POPUP,
                        rect.left, rect.top, Screen_Width, Screen_Height,
                        NULL, NULL, hInst, NULL);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.

```

```

//
// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    int wmlId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    HFONT hfontOld, hfontNew; // handle to old font
    static UINT_PTR theTimer;

    switch (message)
    {
        case WM_CREATE:
            theTimer = SetTimer( hWnd, 0x240, 500, NULL);
            break;
        case WM_KEYDOWN:
            wmlId = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            switch(wmlId){
                case VK_ESCAPE: //Exit the program
                {
                    int Screen_Width, Screen_Height;
                    RECT rect;
                    CHAR devName[30];

                    Cur_Display++;
                    if (Cur_Display<=Total_Display) {
                        KillTimer( hWnd, theTimer );
                        AskToSetDisplayNumber( Cur_Display-1 );
                        GetDisplayName(Cur_Display, devName, (BYTE)
sizeof(devName));

                        GetDisplayArea(devName, &rect);

                        Screen_Width = rect.right - rect.left;
                        Screen_Height = rect.bottom - rect.top;
                        MoveWindow(hWnd, rect.left, rect.top, Screen_Width,

```

```

Screen_Height, TRUE);

        theTimer = SetTimer( hWnd, 0x240, 500, NULL);
    }
    else DestroyWindow(hWnd);
}
break;
}

break;
case WM_PAINT:
{
    int  tStartX, tStartY;
    RECT rect;
    CHAR dName[25];
    CHAR Cur_Disp_Text[5];

    hdc = BeginPaint(hWnd, &ps);
    hfontNew = GetAFont(BOLD, 120, 120);
    hfontOld = (HFONT) SelectObject(hdc, hfontNew);
    SetTextColor( hdc, (COLORREF) 0xff0000);
    SetBkMode( hdc, TRANSPARENT);
    memset(dName, 0, sizeof(dName));
    GetDisplayName(Cur_Display, dName, (BYTE)sizeof(dName));
    GetDisplayArea(dName, &rect);

    SetTextAlign (hdc, GetTextAlign(hdc) |TA_CENTER);
    tStartX = (rect.right-rect.left)/2;
    tStartY = (rect.bottom-rect.top)/2;

    memset(Cur_Disp_Text, 0, sizeof(Cur_Disp_Text));
    sprintf(Cur_Disp_Text, "%d", Cur_Display);
    TextOut( hdc,  tStartX, tStartY-250, Cur_Disp_Text,
            strlen(Cur_Disp_Text));

    DeleteObject( hfontNew );

    SetTextColor( hdc, (COLORREF) 0xffffffff);
    hfontNew = GetAFont(BOLD, 30, 30);
    SelectObject(hdc, hfontNew);
    TextOut( hdc,  tStartX, tStartY-30, "Touch me to match.",

```

```

        strlen("Touch me to match."));

SetTextColor( hdc, (COLORREF) 0x0000ff);
TextOut( hdc, tStartX, tStartY+30, "Press [ESC] to abort.",
        strlen("Press [ESC] to abort.));

        SelectObject(hdc, hfontOld);
        DeleteObject( hfontNew );
        EndPaint(hWnd, &ps);
    }
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_TIMER:

    if (wParam==0x240){
        BOOL Status;

        Status=QueryDisplayBeTouched(); //(5)
        if (Status==TRUE) {
            RECT rect;
            CHAR devName[30];
            int Screen_Width, Screen_Height;

            Cur_Display++;
            if (Cur_Display<=Total_Display) {
                KillTimer( hWnd, theTimer );
                AskToSetDisplayNumber( Cur_Display-1 );
                GetDisplayName(Cur_Display, devName, (BYTE) sizeof(devName));
                GetDisplayArea(devName, &rect);

                Screen_Width = rect.right - rect.left;
                Screen_Height = rect.bottom - rect.top;
                MoveWindow(hWnd, rect.left, rect.top, Screen_Width, Screen_Height,
TRUE);

                theTimer = SetTimer( hWnd, 0x240, 500, NULL);
            }
            else DestroyWindow(hWnd);

```

```

        }
    }
    break;
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

```

```

HFONT GetAFont(int fnFont, LONG Height, LONG Width)
{
    static LOGFONT lf; // structure for font information

    // Get a handle to the ANSI fixed-pitch font, and copy
    // information about the font to a LOGFONT structure.

