Fakultät für Informatik

Bachelor-Arbeit

Entwicklung eines Windows Treibers für die Nintendo Wii Remote

---

Vorgelegt an der SRH Hochschule Heidelberg

bei Prof. Dr. Daniel Görlich und Nick Prühs

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Vorgelegt von Julian Löhr

Eingereicht am 09.10.2013

Ich versichere, dass ich die Kapitel der Arbeit, für die ich als Verfasser genannt werde, selbständig verfasst habe, dass ich keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt habe und dass ich diese Arbeit bei keinem anderen Prüfungsverfahren vorgelegt habe.

Ort, Datum Unterschrift

This paper is about the development of a driver for Windows PC to use the Nintendo Wii Remote as gamecontroller. The Wii Remote is not compatible with the standard Windows providen driver. Although the Wii Remote is identified as a game controller, none of its input is exposed.

The basics of how Windows accesses gamecontroller and the technical side of the Wii Remote are reviewed. Additionaly the reason why the Wii Remote is not compatible with the standard Windows driver is figured out. Then those information are put together to form a basic concept. The basic concept includes how to hold the controller to control games and how to solve the issue.

Later on the basics of driver development for Windows are examined and four approaches are considered. Just one of those approaches is suitable and is therefore developed into an elaborated concept.

At the end the concept is partitialy implemented to prove wether the concept works as expected.

Inhaltsverzeichnis

1. Motivation

Viele Windows PC-Spiele verfügen über Gamecontroller-Unterstützung, so gibt es inzwischen viele Multiplattform-Titel, welche für Windows PC und Spielekonsolen entwickelt werden. Diese haben meist eine für Gamepad optimierte Steuerung. Valves neue Steam Big Picture-Software soll nun auch den PC als Konsolenersatz an den Fernseher bringen und ist auf eine Steuerung per Gamepad ausgelegt.

Die Nintendo Wii Remote ist ein Wireless Gamecontroller. Er wird zu Bedienung der Nintendo Wii und der Nintendo Wii U eingesetzt. Beide Konsolen sind sehr beliebt, sodass auch der Gamecontroller entsprechend oft in Spielerhaushalten vorhanden sein sollte. Mit dem Gamecontroller sollte es auch möglich sein Spiele am Windows PC zu bedienen, da er eine weitverbreitete Alternative zu sonstigen Gamecontroller darstellt.

2. Stand der Technik

Die Nintendo Wii Remote lässt sich per Bluetooth mit dem PC verbinden und wird auch als Gamecontroller erkannt. In Abbildung 2-1 ist auf der linken Seite der Eigenschaften Test Dialog der Wii Remote zu sehen. Dort ist zu erkennen, dass die Inputs, d.h. Buttons, Sensoren und Analog-Sticks, nicht verfügbar sind, womit diese nicht benutzbar sind um Spiele zu steuern. Zum Vergleich ist in Abbildung 2-1 auf der rechten Seite der Eigenschaften Test Dialog des funktiontstüchtigen Logitech F710 Wireles Gamepad mit seinen bereitgestellten Inputs dargestellt.

