# Bluetooth HID profile (client device)

## Overview

Symbian offers ways to show different profiles of services run on phone via Bluetooth. Services are usually run in host mode like OBEX. There is also HID profile available and that is also described phone as host so you can connect BT keyboard to phone. Here is solution to show phone as HID client, now other hosts like Windows/Linux PC can see phone for example as a mouse.

## Create profile and register service

First you need to register service to phone SDP server, through this other hosts will look devices services. You can find more information of HID from USB.org website [1]

Now create device, profile and protocol descriptors. Some other attributes are not crucial but for comfort, define those too.

```
const TUint16 KHidService = 0x1124;
const TInt KReportID = 0x03;
const TInt KMouseDescSize=52;
const TUint8 KMouseDesc[KMouseDescSize] = {
        0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
        0x09, 0x02,                    // USAGE (Mouse)
        0xa1, 0x01,                    // COLLECTION (Application)
        0x85, KReportID,                   //   REPORT_ID (1)
        0x09, 0x01,                    //   USAGE (Pointer)
        0xa1, 0x00,                    //   COLLECTION (Physical)
        0x05, 0x09,                    //     USAGE_PAGE (Button)
        0x19, 0x01,                    //     USAGE_MINIMUM (Button 1)
        0x29, 0x03,                    //     USAGE_MAXIMUM (Button 3)
        0x15, 0x00,                    //     LOGICAL_MINIMUM (0)
        0x25, 0x01,                    //     LOGICAL_MAXIMUM (1)
        0x95, 0x03,                    //     REPORT_COUNT (3)
        0x75, 0x01,                    //     REPORT_SIZE (1)
        0x81, 0x02,                    //     INPUT (Data,Var,Abs)
        0x95, 0x01,                    //     REPORT_COUNT (1)
        0x75, 0x05,                    //     REPORT_SIZE (5)
        0x81, 0x03,                    //     INPUT (Cnst,Var,Abs)
        0x05, 0x01,                    //     USAGE_PAGE (Generic Desktop)
        0x09, 0x30,                    //     USAGE (X)
        0x09, 0x31,                    //     USAGE (Y)
        0x15, 0x81,                    //     LOGICAL_MINIMUM (-127)
        0x25, 0x7f,                    //     LOGICAL_MAXIMUM (127)
        0x75, 0x08,                    //     REPORT_SIZE (8)
        0x95, 0x02,                    //     REPORT_COUNT (2)
        0x81, 0x06,                    //     INPUT (Data,Var,Rel)
        0xc0,                          //   END_COLLECTION
        0xc0                           // END_COLLECTION
};
```

And then create profile and register it

```
void CBTAdvertiser::BuildHIDL()
    {
    iSdpDB.CreateServiceRecordL(TUUID(TUint16(KHidService)),iRecordHandle);

    CSdpAttrValueDES* protocolDescList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueDES* addProtocolDescList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueDES* profileDescList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueDES* browseGroupList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueDES* langList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueBoolean* reconnect = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueBoolean* virtualCable = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueBoolean* sdpDisp = CSdpAttrValueBoolean::NewBoolL(EFalse);

    CSdpAttrValueBoolean* battery = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueBoolean* remoteWake = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueDES* hidDescList = CSdpAttrValueDES::NewDESL(NULL);

    CSdpAttrValueUint* parserVersion = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint16>(0x0111));
    CSdpAttrValueUint* releaseNumber = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint16>(0x0100));
    CSdpAttrValueUint* countryCode = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint8>(0x21));
    CSdpAttrValueUint* subClass = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint8>(KMouseSubClassID)); //mouse
    CSdpAttrValueBoolean* normallyConnectable = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueBoolean* bootDevice = CSdpAttrValueBoolean::NewBoolL(ETrue);
    CSdpAttrValueUint* profileVersion = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint16>(0x0100));


    CSdpAttrValueDES* baseLangList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueDES* serviceClassIdList = CSdpAttrValueDES::NewDESL(NULL);
    CSdpAttrValueUint* supervisionTimeout = CSdpAttrValueUint::NewUintL(TSdpIntBuf<TUint16>(0x1f40));

    CleanupStack::PushL(protocolDescList);
    CleanupStack::PushL(addProtocolDescList);
    CleanupStack::PushL(profileDescList);
    CleanupStack::PushL(browseGroupList);
    CleanupStack::PushL(langList);
    CleanupStack::PushL(reconnect);
    CleanupStack::PushL(virtualCable);
    CleanupStack::PushL(sdpDisp);
    CleanupStack::PushL(bootDevice);
    CleanupStack::PushL(battery);
    CleanupStack::PushL(remoteWake);

    CleanupStack::PushL(parserVersion);
    CleanupStack::PushL(releaseNumber);
    CleanupStack::PushL(countryCode);
```


