HID Device Interface入門

ジョイスティックをアプリケーションで扱うお話。

村上幸雄

開発環境：PowerBook G4 12” MacOSX 10.4.7 Xcode 2.4
発表日：2006/9/30

■はじめに

【図1】Classic Gamepad (JY-P1R)

この発表では、I/O Kit のHID (Human Interface Device) の Device Interface を用いた、ジョイスティックの利用方法を紹介します。サンプルコードで利用するジョイスティックは、昨年末、Cocoa 勉強の有志で開催された USB 勉強会で題材として採用されたサンワサプライ製の Classic Gamepad （図1）です。残念ながら、Classic Gamepad の販売は終わってしまったようですが、サンプルコードは他の同様な USB デバイスでも利用可能なものとなっています。

■開発者向け情報

Mac OS X で、USB デバイス等のハードウェアを扱う為の情報は、ADC (Apple Developer Connection) のサイト (http://developer.apple.com/jp/) から入手できます。日本語に翻訳されている情報も一部ありまして、以下の日本語翻訳された PDF ファイルは、Mac OS X でハードウェアを扱うソフトウェアを開発する際は、一度は目を通しておくべきものとなっています。

- （URL）http://developer.apple.com/jp/documentation/pdf/IOKitFundamentals_j.pdf
  （I/O Kit Fundamentals）
- （URL） http://developer.apple.com/jp/documentation/pdf/AccHardware_j.pdf
  （Accessing Hardware From Applications）

本発表で扱います I/O Kit の HID の Device Interface については、以下の情報が参考になります。ただし、残念ながら英語のドキュメントとなっています。

- （URL）http://developer.apple.com/documentation/DeviceDrivers/Conceptual/HID/index.html
  （HID Class Device Interface Guide）

その他の情報として、著者は以下の書籍を参考にしましたので、紹介します。

- （書籍）最新技術解説 入門USB　（著）鈴木一海、五十嵐顕寿　ISBN 4-7741-1185-6
- （書籍）USBコンプリート [第3版]　（著）ジョン・アクセルソン　ISBN 4-434-08460-7

■用語の説明

本発表では、一般的なデスクトップアプリケーション開発では出てこない用語が出てきますので、初めに説明します。

ジョイスティック

主にゲームでの利用が想定されています、コンピュータのへの入力装置の一種です。ジョイスティックには色々な形状のものがあり、名称もジョイパッドやゲームパッド、ゲームコントローラと色々ですが、USB の HID クラスでは、これらはまとめて、ジョイスティックとして定義されています。

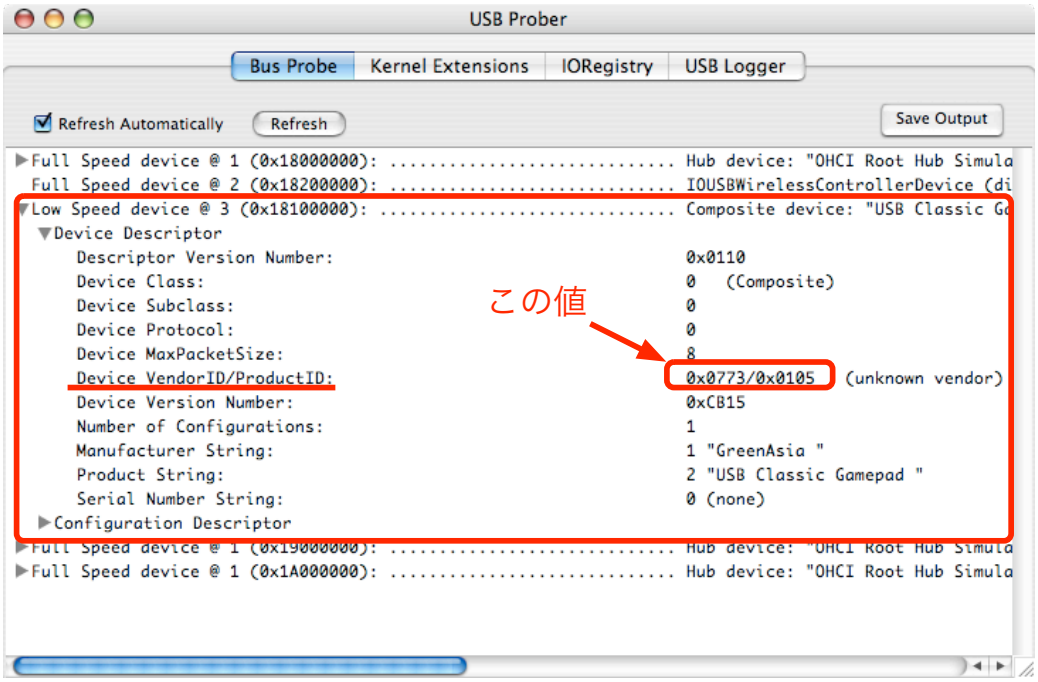
HID (Human Interface Device)