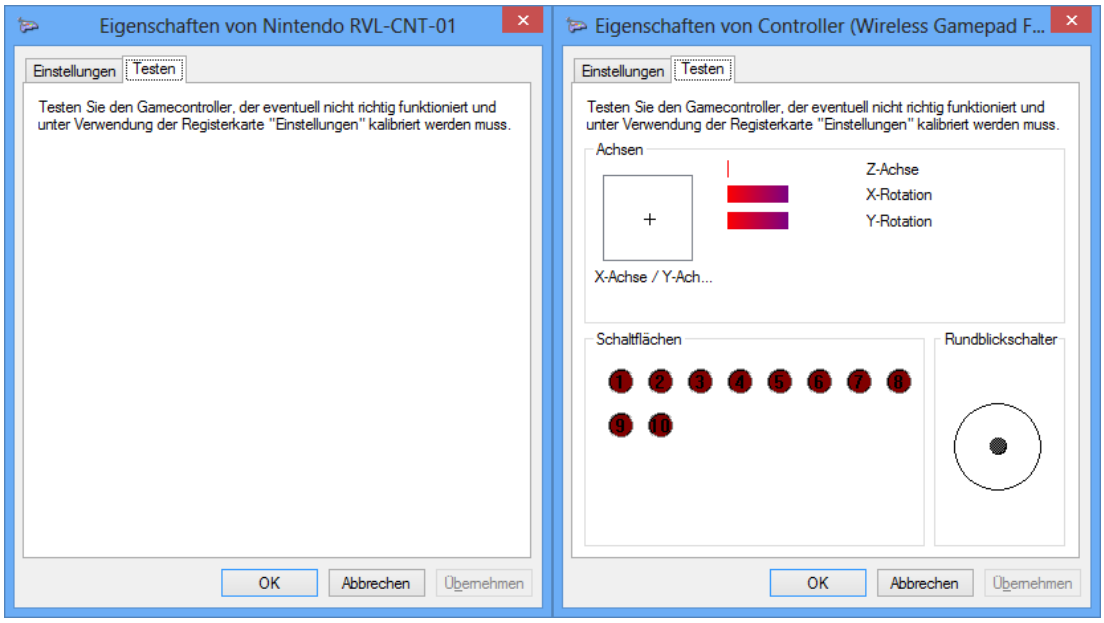

Abbildung 2-1: Eigenschaften Test Dialog der Wii Remote (Links) und des Logitech F710 (Rechts)

Es gibt zwar Programme, welche der Wii Remote eine Funktionalität geben, jedoch sind diese veraltet und funktionieren nicht einwandfrei unter Windows 7 und Windows 8[Loeh2013]. Außerdem sind diese nur bedingt zur Bedienung von Spielen geeignet, sodass die Wii Remote unter den aktuellen Windows Versionen nicht für die Spielsteuerung eingesetzt werden kann[Loeh2013].

3. Ziel und Anforderungen

Das Ziel der Arbeit ist es einen Windows Treiber für die Nintendo Wii Remote zu entwickeln. Dieser soll den Einsatz der Wii Remote als Gamecontroller in Spielen ermöglichen. Da laut Steam-Statistik ca. 64 Prozent der Spieler Windows 7 benutzen und um die aktuelle Windows Version zu unterstützen, wird der Treiber für Windows 7 und Windows 8 entwickelt[Valv2013].

4. Gamecontroller Grundlagen

4.1 DirectInput und XInput

Es gibt zwei APIs um Gamecontroller unter Windows anzusprechen. Die neuere API XInput wurde für Xbox 360 Controller entwickelt, während die ältere API DirectInput seit DirectX Version 1.0 verfügbar ist und allgemein Gamepads, Joysticks und weitere Eingabegeräte verwendbar macht[MSDN2013a; MSDN2013b].

Damit ein Gerät vom Betriebssystem der DirectInput API bereitgestellt wird, muss dieses den Human Interface Device Standard, kurz HID, unterstützen[MSDN2013a]. Der Windows HID-Gerätetreiber ist das Gegenstück zur API. Dieser betreibt und verwaltet alle HID-Geräte und wird standardmäßig von Microsoft bereitgestellt.

Zu der XInput API ist der Treiber XUSB das Gegenstück[MSDN2013a]. Dieser stellt den Gamecontroller sowohl der XInput API, als auch der DirectInput API zur Verfügung, sodass auch ältere Spiele mit dem Xbox 360 Controller bedient werden können[MSDN2013a]. Der Vorteil von Xbox 360 Controllern ist, dass diese immer ein gleiches Layout besitzen. So kann bei Spielen, welche für Xbox 360 und Windows entwickelt werden, die Steuerung für Gamepads von der Konsolenversion übernommen werden. Microsoft empfiehlt XInput anstatt von DirectInput zu verwenden[MSDN2013a].