    GetObject(GetStockObject(ANSI_VAR_FONT), sizeof(LOGFONT),
        &lf);

    // Set the font attributes, as appropriate.

    if (fnFont & BOLD)
        lf.lfWeight = FW_BOLD;
    else
        lf.lfWeight = FW_NORMAL;

    if (fnFont & ITALIC) lf.lfItalic = TRUE;
    else lf.lfItalic = FALSE;

    if (fnFont & ULINE) lf.lfUnderline = TRUE;
    else lf.lfItalic = FALSE;

    // Create the font, and then return its handle.
    lf.lfHeight = Height;
    lf.lfWidth = Width;

    return CreateFont( lf.lfHeight, lf.lfWidth,
        lf.lfEscapement, lf.lfOrientation, lf.lfWeight,
        lf.lfItalic, lf.lfUnderline, lf.lfStrikeOut, lf.lfCharSet,
        lf.lfOutPrecision, lf.lfClipPrecision, lf.lfQuality,

```

```

        lf.IfPitchAndFamily, lf.IfFaceName);
    }

```

## **Example Program #14**

```

/* This example is showing how to use SetMeasureTime, GetMeasureTime. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    UINT   mTime;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }

    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.
    hController=OpenController(DeviceName);

    if (hController==INVALID_HANDLE_VALUE){

```

```

    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Input the value<0~255>: \n");
scanf("%d", &SelectedItem);
Status = SetMeasureTime(hController, (UINT) SelectedItem); //(1)

mTime = GetMeasureTime(hController); //(2)
Printf("\tCurrent Measuring Time: %d\n", mTime);

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #15**

```

/* This example is showing how to use SetMeasureDelay, GetMeasureDelay. */
void main(void)
{
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    UINT   dTime;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices   *****\n");

```

```

for (tcIdx=0; tcIdx<tcNum; tcIdx++)
{
    // get controller's name and use it to get device handle.
    GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
    printf("\t %d) %s\n", tcIdx+1, DeviceName);
}
printf("*****   Devices   *****\n\n");
printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
scanf("%d", &SelectedItem);
}

// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Input the value<0~255>: \n");
scanf("%d", &SelectedItem);
Status = SetMeasureDelay (hController , (UINT) SelectedItem) ; //(1)

dTime = GetMeasureDelay (hController); //(2)
Printf("\tCurrent Measuring Delay Time: %d\n", dTime);

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;
}

```

## **Example Program #16**

```

/* This example is showing how to use SetInfocus, GetInfocus. */
void main(void)
{

```



```

int    SelectedItem;
BYTE   tcNum, tcIdx;
CHAR   DeviceName[MAX_DEVICE_NAME];
BOOL   Status;
UINT   infocus;

//Get existing controller in the computer
tcNum = GetExistControllers();

if (tcNum == 0) {
    printf("No controller in the computer. <Press any key>\n");
    getch();
    goto Exit;
}
else{
    printf("*****   Devices *****\n");
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
    {
        // get controller's name and use it to get device handle.
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
    }
    printf("*****   Devices *****\n\n");
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
    scanf("%d", &SelectedItem);
}

// get the selected controller name from the index of device list.
GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
//Open Controller with its name.
hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t Input the value<0~255>: \n");
scanf("%d", &SelectedItem);
Status = SetInfocus (hController , (UINT) SelectedItem) ; //(1)

```

```
infocus = GetInfocus (hController); //(2)
```

```
Printf("\tCurrent In-focus: %d\n", infocus);
```

Exit:

```
if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
```

```
return;
```

```
}
```