USB では、類似する性質やサービスは、クラスというひとまとまりの仕様として定義しています。本発表で取り上げるジョイスティックやキーボード、マウスは、人がコンピュータを操作する際に利用するデバイスということで、USB では HID (Human Interface Device) クラスとして定義されています。

ベンダ ID とプロダクト ID

USB では、デバイスの種類を識別するために、ベンダ ID とプロダクト ID が割り振られています。 このベンダ ID とプロダクト ID は、Xcode をインストールすると一緒にインストールされる USB Prober （ /Developer/Applications/Utilities/USB Prober.app ）で確認することができます（図2）。

【図2】USB Prober



I/O Kit と Device Interface

I/O Kit とは、Mac OS X でデバイスドライバを開発するためのオブジェクト指向のフレームワークです。Device Interface とは、アプリケーションから I/O Kit のデバイスドライバにアクセスするための機構です。

I/O Kit ファミリ

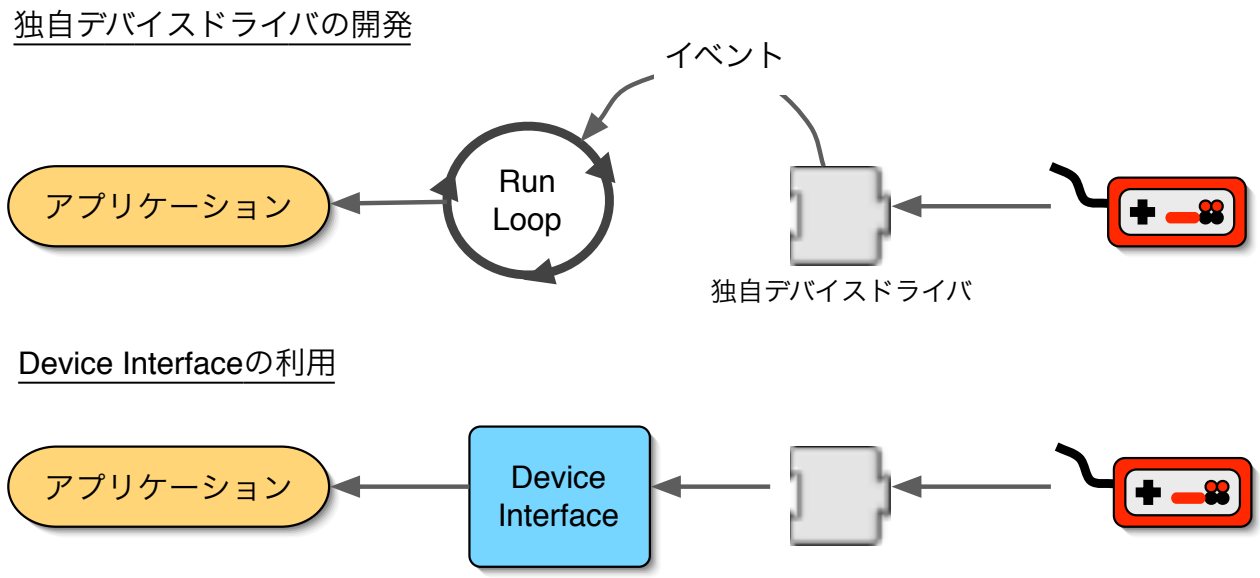
I/O Kit ファミリとは、特定のハードウェアに依存しない、類似する性質やサービスでまとめられたクラスの集まりのことで、バスプロトコル（USB、FireWire等）やストレージ、HID（Human Interface

Device）等があります。あるファミリに属するドライバを開発する場合は、ファミリのクラスのサブクラスとして実装するということになります。

■HID Device Interface について

ジョイスティックをアプリケーションで扱う方法は、HID の Device Interface を利用するだけではありません。独自のデバイスドライバを開発して、例えば、キーボードやマウスのイベントを模するという方法も可能です（図3）。 でも、独自デバイスドライバの開発する方法では、特定のアプリケーションからのみ利用されるのみもかかわらずカーネルリソースが消費されてしまいますので、本発表では、Device Interface の利用を選択しました。

【図3】独自デバイスドライバの開発と Device Interface の利用の比較



それでは、次のページからは以下の内容について、実際のサンプルコードを使って説明します。

■Xcode のプロジェクトの設定
■デバイス検索
　○デバイス検索
　○マッチングしたデバイスへのアクセス
　○デバイス情報の取得
■Device Interface の利用
　○Device Interface の生成
　○クッキーの取得
■ユーザ操作
　○ダイレクトに値を取得
　○ポーリングによる値の取得
　○コールバックによる値の取得
　○ユーザ操作の監視