Zu XUSB hat Mircosoft keine Dokumentation veröffentlicht. Somit müsste zunächst der Treiber mittels Reverse Engineering analysiert werden. Da dies jedoch den Umfang dieser Arbeit sprengen würde, wurde entschieden einen Treiber für die ältere API DirectInput zu entwickeln.

4.2 Human Interface Device

Ein Human Interface Device ist eine USB-Geräte Klasse, also ein Teil des USB-Standards[USBI2001, S. 1]. Inzwischen haben auch andere Übertragungstandards das HID-Protokoll in ihre Spezifikationen aufgenommen, sodass nicht nur USB, sondern auch Bluetooth und Inter-Integrated Circuit, kurz I²C, für die Kommunikation genutzt werden kann[MSDN2013c]. Human Interface Devices sind primär Geräte mit denen ein Mensch ein Computersystem bedienen kann. Diese können sowohl Eingaben entgegennehmen, als auch kleine Ausgaben, wie Force Feedback, Audio oder kleine Anzeigen, wiedergeben[USBI2001, S. 1-2]. Aber auch Geräte die keine direkte Bedienung benötigen, aber Daten in einem ähnlichen Format bereitstellen, wie Barcode Scanner, Thermometer oder Voltmeter, können das HID-Protokoll benutzen[USBI2001, S. 2].

HID ist wie USB selbstbeschreibend und besteht aus Descriptoren und Reports[USBI2001, S. 2]. Zu Beginn werden mittels Descriptoren das Gerät und dessen Eigenschaften beschrieben. So beschreibt der Report Descriptor die Struktur und Größe der Reports, sodass diese von Gerät zu Gerät unterschiedlich sein können[USBI2001, S. 4 - 5]. Die Reports sind die eigentlichen Ein- und Ausgabedaten, die somit je nach Bedarf in Größe und Struktur variieren können. Es gibt folgende drei Report Typen: Input und Output Reports enthalten Daten die vom Gerät als Eingabe gelesen oder als Ausgabe ausgegeben werden[USBI2001, S. 29]; Feature Reports werden an das Gerät gesendet, um diese zu konfigurieren[USBI2001, S. 29]. Der USB-HID-Standard benutzt zur Kommunikation die USB-Pipes. Es gibt eine Control-Pipe und eine Interrupt-Pipe[USBI2001, S. 10]. Die Control-Pipe wird zum Austausch von Steuerungsdaten, zur Übertragung von angeforderten Reports und zur Übertragung von Daten zum Gerät genutzt und ist somit bidirektional[USBI2001, S. 10]. Die Interrupt-Pipe kann unidirektional sein, muss aber vom Gerät zum Host gerichtet sein, sodass dieses asynchrone nicht angeforderte Daten, wie Input Reports, übertragen kann. Die Gegenrichtung, also vom Host zum Gerät, ist optional und würde zur Übertragung von Output Reports benutzt werden[USBI2001, S. 10]. Diese können jedoch auch über die Control Pipe gesendet werden[USBI2001, S. 10].

4.3 Bluetooth HID-Profile

Der Bluetooth-Standard definiert verschiedene Profiles, dies sind Anwendungen oder Dienste, die ein Bluetooth-Gerät unterstützt oder anbietet[BSIG2013a]. So gibt es zum Beispiel das Headset Profile, welches beschreibt wie ein Bluetooth Headset mit einem anderen Bluetooth-fähigen Gerät kommuniziert, oder das Message Access Profile, welches verwendet wird um Textnachrichten zwischen Geräten auszutauschen[BSIG2013a].

Das HID-Profile wurde entwickelt, damit HIDs über eine drahtlose Verbindung kommunizieren können[BSIG2012, S. 12]. Es wurde vieles aus dem USB-Standard übernommen und die USB spezifischen Eigenschaften dem Bluetooth-Standard angepasst[BSIG2012, S. 12].