## **Example Program #17**

```
/* This example is showing how to use SetOutfocus, GetOutfocus. */
```

```
void main(void)
```

```
{
```

```
int    SelectedItem;
```

```
BYTE   tcNum, tcIdx;
```

```
CHAR   DeviceName[MAX_DEVICE_NAME];
```

```
BOOL   Status;
```

```
UINT   outfocus;
```

```
//Get existing controller in the computer
```

```
tcNum = GetExistControllers();
```

```
if (tcNum == 0) {
```

```
    printf("No controller in the computer. <Press any key>\n");
```

```
    getch();
```

```
    goto Exit;
```

```
}
```

```
else{
```

```
    printf("*****   Devices *****\n");
```

```
    for (tcIdx=0; tcIdx<tcNum; tcIdx++)
```

```
    {
```

```
        // get controller's name and use it to get device handle.
```

```
        GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
```

```
        printf("\t %d) %s\n", tcIdx+1, DeviceName);
```

```
    }
```

```
    printf("*****   Devices *****\n\n");
```

```
    printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
```

```

        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.
    hController=OpenController(DeviceName);

    if (hController==INVALID_HANDLE_VALUE){
        printf("Can't open the controller.<Press any key>");
        getch();
        goto Exit;
    }

    printf("\t Input the value<0~255>: \");
    scanf("%d", &SelectedItem);
    Status = SetOutfocus (hController, (UINT) SelectedItem) ; //(1)

    outfocus = GetOutfocus (hController); //(2)
    Printf("\tCurrent Out-focus: %d\n", outfocus);

Exit:
    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #18**

```

/* This example is showing how to use SetCalibrationPoint , GetCalibrationData ,
SaveCalibrationData, and IsPenUp to do 3 pts calibration */
//

#include "stdafx.h"
#include "utpapi.h"
#include "resource.h"

#define MAX_LOADSTRING    100
#define MAX_NAME          20

```

```
#define MAX_TIMES
```

```
3
```

```
// Global Variables:
```

```
HINSTANCE hInst; // current instance
```

```
TCHAR szTitle[MAX_LOADSTRING]; // The title bar text
```

```
TCHAR szWindowClass[MAX_LOADSTRING]; // The  
title bar text
```

```
BYTE CurrPoint = 0;
```

```
BYTE CaliMode, oldMode;
```

```
HANDLE hDevice = INVALID_HANDLE_VALUE;
```

```
BOOL SendCommand(HANDLE hHandle, UCHAR *Cmd, BYTE Len);
```

```
// Forward declarations of functions included in this code module:
```

```
ATOM MyRegisterClass(HINSTANCE hInstance);
```

```
BOOL InitInstance(HINSTANCE, int);
```

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
void DrawGridLine(HDC hdc);
```

```
void DrawCaliPnt(HDC hdc, int Num);
```

```
void DrawCaliPnt(HDC hdc, BYTE Num, int Mode);
```

```
int APIENTRY WinMain(HINSTANCE hInstance,  
HINSTANCE hPrevInstance,  
LPSTR lpCmdLine,  
int nCmdShow)
```

```
{
```

```
    // TODO: Place code here.
```

```
    MSG msg;
```

```
    HACCEL hAccelTable;
```

```
    BYTE ExistDev=0;
```

```
    BOOL status;
```

```
    CHAR devName[MAX_NAME];
```

```
    UCHAR CmdBuf[12];
```

```
    // Initialize global strings
```

```
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
```

```
    LoadString(hInstance, IDC_CALIEXAMPLE, szWindowClass, MAX_LOADSTRING);
```

```
    ExistDev=GetExistControllers();
```

```
    if (!ExistDev) {
```

```

        MessageBox(NULL, "No device exist.", "Error!!", MB_OK);
        return FALSE;
    }
    else{
        status=GetControllerName(1, devName, MAX_NAME); //get the first controller's name
        if (!status) return FALSE;
        hDevice=OpenController(devName);
        if (hDevice==INVALID_HANDLE_VALUE) {
            MessageBox(NULL, "Error to open the first controller.", "Error!!", MB_OK);
            return FALSE;
        }
    }
}

//Enter MS Mode
oldMode = GetReportMode(hDevice); //(1)
SetReportMode(SetReportMode(hDevice, 2);

MyRegisterClass(hInstance);

// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
    if (hDevice!=INVALID_HANDLE_VALUE)
        CloseController(hDevice);
    return FALSE;
}

hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_CALIEXAMPLE);

// Main message loop:
while (GetMessage(&msg, NULL, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

if (hDevice!=INVALID_HANDLE_VALUE) {
    SetReportMode(SetReportMode(hDevice, oldMode);