■Xcode のプロジェクトの設定

プログラムで HID の Device Interface を利用できるようにする手順は、以下のとおりです。

1. CoreFoundation.framework をプロジェクトに追加する。
2. IOKit.framework をプロジェクトに追加する。
3. ソースに必要なヘッダファイルの include するコードを追加する。

include が必要なヘッダファイルは、プログラムによって異なることになりますが、サンプルコードでは、以下のとおりとなっています。

gamepad.c / include文

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdbool.h>
#include <sys/errno.h>
#include <sysexits.h>
#include <mach/mach.h>
#include <mach/mach_error.h>
#include <IOKit/IOKitLib.h>
#include <IOKit/IOCFPlugIn.h>
#include <IOKit/IOMessage.h>
#include <IOKit/hid/IOHIDLib.h>
#include <IOKit/hid/IOHIDKeys.h>
#include <IOKit/hid/IOHIDUsageTables.h>
#include <IOKit/hidsystem/IOHIDLib.h>
#include <IOKit/hidsystem/IOHIDShared.h>
#include <IOKit/hidsystem/IOHIDParameter.h>
#include <CoreFoundation/CoreFoundation.h>
```

■デバイス検索

○デバイス検索

　プログラムからジョイスティックにアクセスする為には、I/O Kit のデバイスマッチング処理を使って、デバイスを特定する必要があります。このデバイスマッチング処理の方法は複数ありますが、サンプルコードでは、USB デバイスのベンダ ID とプロダクト ID をマッチングのキーにして、デバイスが抜き差しされた際に登録したコールバック関数が呼ばれる内容となっています。

gamepad.c / main()

```
int main (int argc, const char * argv[]) {
    kern_return_t           kr;
    SInt32                  hidVendor = kClassicGamepadVendorID;
    SInt32                  hidProduct     = kClassicGamepadProductID;
    mach_port_t             masterPort;
    CFRunLoopSourceRef      runLoopSource;
    CFMutableDictionaryRef  hidMatchDictionary = 0;

    if (argc > 1)  hidVendor = atoi(argv[1]);
    if (argc > 2)  hidProduct = atoi(argv[2]);

(1) kr = IOMasterPort(MACH_PORT_NULL, &masterPort);

(2) hidMatchDictionary = IOServiceMatching(kIOHIDDeviceKey);


(3) CFDictionarySetValue(hidMatchDictionary, CFSTR(kIOHIDVendorIDKey),
                CFNumberCreate(kCFAllocatorDefault, kCFNumberIntType, &hidVendor));
    CFDictionarySetValue(hidMatchDictionary, CFSTR(kIOHIDProductIDKey),
                CFNumberCreate(kCFAllocatorDefault, kCFNumberIntType, &hidProduct));

(4) gNotifyPortRef = IONotificationPortCreate(masterPort);
    runLoopSource = IONotificationPortGetRunLoopSource(gNotifyPortRef);
    CFRunLoopAddSource(CFRunLoopGetCurrent(), runLoopSource, kCFRunLoopDefaultMode);

    hidMatchDictionary = (CFMutableDictionaryRef)CFRetain(hidMatchDictionary);

(5) kr = IOServiceAddMatchingNotification(gNotifyPortRef,
              kIOFirstMatchNotification, hidMatchDictionary,
              HIDAdded, NULL, &gHIDAddedIter);
    HIDAdded(NULL, gHIDAddedIter);

(6) kr = IOServiceAddMatchingNotification(gNotifyPortRef,
              kIOTerminatedNotification, hidMatchDictionary,
              HIDRemoved, NULL, &gHIDRemovedIter);
    HIDRemoved(NULL, gHIDRemovedIter);

    mach_port_deallocate(mach_task_self(), masterPort);
    masterPort = 0;