Zum Bluetooth-Standard gehört unteranderem das Service Discovery Protocol, kurz SDP. Dieses gibt mittels Records Auskunft über verschiedene Eigenschaften des Gerätes, so auch über die unterstützten Profile[BSIG2012, S. 20]. Die SDP-Records haben eine ähnliche Funktion wie die USB-Discriptoren, so beinhaltet das HID-Profile den Report Descriptor des HIDs[BSIG2012, S. 20].

Zur direkten Kommunikation mit dem Gerät wird das Logical Link Control and Adaptation Protocol, kurz L2CAP, benutzt[BSIG2012, S. 24]. Dieses benutzt einen Asynchronous Connection-Less Link, kurz ACL Link, um Daten mit dem Gerät auszutauschen. Das Protokoll übernimmt dabei die Sicherung des Quality of Service und das Multiplexing bei mehreren Anwendungen[BSIG2013b]. Ähnlich den Ports bei Netzwerkprotokollen, benutzt L2CAP sogenannte Channels, welche logische Verbindungen zum Gerät darstellen und mit einem Protocol/Service Multiplexer Wert, kurz PSM, identifiziert werden[BSIG2013b]. Diese Channels werden analog zu den USB-Pipes genutzt. Es werden zwei Channels benötigt, einen Control Channel und einen Interrupt Channel[BSIG2012, S. 24].

Der Datenaustausch erfolgt über Bluetooth HID Protocol Messages, diese bestehen aus einem ein Byte großen Header und aus den zu übertragenen Daten. Jedoch bestehen einige Nachrichten nur aus einem Header, sodass der Header auch zur Steuerung benutzt wird. Der Header ist in zweimal vier Bits unterteilt. Die ersten vier Bits definieren den Nachrichten Typ und die letzten Vier fungieren als Parameter. In der Tabelle 4-1 sind die von der aktuellen Version 1.1 des HID-Profile unterstützten Nachrichten Typen aufgelistet.

Tabelle 4-1: Die unterstützten HIDP Nachrichten Typen[BSIG2012, S. 24]

Hex	Message Type	Sent By	Message Length (Octets)
0	HANDSHAKE	Device	1
1	HID_CONTROL	Device and Host	1
2-3	Reserved	N/A	
4	GET_REPORT	Host	1 to 4
5	SET_REPORT	Host	1 + Report data payload
6	GET_PROTOCOL	Host	1
7	SET_PROTOCOL	Host	1
8	GET_IDLE[DEPRECATED]	Host	1
9	SET_IDLE[DEPRECATED]	Host	2
A	DATA	Device and Host	1 + Report data payload
B	DATC[DEPRECATED]	Device and Host	1 + Continuation of report data payload

Die HANDSHAKE Nachricht wird zum Bestätigen von GET_REPORT, SET_REPORT, GET_PROTOCOL und SET_PROTOCOL oder im Falle eines Fehlers verwendet[BSIG2012, S. 25]. Sie wird immer von Gerät über den Control Channel gesendet[BSIG2012, S. 25]. Der Parameter beinhaltet im Falle eines Fehlers einen Code, welcher den Fehler beschreibt[BSIG2012, S. 25].

HID_CONTROL wird benutzt um den Status des Gerätes zu ändern, z.B. um es in den Energiesparmodus oder aus diesem wieder in den aktiven Modus zu versetzen[BSIG2012, S. 26-27]. Der Parameter 0x3 steht für SUSPEND und signalisiert dem Gerät in den Energiesparmodus zu wechseln, während 0x4 für EXIT_SUSPEND steht und den Energiesparmodus wieder beendet.

Zum Anfordern eines Reports wird GET_REPORT verwendet[BSIG2012, S. 28]. Es wird nur vom Host verwendet und kann optional eine Report ID angeben, wodurch die Nachricht dann 2 Byte groß wäre[BSIG2012, S. 30]. Weiterhin kann optional noch mit 2 weiteren Bytes die Größe des angeforderten Reports angegeben werden[BSIG2012, S. 30]. Der Parameter im ersten Byte gibt den Reporttypen an[BSIG2012, S. 30]. Ist dieser Output oder Feature, so wird der letzte empfangende Output Report oder die aktuellen Konfigurationswerte zurückgegeben[BSIG2012, S. 29]. Der angeforderte Report wird mit einer DATA Nachricht über den Control Channel zurückgesendet[BSIG2012, S. 28].