```

```

        CloseController(hDevice);
    }
    return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//      This function and its usage is only necessary if you want this code
//      to be compatible with Win32 systems prior to the 'RegisterClassEx'
//      function that was added to Windows 95. It is important to call this function
//      so that the application will get 'well formed' small icons associated
//      with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style      = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)WndProc;
    wcex.cbClsExtra    = 0;
    wcex.cbWndExtra    = 0;
    wcex.hInstance     = hInstance;
    wcex.hIcon         = LoadIcon(hInstance, (LPCTSTR)IDI_CALIEXAMPLE);
    wcex.hCursor       = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH) GetStockObject(GRAY_BRUSH);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm       = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

    return RegisterClassEx(&wcex);
}

```

```

//
//  FUNCTION: InitInstance(HANDLE, int)
//
//  PURPOSE: Saves instance handle and creates main window
//
//  COMMENTS:
//
//      In this function, we save the instance handle in a global variable and
//      create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    int Screen_Height, Screen_Width;
    LONG    Value = 0;
    BOOLEAN tt=FALSE;

    hInst = hInstance; // Store instance handle in our global variable

    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

    hWnd = CreateWindow(szWindowClass, NULL, WS_POPUP,
        0, 0, Screen_Width, Screen_Height, NULL, NULL, hInstance, NULL);

    if (!hWnd)    return FALSE;

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//

```

```

// WM_COMMAND - process the application menu
// WM_PAINT - Paint the main window
// WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    int wmlId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    static BOOL IsGreen=FALSE;
    BOOL ReadyStatus, UpStatus;
    ULONG AvgPnt[2];

    switch (message)
    {
        case WM_CREATE:
            CurrPoint = 0;
            CaliMode = 3; //3-points calibration
            SetCalibrationPoint(hDevice, CurrPoint); //(1)
            SetTimer( hWnd, 0x240, 500, NULL);
            break;
        case WM_KEYDOWN:
            wmlId = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            switch(wmlId){
                case VK_ESCAPE: //Exit the program
                {
                    UCHAR CmdBuf[8];
                    CmdBuf[0]=0x01; CmdBuf[1]='R'; CmdBuf[2]=0x0D;
                    SendCommand(hDevice, CmdBuf, 3);
                    DestroyWindow(hWnd);
                }
                break;
            }
            break;
        case WM_PAINT:
            hdc = BeginPaint(hWnd, &ps);
            DrawGridLine(hdc);

```



```

        EndPaint(hWnd, &ps);
        break;
case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_TIMER:

    if (wParam==0x240){
        hdc = GetDC( hWnd );

        ReadyStatus= GetCalibrationData(hDevice, AvgPnt); //(2)
        if (ReadyStatus==TRUE) { //is ready to collect samples.
            DrawCaliedPnt(hdc, (int)CurrPoint);
            Beep(1000, 10);
            UpStatus=IsPenUp(hDevice); //(3)
            if (UpStatus==TRUE){
                Beep(2000, 20);
                CurrPoint++;
                if (CurrPoint>=CaliMode){
                    // store calibration data
                    SaveCalibrationData (hDevice, AvgPnt, sizeof(AvgPnt)); //(4)
                    DestroyWindow(hWnd);
                }
                else SetCalPoint(hDevice, CurrPoint, SAMPLES, CaliMode);
            }
        }
    }

    if (!ReadyStatus){
        if (IsGreen==FALSE){ //Show Green
            IsGreen=TRUE;
            DrawCaliPnt(hdc, CurrPoint, 1);
        }
        else{ //Show Gray
            IsGreen=FALSE;
            DrawCaliPnt(hdc, CurrPoint, 0);
        }
    }

    ReleaseDC( hWnd, hdc );
}
break;