(7) CFRunLoopRun();
    return EXIT_SUCCESS;
}
```

　上記コードの番号が振られた箇所について、次のページで説明します。

(1) カーネルとの通信の為の Mach ポートの取得
プログラムがカーネルと通信する為の Mach カーネルのポートと呼ばれる通信チャネルを取得しています。

(2) HID のデバイスマッチングの為の辞書の取得
関数 IOServiceMatching() に kIOHIDDeviceKey を指定して、HID デバイスのマッチング処理を行うための CFMutableDictionary 型のディクショナリを取得しています。

(3) ベンタ ID ／プロダクト ID の指定
ベンダ ID とプロダクト ID を(2)で取得したディクショナリに設定しています。
HID のデバイスマッチングではベンダ ID とプロダクト ID 以外のキーを指定することが可能です。具体的にどのようなキーが指定可能かの情報は、以下のヘッダファイルを調べることによって得ることが出来ます。

/System/Library/Frameworks/IOKit.framework/Versions/A/Headers/hid/IOHIDKeys.h

(4) コールバック登録の為の準備
コールバック関数を登録するために必要な手続きです。

(5) USB デバイスが差し込まれた際に呼ばれるコールバック関数の登録
コールバック関数として HIDAdded() を登録しています。関数 IOServiceAddMatchingNotification() 読み出し時にデバイスがマッチングした場合は、イタレータ変数 gHIDAddedIter にマッチングしたデバイスの情報が設定されています。その為、直後に登録したコールバック関数 HIDAdded() を呼んで、マッチングしたデバイスに対する処理を行います。

(6) USB デバイスが抜かれた際に呼ばれるコールバック関数の登録
(5) と同様です。コールバック関数として HIDRemoved() を登録しています。

(7) Run Loop開始
コールバック関数はRun Loopの仕組みで呼ばれますので、このコードは必要です。

○マッチングしたデバイスへのアクセス

gamepad.c / HIDAdded()の抜粋

```
void HIDAdded(void *refCon, io_iterator_t iterator)
{
        io_service_t                hidDevice;
        ・・・

        while (hidDevice = IOIteratorNext(iterator)) {
                ・・・
                kr = IOObjectRelease(hidDevice);
                ・・・
        }
}
```

マッチングしたデバイスは、コールバック関数の引数として渡される、I/O Kitのイタレータ変数を使ってアクセスします。IOIteratorNext() で NULL が返えされるまでループすれば、マッチングした全てのデバイスが取得できるということになります。

○デバイス情報の取得

I/O Kit の API を使用すれば、デバイスマッチングで得られたデータから、デバイスの情報を取得することができます。以下のコードでは、ベンダ ID と プロダクト ID を取得して、取得した値を標準出力に印字しています。

gamepad.c / ShowHIDProperties()

```
void ShowHIDProperties(io_registry_entry_t hidDevice)
{
        kern_return_t          result;
        CFMutableDictionaryRef properties = 0;
        char                   path[512];
        int                    vendor = 0, product = 0;

        result = IORegistryEntryGetPath(hidDevice, kIOServicePlane, path);
        if (result == KERN_SUCCESS)   printf("[ %s ]\n", path);

        result = IORegistryEntryCreateCFProperties(hidDevice, &properties,
                        kCFAllocatorDefault, kNilOptions);
        if ((result == KERN_SUCCESS) && properties) {
                CFNumberRef vendorRef, productRef;
                vendorRef = CFDictionaryGetValue(properties, CFSTR(kIOHIDVendorIDKey));
                productRef = CFDictionaryGetValue(properties, CFSTR(kIOHIDProductIDKey));
                if (vendorRef) {
                        CFNumberGetValue(vendorRef, kCFNumberIntType, &vendor);
                }
                if (productRef) {
                        CFNumberGetValue(productRef, kCFNumberIntType, &product);
                }
                CFRelease(properties);
        }
        printf("Got a device: vendor %04x, product %04x\n", vendor, product);
}
```

■Device Interface の利用

○Device Interface の生成

Device Interface の生成は、以下の２段階で行われます。

1. 中間 I/O Kit プラグインインタフェースの生成
2. 中間 I/O Kit プラグインインタフェースから Device Interface を生成。

以下が具体的なコードです。コード中の番号と、上記箇条書きの番号は一致します。

gamepad.c / CreateHIDDeviceInterface()

```
void CreateHIDDeviceInterface(io_object_t hidDevice,
          IOHIDDeviceInterface ***hidDeviceInterface)
{
        io_name_t                       className;
        IOCFPlugInInterface     **plugInInterface = NULL;
        HRESULT                         plugInResult = S_OK;
        SInt32                          score = 0;
        IOReturn                        ioReturnValue = kIOReturnSuccess;


        ioReturnValue = IOObjectGetClass(hidDevice, className);
        print_errmsg_if_io_err(ioReturnValue, "Failed to get class name.");
        printf("Found device type %s\n", className);

(1)     /* Getting an intermediate IOCFPlugInInterface object */
        ioReturnValue = IOCreatePlugInInterfaceForService(hidDevice,
                        kIOHIDDeviceUserClientTypeID,
                        kIOCFPlugInInterfaceID, &plugInInterface, &score);
        if ((kIOReturnSuccess != ioReturnValue) || !plugInInterface) {
                printf("Unable to create a plug-in (%08x)\n", ioReturnValue);
                exit(EXIT_FAILURE);
        }