Mit SET_REPORT kann der Host einen Report an das Gerät übertragen[BSIG2012, S. 31]. Der Parameter des Headers gibt den Typ des Reports an[BSIG2012, S. 32]. Nach dem Header folgen die Daten des Reports[BSIG2012, S. 32].

GET_PROTOCOL und SET_PROTOCOL werden zum Lesen und Schreiben des Protocol Modes benutzt[BSIG2012, S. 32]. Diese sind jedoch für einen Gamecontroller nicht notwendig, da das standardmäßige Report Protocol benutzt wird[BSIG2012, S. 22].

Der Nachrichtentyp DATA wird zum Senden von Reports benutzt. Der Parameter gibt dabei den Reporttyp an: 0x0 steht Other, 0x1 für Input, 0x2 für Output und 0x3 für Feature[BSIG2012, S. 35]. Der Reporttype Other wird für die Antwort der GET_PROTOCOL Anfrage genutzt, während die Anderen entsprechend dem zu übertragendem Report zu setzen sind[BSIG2012, S. 34]. Zum Senden von nicht angeforderten Reports wird die Interrupt Pipe benutzt, dabei muss es sich bei einer Übertragung vom Host zu Gerät um einen Output Report handeln und bei einer Übertragung vom Gerät zum Host um einen Input Report[BSIG2012, S. 34]. Dem Header folgen die Daten des Reports.

4.4 Wii Remote

Die Nintendo Wii Remote ist der drahtlose Controller der Spielkonsole Nintendo Wii. Die Nachfolgekonsole Nintendo Wii U unterstützt ebenfalls die Wii Remote und benötigt diese zum Steuern einiger Spiele. Neben der ursprünglichen Version der Wii Remote gibt es auch die Wii Remote Plus, welche die Erweiterung Wii Motion Plus bereits integriert hat.