```

```

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

void DrawCaliedPnt(HDC hdc, int Num)
{
    int Screen_Width, Screen_Height;
    HPEN hNewPen, hOldPen;
    int CXY[3][2]={128, 128}, {3968, 3968}, {3968, 128}};

    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

    hNewPen = CreatePen( PS_SOLID, 2, RGB(255, 255, 0)); //

    hOldPen = (HPEN)SelectObject (hdc, hNewPen);

    Arc( hdc, ((CXY[Num][0]*Screen_Width)>>12)-20, // x-coord of rectangle's upper-left corner
        ((CXY[Num][1]*Screen_Height)>>12)-20,
        ((CXY[Num][0]*Screen_Width)>>12)+20, // x-coord of rectangle's lower-right
corner
        ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of rectangle's lower-right
corner
        ((CXY[Num][0]*Screen_Width)>>12), // x-coord of first radial ending point
        ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of first radial ending point
        ((CXY[Num][0]*Screen_Width)>>12), // x-coord of second radial ending
point
        ((CXY[Num][1]*Screen_Height)>>12)+20 // y-coord of second radial ending
point
        );

    SelectObject (hdc, hOldPen);
    DeleteObject (hNewPen);

}

void DrawGridLine(HDC hdc)

```

```

{
    POINT    PrePoint;
    HPEN      hNewPen, hOldPen;
    int       Screen_Width, Screen_Height;

    // Get System Screen Width, Height
    Screen_Width=GetSystemMetrics( SM_CXSCREEN );
    Screen_Height=GetSystemMetrics( SM_CYSCREEN );

    hNewPen = CreatePen( PS_SOLID, 2, RGB(255, 255, 255)); //White color
    hOldPen = (HPEN)SelectObject (hdc, hNewPen);
    //
    MoveToEx( hdc, 0, (128*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width, (128*Screen_Height)>>12);

    MoveToEx( hdc, 0, (3968*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width, (3968*Screen_Height)>>12);

    MoveToEx( hdc, (128*Screen_Width)>>12, 0, &PrePoint);
    LineTo(hdc, (128*Screen_Width)>>12, Screen_Height);

    MoveToEx( hdc, (3968*Screen_Width)>>12, 0, &PrePoint);
    LineTo(hdc, (3968*Screen_Width)>>12, Screen_Height);

    MoveToEx( hdc, Screen_Width>>1, (128*Screen_Height)>>12, &PrePoint);
    LineTo(hdc, Screen_Width>>1, (3968*Screen_Height)>>12);

    MoveToEx( hdc, (128*Screen_Width)>>12, Screen_Height>>1, &PrePoint);
    LineTo(hdc, (3968*Screen_Width)>>12, Screen_Height>>1);

    SelectObject (hdc, hOldPen);
    DeleteObject (hNewPen);
}

```

```

void DrawCaliPnt(HDC hdc,  BYTE Num, int Mode)
{
    int Screen_Width, Screen_Height;
    HPEN  hNewPen, hOldPen;
    int CXY[9][2]={128, 128}, {3968, 3968}, {3968, 128}};

```

```

Screen_Width=GetSystemMetrics( SM_CXSCREEN );
Screen_Height=GetSystemMetrics( SM_CYSCREEN );
if (Mode ==1) hNewPen = CreatePen( PS_SOLID, 2, RGB(0, 255, 0)); //
else hNewPen = CreatePen( PS_SOLID, 2, RGB(200, 2000, 200)); //
hOldPen = (HPEN)SelectObject (hdc, hNewPen);

Arc( hdc, ((CXY[Num][0]*Screen_Width)>>12)-20, // x-coord of rectangle's upper-left corner
      ((CXY[Num][1]*Screen_Height)>>12)-20,
      ((CXY[Num][0]*Screen_Width)>>12)+20, // x-coord of rectangle's lower-right
corner
      ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of rectangle's lower-right
corner
      ((CXY[Num][0]*Screen_Width)>>12), // x-coord of first radial ending point
      ((CXY[Num][1]*Screen_Height)>>12)+20, // y-coord of first radial ending point
      ((CXY[Num][0]*Screen_Width)>>12), // x-coord of second radial ending
point
      ((CXY[Num][1]*Screen_Height)>>12)+20 // y-coord of second radial ending
point
      );

SelectObject (hdc, hOldPen);
DeleteObject (hNewPen);

}