(2)     /* Getting a specific device interface object (COM) */
        if (ioReturnValue == kIOReturnSuccess) {
                plugInResult = (*plugInInterface)->QueryInterface(plugInInterface,
                        CFUUIDGetUUIDBytes(kIOHIDDeviceInterfaceID),
                        (LPVOID)hidDeviceInterface);
                print_errmsg_if_err(plugInResult != S_OK,
                        "Couldn't create HID class device interface");
                (*plugInInterface)->Release(plugInInterface);
                if (plugInResult || !(*hidDeviceInterface)) {
                        printf("Couldn't create a device interface (%08x)\n",
                                (int)plugInResult);
                }
        }
}
```

○クッキーの取得

HID の Device Interface では、ジョイスティックの十字キーやボタン等の要素は、クッキーと呼ばれるデータで識別されます。次ページのコードでは、クッキーを取得して、十字キーやボタンとの対応を覚えています。

gamepad.c / GetHIDCookies()

```
HIDCookiePtr GetHIDCookies(IOHIDDeviceInterface122 **handle)
{
     HIDCookiePtr            cookies = calloc(1, sizeof(HIDCookie));
     CFTypeRef               object;
     IOHIDElementCookie      cookie;
     long                    number, usage, usagePage;
     CFArrayRef              elements;
     CFDictionaryRef         element;
     IOReturn                success;

     if (!handle || !(*handle))    return cookies;
     success = (*handle)->copyMatchingElements(handle, NULL, &elements);
     if (success == kIOReturnSuccess) {
          CFIndex i;
          for (i = 0; i < CFArrayGetCount(elements); i++) {
               element = CFArrayGetValueAtIndex(elements, i);
               object = (CFDictionaryGetValue(element, CFSTR(kIOHIDElementCookieKey)));
               if (object == 0 || CFGetTypeID(object) != CFNumberGetTypeID())   continue;
               if (!CFNumberGetValue((CFNumberRef)object, kCFNumberLongType, &number))continue;
               cookie = (IOHIDElementCookie)number;

               object = CFDictionaryGetValue(element, CFSTR(kIOHIDElementUsageKey));
               if (object == 0 || CFGetTypeID(object) != CFNumberGetTypeID())   continue;
               if (!CFNumberGetValue((CFNumberRef)object, kCFNumberLongType, &number))continue;
               usage = number;

               object = CFDictionaryGetValue(element, CFSTR(kIOHIDElementUsagePageKey));
               if (object == 0 || CFGetTypeID(object) != CFNumberGetTypeID())   continue;
               if (!CFNumberGetValue((CFNumberRef)object, kCFNumberLongType, &number))continue;
               usagePage = number;

               if (usagePage == kHIDPage_GenericDesktop) {  /* GenericDesktop Page (0x01) */
                    if (usage == kHIDUsage_GD_Joystick)          ;
                    else if (usage == kHIDUsage_GD_X)       cookies->xAxis = cookie;
                    else if (usage == kHIDUsage_GD_Y)       cookies->yAxis = cookie;
               }
               else if (usagePage == kHIDPage_Button) {     /* Button Page (0x09) */
                    if (usage == kHIDUsage_Button_1)        cookies->btn1 = cookie;
                    else if (usage == kHIDUsage_Button_2)   cookies->btn2 = cookie;
                    else if (usage == kHIDUsage_Button_3)   cookies->btn3 = cookie;
                    else if (usage == kHIDUsage_Button_4)   cookies->btn4 = cookie;
                    else if (usage == 0x05)                 cookies->btn5 = cookie;
                    else if (usage == 0x06)                 cookies->btn6 = cookie;
               }
               else if (usagePage == kHIDPage_VendorDefinedStart) {/* VendorDefined (0xFF00) */
                    if (usage == 0x01)                      cookies->vd1 = cookie;
                    else if (usage == 0x02)                 cookies->vd2 = cookie;
               }
          }
     }
     else { printf("copyMatchingElements failed with error %d\n", success); }
     printf(
"XAxis[%lx], YAxis[%lx], Button[%lx][%lx][%lx][%lx][%lx][%lx], VendorDefined[%lx][%lx]\n",
          (long)cookies->xAxis, (long)cookies->yAxis,
          (long)cookies->btn1, (long)cookies->btn2,
          (long)cookies->btn3, (long)cookies->btn4,
          (long)cookies->btn5, (long)cookies->btn6,
          (long)cookies->vd1, (long)cookies->vd2);
     return cookies;
}
```

HID デバイスにどのような要素が存在するかは、以下のヘッダファイルを調べることによって確認することができます。

/System/Library/Frameworks/IOKit.framework/Versions/A/Headers/hid/IOHIDUsageTables.h

では、実際の HID デバイスで使用可能な要素は、どうすれば知ることができるのでしょうか？
著者の場合、前ページのコードでは、全要素のリストを得るということで、変数 usage と usagePage の値をダンプして、 IOHIDUsageTables.h の定義と比較することによって、使える要素を調べました。

column USB-IF の紹介

USB-IF（USB Implementers Forum, Inc.）は、USB の仕様を策定した各企業によって設立された非営利団体です。USB 技術そのものについての情報は、ここのサイトから入手できますので、紹介しておきます。

- USB-IF の Web サイト
  http://www.usb.org/

- USB 2.0 の仕様書
  http://www.usb.org/developers/docs/usb_20_05122006.zip

- HID の情報
  http://www.usb.org/developers/hidpage/

- HID の仕様書
  http://www.usb.org/developers/devclass_docs/HID1_11.pdf

- HID Usage Table のドキュメント
  http://www.usb.org/developers/devclass_docs/Hut1_12.pdf

■ユーザ操作

ここでは、ユーザによるジョイスティックの十字キーやボタン操作を監視する方法を説明します。以下がサンプルコードです。

gamepad.c / HIDAdded()の抜粋