(C) Copyright Nokia 2013. All rights reserved.

```
CleanupStack::PushL(hidDescList);
CleanupStack::PushL(subClass);
CleanupStack::PushL(normallyConnectable);
CleanupStack::PushL(profileVersion);
CleanupStack::PushL(baseLangList);
CleanupStack::PushL(serviceClassIdList);
CleanupStack::PushL(supervisionTimeout);


protocolDescList
    ->StartListL()
        ->BuildDESL()
            ->StartListL()
                ->BuildUUIDL(TUUID(TUint16(KL2CAP)))
                ->BuildUintL(TSdpIntBuf<TUint8>(KControlChannel))
            ->EndListL()
        ->BuildDESL()
            ->StartListL()
                ->BuildUUIDL(TUUID(TUint16(KHIDDesc)))
            ->EndListL()
        ->BuildDESL()
            ->StartListL()
                ->BuildUUIDL(TUUID(TUint16(KL2CAP)))
                ->BuildUintL(TSdpIntBuf<TUint8>(KInterruptChannel))
            ->EndListL()
        ->BuildDESL()
            ->StartListL()
                ->BuildUUIDL(TUUID(TUint16(KHIDDesc)))
            ->EndListL()
    ->EndListL();

addProtocolDescList
->StartListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUUIDL(TUUID(TUint16(KL2CAP)))
            ->BuildUintL(TSdpIntBuf<TUint8>(KInterruptChannel))
        ->EndListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUUIDL(TUUID(TUint16(KHIDDesc)))
        ->EndListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUUIDL(TUUID(TUint16(KL2CAP)))
            ->BuildUintL(TSdpIntBuf<TUint8>(KControlChannel))
        ->EndListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUUIDL(TUUID(TUint16(KHIDDesc)))
        ->EndListL()
->EndListL();

profileDescList
->StartListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUUIDL(TUUID(TUint16(KHidService)))
            ->BuildUintL(TSdpIntBuf<TUint16>(0x0100))
        ->EndListL()
->EndListL();

langList
        ->StartListL()
            ->BuildUintL(TSdpIntBuf<TUint16>(KLanguageEnglish)) // en
            ->BuildUintL(TSdpIntBuf<TUint16>(0x006a)) // utf8
            ->BuildUintL(TSdpIntBuf<TUint16>(0x0100)) // version
        ->EndListL();


HBufC8* data = NULL;
TPtr8 ptr(0,0);
TInt dataSize(0);
TUint8* desc = NULL;;
dataSize += KMouseDescSize;
data = HBufC8::NewLC(dataSize);
ptr.Set(data->Des());

desc = (TUint8*)KMouseDesc;
for(TInt i=0;i<KMouseDescSize;i++)
    {
    ptr.Append(TChar(desc[i]));
    }


hidDescList
->StartListL()
    ->BuildDESL()
        ->StartListL()
            ->BuildUintL(TSdpIntBuf<TUint8>(0x22))
            ->BuildStringL(ptr)
        ->EndListL()
 ->EndListL();
CleanupStack::PopAndDestroy(data);

baseLangList
->StartListL()
    ->BuildDESL()
    ->StartListL()
        ->BuildUintL(TSdpIntBuf<TUint16>(0x0409))
        ->BuildUintL(TSdpIntBuf<TUint16>(0x0100))
    ->EndListL()
->EndListL();

serviceClassIdList
->StartListL()
    ->BuildUUIDL(TUUID(TUint16(KHidService)))
->EndListL();

browseGroupList
->StartListL()
    ->BuildUUIDL(TUUID(TUint16(KPublicBrowseGroupUUID)))
->EndListL();


// set protocol list to the record
iSdpDB.UpdateAttributeL(iRecordHandle,KSubClass, *subClass);
iSdpDB.UpdateAttributeL(iRecordHandle,KCountryCode, *countryCode);
iSdpDB.UpdateAttributeL(iRecordHandle,KReleaseNumber, *releaseNumber);
```


