Console Mode Program

An example UPDD API program, cbconsole, shipped with source code, has been created to demonstrate the UPDD V6 API interface and can be downloaded from here.

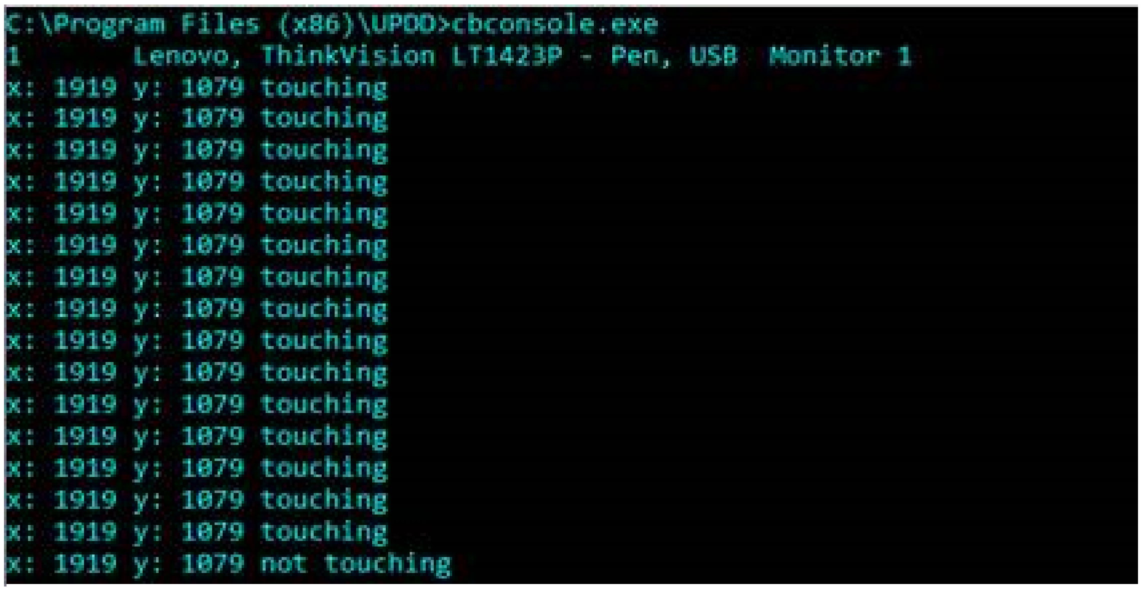
The cbconsole sample is a simple console mode application written in C++ to illustrate use of the UPDD API in the most basic form, without the complexities of a GUI interface.

This sample is intended as a starting point for integration of the UPDD API into UPDD Client applications.

In addition to illustrating the initialisation of the API interface, the example shows how to enumerate the list of defined devices and then read touch data from them.

Build the software as appropriate on your target system using appropriate development tools.

The example output is shown below:



```
C:\Program Files (x86)\UPDD>cbconsole.exe
1       Lenovo, ThinkVision LT1423P - Pen, USB  Monitor 1
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 touching
x: 1919 y: 1079 not touching
```

Quick look at the source code

The code of the console program, available on the download and reproduced here, may answer any immediate source related queries:

Console Mode Program

```
#include"upddapi.h"

#ifdef_WIN32
#include<windows.h>
#else
#include<unistd.h>
#endif
#include<iostream>
#include<string>
#include<vector>
usingnamespacestd;
#ifdef_LINUX_
#include<sys/types.h>
#include<sys/wait.h>
#endif

intDoList();
staticvoidTBCALLMyCallback(unsignedlongcontext, _PointerEvent* data);
staticvoidTBCALLMyConnectCallback(unsignedlongcontext, _PointerEvent* data);
boolrunning = true;
boolconnected = false;

intmain(intargc, char* argv[])
{
  // TBApiOpen executes asynchronously
  // a client program must wait for a connection
  // register to receive events when  the connection to the driver is made or broken
  // this is the only type of event that can be registered before TBApiOpen is called;
  TBApiRegisterEvent(0, 0, _EventConfiguration, MyConnectCallback);

 TBApiOpen();

  while (!connected) // in this contrived example we just wait for the connect event; in
a real scenario someting useful would be done at this point such as executing a programs
message dispatch loop
  {
#ifdef_WIN32
   Sleep(10);
#else
   usleep(100);
#endif
  }

  DoList();

// register for various data callbacks to receive information from UPDD
// NB ALL programs which will run for more than a second or so MUST register
_ReadDataTypeUnload and exit if this message is received
  TBApiRegisterEvent(0, // specify 0 to receive callbacks for all active devices or a
handle for a specific device
           0,   // this value gets passed to the callback function
           _EventTypeUnload |       // notification that we must unload (during driver
uninstall)
           _EventTypeDigitiserEvent, // digitiser events give information related to
touches
           MyCallback);

   while(running)
  {
```

```
#ifdef_WIN32
   Sleep(1000);
#else
   usleep(10000);
#endif
  }

// tell the api we no longer want to receive callbacks
  TBApiUnregisterEvent(MyCallback);
  TBApiClose();
  return(0);
}
intDoList()
{
  HTBDEVICEdevice = TBApiGetRelativeDevice(0);

  for(inti=0; device != TB_INVALID_HANDLE_VALUE;)
  {
   chardeviceName[256];
   TBApiGetSetting(device,"device_name",deviceName,sizeof(deviceName));

   intmonitor_number;
   TBApiGetSettingAsInt(device, "monitor_number", &monitor_number);

   cout << (char)(device + '0') << "\t" << deviceName << "\tMonitor " << monitor_number
<< endl;
   device = TBApiGetRelativeDevice(++i);
  }
  return(0);
```

```
}

staticvoidTBCALLMyConnectCallback(unsignedlong/*context*/, _PointerEvent* ev)
{
  if (ev->pe.config.configEventType == CONFIG_EVENT_CONNECT)
  {
   connected = true;
  }
}

staticvoidTBCALLMyCallback(unsignedlong/*context*/, _PointerEvent* ev)
{
  // the api calls this function to send events of the type we requested with
TBApiRegisterDataCallback()
  // NB this callback is called in the context of it's own thread
  // the user must implement thread synchronisation code when performing operations that
might content with
  // operations in other threads

 switch(ev->type)
  {
  case_EventTypeUnload:
   //[upddapi] driver is being uninstalled - all api programs must terminate
   TBApiClose();
   exit(0);
   break;
  case_EventTypeDigitiserEvent:
   {
    cout << "x: " << ev->pe.digitiserEvent.screenx
```

```
     << " y: " << ev->pe.digitiserEvent.screeny

     << (ev->pe.digitiserEvent.de.touchEvent.touchingLeft ? " touching" : " not
touching") << ends << endl;

   break;

   }

  }

}
```