```
void HIDAdded(void *refCon, io_iterator_t iterator)
{
      . . .
     while (hidDevice = IOIteratorNext(iterator)) {
          . . .
          if ((hidDeviceInterface != NULL) && (gHIDDataPtr == NULL)) {
               gHIDDataPtr = calloc(1, sizeof(HIDData));
               gHIDDataPtr->hidDeviceInterface
                    = (IOHIDDeviceInterface122 **)hidDeviceInterface;
(1)            gHIDDataPtr->cookies = cookies;

(2)            kr = (*(gHIDDataPtr->hidDeviceInterface))->open(gHIDDataPtr->hidDeviceInterface,
                    kIOHIDOptionsTypeNone);
               . . .
               /* HID Interface */
(3)            DumpHIDInterface((IOHIDDeviceInterface **)gHIDDataPtr->hidDeviceInterface,
                    gHIDDataPtr->cookies);

               /* Polling */
(4)            DumpQueues((IOHIDDeviceInterface **)gHIDDataPtr->hidDeviceInterface,
                    gHIDDataPtr->cookies);

               /* Using Queue Callbacks */
(5)            pass = SetupQueue(gHIDDataPtr);

(6)            pass = SetupInterrupt(gHIDDataPtr->hidDeviceInterface);

(7)            IOServiceAddInterestNotification(
                              gNotifyPortRef,              // notifyPort
                              hidDevice,                   // service
                              kIOGeneralInterest,          // interestType
                              DeviceNotification,          // callback
                              gHIDDataPtr,                 // refCon
                              &(gHIDDataPtr->notification) // notification
                              );
               }
        }
}
```

上記コードの番号が振られた箇所について、以下で説明します。

(1) クッキーの設定
前に説明した関数 GetHIDCookies() で取得したデータです。ジョイスティックから値を取得する際に必要となります。

(2) Device Interface の open
HID の Device Interface の open() 関数を呼んでいます。Device Interface 利用開始時に呼ぶべき関数は Device Interface の種類毎に異なります。HID の場合は、open() だということです。

(3) ダイレクトに値を取得
HID の Device Interface の HID Manager のキューを使用しないで、値を取得する関数の呼び出しです。この関数の詳細は、後のページで説明します。

(4) ポーリングによる値の取得
HID Manager のキューを使用して、ポーリングによる値の取得を行っている関数の呼び出しです。この関数の詳細は、後のページで説明します。

(5) コールバックによる値の取得
⑷と異なり、コールバックで値をする為の手続きを行っている関数の呼び出しです。この関数の詳細は、後のページで説明します。

(6) ユーザ操作の監視
キー押下等のユーザによる何らかの操作を監視する為のコールバックを設定する関数の呼び出しで。この関数の詳細は、後のページで説明します。

(7) 状態監視
ジョイスティックが抜かれた場合に呼ばれるコールバック関数 DeviceNotification() を登録しています。このコールバック関数のコードは、以下のとおりです。

gamepad.c / DeviceNotification()

```
void DeviceNotification(void *refCon, io_service_t service, natural_t messageType,
          *messageArgument )
{
    kern_return_t   kr;
    HIDDataPtr hidDataPtr = (HIDDataPtr) refCon;
    if ((hidDataPtr != NULL) && (messageType == kIOMessageServiceIsTerminated)) {
        if (hidDataPtr->hidQueueInterface != NULL) {
            kr = (*(hidDataPtr->hidQueueInterface))->stop((hidDataPtr->hidQueueInterface));
            kr = (*(hidDataPtr->hidQueueInterface))->dispose((hidDataPtr->hidQueueInterface));
            kr = (*(hidDataPtr->hidQueueInterface))->Release (hidDataPtr->hidQueueInterface);
            hidDataPtr->hidQueueInterface = NULL;
        }
        if (hidDataPtr->hidDeviceInterface != NULL) {
            kr = (*(hidDataPtr->hidDeviceInterface))->close (hidDataPtr->hidDeviceInterface);
            kr = (*(hidDataPtr->hidDeviceInterface))->Release (hidDataPtr->hidDeviceInterface);
            hidDataPtr->hidDeviceInterface = NULL;
        }
        if (hidDataPtr->notification != IO_OBJECT_NULL) {
            kr = IOObjectRelease(hidDataPtr->notification);
            hidDataPtr->notification = IO_OBJECT_NULL;
        }
    }
}
```

○ダイレクトに値を取得

HID の Device Interface の関数 getElementValue() を使って、値を取得します。この方法では、常に値を取得していませんと、取りこぼしが発生してしまいます。

gamepad.c / DumpHIDInterface()