(C) Copyright Nokia 2013. All rights reserved.

```
    iSdpDB.UpdateAttributeL(iRecordHandle,KParserVersion, *parserVersion);
    iSdpDB.UpdateAttributeL(iRecordHandle,KBootDevice,*bootDevice);
    iSdpDB.UpdateAttributeL(iRecordHandle,KRemoteWake,*remoteWake);
    iSdpDB.UpdateAttributeL(iRecordHandle,KBatteryPower,*battery);
    iSdpDB.UpdateAttributeL(iRecordHandle,KVirtualCable,*virtualCable);
    iSdpDB.UpdateAttributeL(iRecordHandle,KReconnectInitiate,*reconnect);
    iSdpDB.UpdateAttributeL(iRecordHandle,KSdpDisposable,*sdpDisp);
    iSdpDB.UpdateAttributeL(iRecordHandle,KSdpAttrIdBrowseGroupList,*browseGroupList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KSdpAttrIdProtocolDescriptorList,*protocolDescList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KAdditionalProtocol,*addProtocolDescList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KSdpAttrIdBluetoothProfileDescriptorList,*profileDescList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KHIDBaseLangList,*baseLangList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KHIDDescList, *hidDescList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KHIDProfileVersion,*profileVersion);
    iSdpDB.UpdateAttributeL(iRecordHandle,KHIDNormallyConnectable,*normallyConnectable);
    iSdpDB.UpdateAttributeL(iRecordHandle,KSdpAttrIdServiceClassIDList, *serviceClassIdList);
    iSdpDB.UpdateAttributeL(iRecordHandle,KHIDSupervisionTimeout,*supervisionTimeout);


    CleanupStack::PopAndDestroy(supervisionTimeout);
    CleanupStack::PopAndDestroy(serviceClassIdList);
    CleanupStack::PopAndDestroy(baseLangList);
    CleanupStack::PopAndDestroy(profileVersion);
    CleanupStack::PopAndDestroy(normallyConnectable);
    CleanupStack::PopAndDestroy(subClass);
    CleanupStack::PopAndDestroy(hidDescList);
    CleanupStack::PopAndDestroy(countryCode);
    CleanupStack::PopAndDestroy(releaseNumber);
    CleanupStack::PopAndDestroy(parserVersion);
    CleanupStack::PopAndDestroy(remoteWake);
    CleanupStack::PopAndDestroy(battery);
    CleanupStack::PopAndDestroy(bootDevice);
    CleanupStack::PopAndDestroy(sdpDisp);
    CleanupStack::PopAndDestroy(virtualCable);
    CleanupStack::PopAndDestroy(reconnect);
    CleanupStack::PopAndDestroy(langList);
    CleanupStack::PopAndDestroy(browseGroupList);
    CleanupStack::PopAndDestroy(profileDescList);
    CleanupStack::PopAndDestroy(addProtocolDescList);
    CleanupStack::PopAndDestroy(protocolDescList);


    // add a name to the record
    iSdpDB.UpdateAttributeL(iRecordHandle,
                            KSdpAttrIdBasePrimaryLanguage +
                                      KSdpAttrIdOffsetServiceName,
                                  KServiceName());

    // add a description to the record
    iSdpDB.UpdateAttributeL(iRecordHandle,
                                  KSdpAttrIdBasePrimaryLanguage +
                                      KSdpAttrIdOffsetServiceDescription,
                                      KServiceDesc());

    iSdpDB.UpdateAttributeL(iRecordHandle,
                           KSdpAttrIdOffsetProviderName + KSdpAttrIdBasePrimaryLanguage,
                           KServiceProvider());
    }
```

## Setting up connections

After creating and registering profile we are not finished yet. Now we need to wait socket connections and possibly make connections. HID devices use two ports or channels; 0x11 for control channel and 0x13 for interrupt channel. Note that these are 17 and 19 in decimals. Fortunately Symbian allows to use these and there is not limitations not to use lower channels, even in **bt_sock.h** is mentioned that minimum for user is 0x1001 and smaller are possible but reserved for Bluetooth stack use.

For interrupt channel you don't need encryption but control channel has to be encrypted. This is done only for listening sockets, connecting sockets are handled by host.

```
    TL2CAPSockAddr addr;
    addr.SetPort(iPort);
    TBTServiceSecurity serverSecurity;
    serverSecurity.SetUid(KAppUid);
    serverSecurity.SetAuthentication(EFalse);
    serverSecurity.SetAuthorisation(EFalse);
    serverSecurity.SetDenied(EFalse);
    serverSecurity.SetEncryption(ETrue);

    if (KInterruptChannel == iPort)
        {
        serverSecurity.SetEncryption(EFalse);
        }

    addr.SetSecurity(serverSecurity);

    //Bind
    iListeningSocket.Bind(addr);
    iListeningSocket.Listen(1);
```

### Check received data

Almost done, we have advertising and sockets with security but how about data. What to send and what we are receiving from host. We don't now care what host will send us except disconnect message:

```
void CSocketHandler::CheckData()
    {
    TUint cmd(0);
    TUint param(0);
    cmd = iDataPtr[0] & 0xf0;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    param = iDataPtr[0] & 0x0f;

    switch(cmd)
        {
        case 0x10:
            {
            if(param == 0x05)
                {
                // 0x15 == Virtual cable unplug
                DisconnectHost();
                iObserver.Disconnected();
                }
            }break;

        default:
            break;
        }
    }
```