```

```

BOOL SendCommand(HANDLE hHandle, UCHAR *Cmd, BYTE Len)
{
    BYTE    i;
    UCHAR    RspBuf[12];
    DWORD    gLen, RLen;

    for (i=0; i<MAX_TIMES; i++){
        FlushReadBuffer(hHandle);
        WriteBuffer(hHandle, Cmd, (DWORD) Len);
        Sleep(100);

        gLen = ReadLength(hHandle);
        if (gLen!=3) continue;
        RLen = ReadBuffer(hHandle, RspBuf, gLen);
    }
}

```

```

        if (RLen != 3) continue;
        else{
            if (RspBuf[0]==0x01 &&
                RspBuf[1]==0x30 &&
                RspBuf[2]==0x0D) return TRUE;
        }

    }

    return FALSE;
}

```

## **Example Program #19**

```

/* This example is showing how to use Set_Sound_Type, Get_Sound_Type. */
void main(void)
{
    int    SelectedItem;
    BOOL   Status;
    BYTE   SoundType;

    printf("\t 1) Buzzer Sound \n");
    printf("\t 2) Audio Sound \n");
    printf("\t Input the value<1 or 2>: ");

    scanf("%d", &SelectedItem);
    Status = Set_Sound_Type ( ( BYTE ) SelectedItem ) ; //(1)

    SoundType = Set_Sound_Type (); //(2)
    switch (SoundType){
        case 1:
            printf("\tCurrent is Buzzer Sound\n");
            break;
        case 2:
            printf("\tCurrent is Audio Sound\n");
            break;
    }
}

```

Exit:

```

    if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
    return;

}

```

## **Example Program #20**

```

/* This example is showing how to use SetTouchSensitivity, GetTouchSensitivity. */
void main(void)
{
    HANDLE hController;
    int    SelectedItem;
    BYTE   tcNum, tcIdx;
    CHAR   DeviceName[MAX_DEVICE_NAME];
    BOOL   Status;
    BYTE   Level;

    //Get existing controller in the computer
    tcNum = GetExistControllers();

    if (tcNum == 0) {
        printf("No controller in the computer. <Press any key>\n");
        getch();
        goto Exit;
    }
    else{
        printf("*****   Devices *****\n");
        for (tcIdx=0; tcIdx<tcNum; tcIdx++)
        {
            // get controller's name and use it to get device handle.
            GetControllerName((BYTE)(tcIdx+1), DeviceName, MAX_DEVICE_NAME);
            printf("\t %d) %s\n", tcIdx+1, DeviceName);
        }
        printf("*****   Devices *****\n\n");
        printf("\tSelect a device(1~%d)<Enter>:  ", tcNum);
        scanf("%d", &SelectedItem);
    }
    // get the selected controller name from the index of device list.
    GetControllerName((BYTE)SelectedItem, DeviceName, MAX_DEVICE_NAME);
    //Open Controller with its name.

```

```

hController=OpenController(DeviceName);

if (hController==INVALID_HANDLE_VALUE){
    printf("Can't open the controller.<Press any key>");
    getch();
    goto Exit;
}

printf("\t 0) Touch Sensitivity Level Is Low. \n");
printf("\t 1) Touch Sensitivity Level Is Normal. \n");
printf("\t 2) Touch Sensitivity Level Is More. \n");
printf("\t 3) Touch Sensitivity Level Is Very. \n");
printf("\t 4) Touch Sensitivity Level Is Most. \n");
printf("\t Input the value<0~4>. \n");
scanf("%d", &SelectedItem);
Status = SetTouchSensitivity (hController, (BYTE) SelectedItem); //(1)

Level = GetTouchSensitivity (hController); //(2)
switch(Level){
    case 0:
        printf("\t Current Touch Sensitivity Level Is Low. \n");
        break;
    case 1:
        printf("\t Current Touch Sensitivity Level Is Normal. \n");
        break;
    case 2:
        printf("\t Current Touch Sensitivity Level Is More. \n");
        break;
    case 3:
        printf("\t Current Touch Sensitivity Level Is Very. \n");
        break;
    case 4:
        printf("\t Current Touch Sensitivity Level Is Most. \n");
        break;
}

Exit:
if (hController!=INVALID_HANDLE_VALUE) CloseController(hController);
return;

```

}