```
void DumpHIDInterface(IOHIDDeviceInterface ** hidDeviceInterface, HIDCookiePtr cookies)
{
     HRESULT             result;
     IOHIDEventStruct    hidEvent;
     long                index;
     printf("START: Checking hidEvent.value\n");
     for (index = 0; index < 10; index++) {
          long xAxis, yAxis, btn1, btn2, btn3, btn4, btn5, btn6, vd1, vd2;
          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->xAxis,
               &hidEvent);
          xAxis = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->yAxis,
               &hidEvent);
          yAxis = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn1,
               &hidEvent);
          btn1 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn2,
               &hidEvent);
          btn2 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn3,
               &hidEvent);
          btn3 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn4,
               &hidEvent);
          btn4 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn5,
               &hidEvent);
          btn5 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->btn6,
               &hidEvent);
          btn6 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->vd1,
               &hidEvent);
          vd1 = hidEvent.value;

          result = (*hidDeviceInterface)->getElementValue(hidDeviceInterface, cookies->vd2,
               &hidEvent);
          vd2 = hidEvent.value;

          printf("%ld %ld %s%s%s%s%s%s%s%s\n", xAxis, yAxis, btn1 ? "button1 " : ""
               btn2 ? "button2 " : "", btn3 ? "button3 " : "", btn4 ? "button4 " : "",
               btn5 ? "button5 " : "", btn6 ? "button6 " : "", vd1 ? "vd1 " : "",
               vd2 ? "vd2 " : "");
          sleep(1);
        }
        printf("STOP: Checking hidEvent.value\n");
}
```

○ポーリングによる値の取得

HID の Device Interface の HID Manager のキューを利用すれば、ユーザ操作の取りこぼしを防ぐことが可能となります。ただし、ユーザ操作の有無にかかわらず、値をチェックしないといけませんので、効率に難点があります。

gamepad.c / DumpQueues()

```
void DumpQueues(IOHIDDeviceInterface **hidDeviceInterface, HIDCookiePtr cookies)
{
     HRESULT             result;
     IOHIDQueueInterface **queue;
     long                index;
     IOHIDEventStruct    event;

     queue = (*hidDeviceInterface)->allocQueue(hidDeviceInterface);
     if (queue) {
          result = (*queue)->create(queue, 0, 8);
(1)       result = (*queue)->addElement(queue, cookies->xAxis, 0);
          result = (*queue)->addElement(queue, cookies->yAxis, 0);
          result = (*queue)->addElement(queue, cookies->btn1, 0);
          result = (*queue)->addElement(queue, cookies->btn2, 0);
          result = (*queue)->addElement(queue, cookies->btn3, 0);
          result = (*queue)->addElement(queue, cookies->btn4, 0);
          result = (*queue)->addElement(queue, cookies->btn5, 0);
          result = (*queue)->addElement(queue, cookies->btn6, 0);
          result = (*queue)->addElement(queue, cookies->vd1, 0);
          result = (*queue)->addElement(queue, cookies->vd2, 0);

          result = (*queue)->start(queue);
          sleep(1);
          printf("START: Checking queue\n");
          for (index = 0; index < 10; index++) {
               AbsoluteTime   zeroTime = {0,0};
(2)            result = (*queue)->getNextEvent(queue, &event, zeroTime, 0);
               if (result)
                    printf("Queue getNextEvent result: %lx\n", result);
               else
                    printf("Queue: event:[%lx] %ld\n",
                              (unsigned long) event.elementCookie, event.value);
               sleep(1);
          }
          printf("STOP: Checking queue\n");
          result = (*queue)->stop(queue);
          result = (*queue)->dispose(queue);
          (*queue)->Release(queue);
        }
}
```

上記コードの番号が振られた箇所について説明します。

(1) キューで管理する属性を登録
HID Manager は登録されたクッキーの属性のみをキューで管理しますので、このような登録は必要となります。

(2) イベントの取得
キューに溜まっているイベントを一個ずつ溜まった順番に取得します。

○コールバックによる値の取得

　ポーリングによる値の取得は、効率が悪いということで、HID Manager では、コールバックによる非同期の値の取得が可能となっています。以下のコードでは、関数 SetupQueue() でキューの用意とコールバック関数の登録を行い、コールバック関数 QueueCallbackFunction() でキューに溜まっているイベントを取得しています。

gamepad.c / SetupQueue()