## Sending data

And what we send to host is header, device descriptors report ID, button bitmap mask and coordinate movement:

```
void CBlueMouseAppUi::SendMouseData(TInt aX, TInt aY, TUint8 aButtonMask)
    {
    TUint8* data = (TUint8*)User::AllocLC(KDataSize);
    TPtr8 ptr(data,KDataSize,KDataSize);
    TInt x(0);
    TInt y(0);

    x = aX/(iSettings.iResolution/5);
    y = aY/(iSettings.iResolution/5);

    ptr[0] = 0xa1;
    ptr[1] = KReportID;
    ptr[2] = aButtonMask;

    if(aButtonMask != 0x00)
        {
        ptr[3] = 0;
        ptr[4] = 0;
        }
    else
        {
        ptr[3] = (TUint8)x;
        ptr[4] = (TUint8)y;
        }
    iSocketHandler->SendDataL(ptr);
    CleanupStack::PopAndDestroy(data);
    }
```

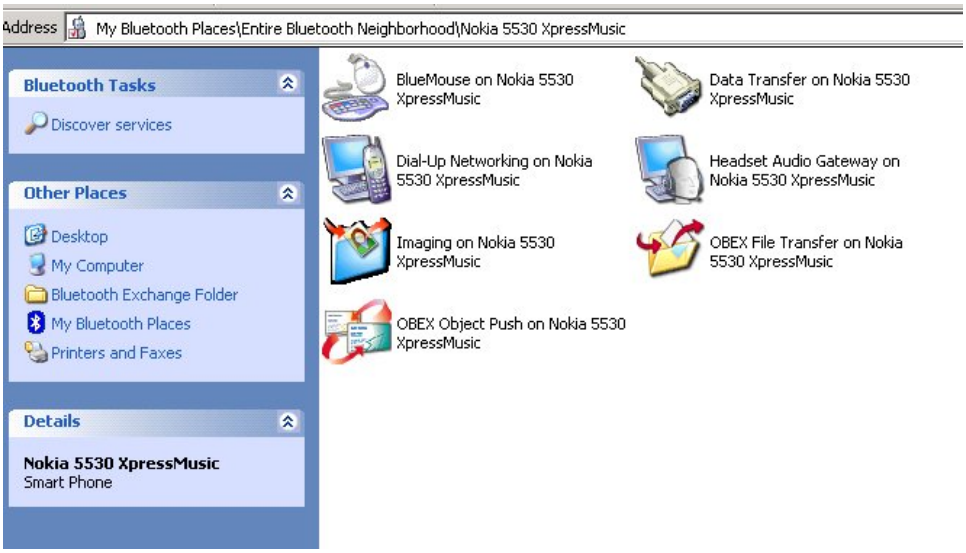
Send data, remember this is unsigned...

```
void CSocketHandler::SendDataL(const TDesC8& aData)
    {
    if(!IsConnected())
        return;

    if(IsActive())
        Cancel();

    TInt len(aData.Length());
    TUint8* data = (TUint8*)User::AllocLC(len);
    TPtr8 ptr(data,len,len);
    ptr.Copy(aData);

    iInterrupt.Write(ptr,iStatus);
    CleanupStack::PopAndDestroy(data);

    iState = ESending;
    SetActive();
    }
```

## Outcome

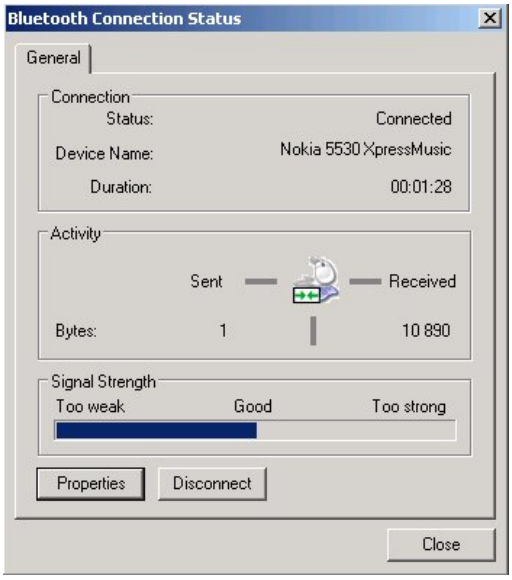
Pair your phone with computer as normal, then open Bluetooth connections and you see this:


(C) Copyright Nokia 2013. All rights reserved.




Right click HID icon and choose connect, you will be informed a new device has been found. Windows installs this automatically:

Now we are connected, to be sure you can see connection status and sent/received data amount by clicking HID icon:




Same can be done in Linux with **hidd**, just connect or set hidd to be server and connect from phone.

## Conclusion

This is tested with Windows and Linux desktop. It is working with BlueZ and Broadcom Bluetooth stacks, unfortunately Microsoft stack don't work and MacOS is not tested.

See attached source code example File:BlueMouse.zip for help. It creates profile as described before, sets up socket connections and use touch screen or accelerometer for mouse XY-data. Remember, it is code example and not fully featured program.


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.