```
_Bool SetupQueue(HIDDataPtr hidDataPtr)
{
     HRESULT                 result;
     IOHIDDeviceInterface    **hidDeviceInterface
          = (IOHIDDeviceInterface **)hidDataPtr->hidDeviceInterface;
     HIDCookiePtr            cookies = hidDataPtr->cookies;
     IOHIDQueueInterface     **queue;
     IOReturn                ioret;
     _Bool                   boolRet = true;

     queue = (*hidDeviceInterface)->allocQueue(hidDeviceInterface);
     hidDataPtr->hidQueueInterface = queue;
     if (queue) {
          result = (*queue)->create(queue, 0, 8);
          result = (*queue)->addElement(queue, cookies->xAxis, 0);
          result = (*queue)->addElement(queue, cookies->yAxis, 0);
          result = (*queue)->addElement(queue, cookies->btn1, 0);
          result = (*queue)->addElement(queue, cookies->btn2, 0);
          result = (*queue)->addElement(queue, cookies->btn3, 0);
          result = (*queue)->addElement(queue, cookies->btn4, 0);
          result = (*queue)->addElement(queue, cookies->btn5, 0);
          result = (*queue)->addElement(queue, cookies->btn6, 0);
          result = (*queue)->addElement(queue, cookies->vd1, 0);
          result = (*queue)->addElement(queue, cookies->vd2, 0);

          ioret = (*queue)->createAsyncEventSource(queue, &hidDataPtr->eventSource);
          ioret = (*queue)->setEventCallout(queue, QueueCallbackFunction, NULL, hidDataPtr);

          CFRunLoopAddSource(CFRunLoopGetCurrent(), hidDataPtr->eventSource,
                    kCFRunLoopDefaultMode);
          ioret = (*queue)->start(queue);
     }
     return boolRet;
}
```

gamepad.c / QueueCallbackFunction()

```
void QueueCallbackFunction(void *target, IOReturn result, void *refcon, void *sender)
{
     HIDDataPtr          hidDataPtr = (HIDDataPtr)refcon;
     AbsoluteTime        zeroTime = {0,0};
     IOHIDEventStruct    event;
     while (result == kIOReturnSuccess)  {
          result = (*hidDataPtr->hidQueueInterface)->getNextEvent(
                    hidDataPtr->hidQueueInterface, &event, zeroTime, 0);
          if (result)    printf("Queue getNextEvent result: %lx\n", result);
          else           printf("Queue: event:[%lx] %ld\n",
                                   (unsigned long) event.elementCookie, event.value);
     }
}
```

○ユーザ操作の監視

Mac OS X 10.3から、割り込みのような仕組みでユーザ操作の監視が可能となりました。

以下のコードでは、関数 SetupInterrupt() でコールバックの設定を行い、ユーザがキー押下などの操作を行いますと、コールバック関数 InterruptReportCallbackFunction() が呼ばれるようになっています。

gamepad.c / SetupInterrupt()

```
_Bool SetupInterrupt(IOHIDDeviceInterface122 **hidDeviceInterface)
{
     _Bool               boolRet = true;
     kern_return_t       kr;
     CFRunLoopSourceRef  eventSource;
     static UInt8        buffer[HIDBUFSIZ];

     kr = (*hidDeviceInterface)->createAsyncEventSource(hidDeviceInterface, &eventSource);
     kr = (*hidDeviceInterface)->setInterruptReportHandlerCallback(hidDeviceInterface, buffer,
               sizeof(buffer), InterruptReportCallbackFunction, NULL, buffer);
     CFRunLoopAddSource(CFRunLoopGetCurrent(), eventSource, kCFRunLoopDefaultMode);
     return boolRet;
}
```

gamepad.c / InterruptReportCallbackFunction()

```
void InterruptReportCallbackFunction(void *target, IOReturn result, void *refcon,
          void *sender, UInt32 bufferSize)
{
     UInt8     *buffer = (UInt8 *)refcon;
     int index;

     if (!buffer)   return;
     printf("Buffer = ");
     for (index=0; index < bufferSize; index++)   printf("%2.2x ", buffer[index]);
     printf("\n");
}
```

■おわりに

本発表は、当初は Classic Gamepad のデバイスドライバ開発にチャレンジということで着手したのですが、Device Interface の奥深さを知り、方向転換をしました。

したがいまして、機会がありましたら、このやり残した、デバイスドライバの開発に、挑戦したいと思っています。