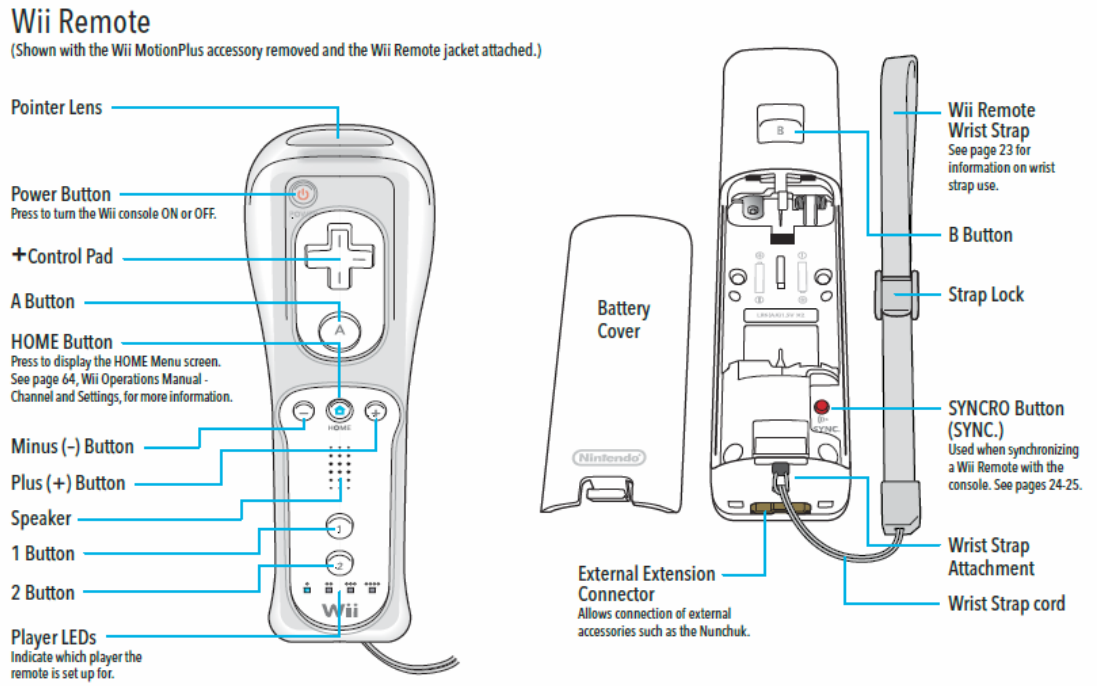

Abbildung 4-1: Die Nintendo Wii Remote mit Beschriftung[Nint2009, Seite 6]

4.4.1 Gamecontrollereigenschaften

Der Controller verfügt über 11 Tasten, einen 3-Achsen Beschleunigungssensor und einen Infrarotsensor[Wiib213]. Der Infrarotsensor erkennt bis zu 4 Infrarotpunkte und bestimmt deren Position[Wiib213]. Die der Wii und Wii U mitgelieferten Sensorleiste besteht aus zwei 20cm entfernten LED-Clustern und wird beim normalen Gebrauch auf oder unter dem Fernseher platziert. Damit lässt sich die Position und Bewegung der Wii Remote relativ zum Fernseher berechnen und man kann mit ihr durch Zeigen einen Cursor steuern. Für Feedback verfügt der Controller über einen Vibrationsmotor, einem kleinen Lautsprecher für kurze Sound Effekte und, wie in Abbildung 4-1 gezeigt, vier neben einander gereihte LEDs[Wiib213]. Ist die Wii Remote mit einer Wii oder Wii U verbunden zeigt eine LED die Spielernummer des Controllers an. Außerdem ist auch noch ein 16 kByte großer EEPROM Speicher vorhanden, welcher zum einen Kalibrierungsdaten enthält und zum anderen beliebig genutzt werden kann[Wiib213]. Der Power Button oder der rote Sync Knopf gehört in dem Sinne nicht zu den Tasten, da sie keinen Input für den Gamecontroller generieren. Der Power Button schaltet die Wii Remote aus und trennt die Verbindung zum Host[Wiib213].

Die Wii Remote benutzt zur Kommunikation mit einem Host Bluetooth und ist somit ein kabelloser Controller[Wiib213]. Für die Stromversorgung werden zwei AA-Batterien benötigt, welche in einem Batteriefach untergebracht sind. Durch Betätigen des roten Sync Knopfes im Batteriefach oder gleichzeitiges Drücken der Tasten 1 und 2 wird die Wii Remote in einen Discovery Modus versetzt, in welchem diese Verbindungsanfragen annimmt und so mit einem Bluetooth Host verbunden werden kann. In diesem Modus blinken die LEDs, jedoch nur so viel wie dem aktuellen Batteriestand entsprechen, sodass bei vollen Batterien alle Vier blinken und bei fast Aufgebrauchten nur Eine.

In Abbildung 4-1 links ist am unteren Ende des Controllers eine 6-pin Buchse vorhanden, welche zum Anstecken von Erweiterungen gedacht ist. Die Wii Motion Plus Erweiterung besteht aus einem 3-Achsen Gyroskop und einer weiteren Erweiterungsbuchse. Die Wii Remote Plus hat diese Erweiterung bereits integriert.

Wie in Abbildung 4-2 zu sehen, ist der Nunchuk ein kleiner Controller für die andere Hand und erweitert den Controller um einen Analogstick, zwei weitere Tasten und besitzt einen eigenen 3-Achsen Beschleunigungssensor[Wiib2013]. Der Classic Controller und Classic Controller Pro ist einem Gamepad nachempfunden und hat ein ähnliches Layout wie die Xbox 360 Controller. So soll diese Erweiterung die Wii Remote nicht erweitern sondern als Controller ersetzen und die Wii Remote zur Kommunikation und Identifikation des Spielers nutzen. Dies wird dadurch ersichtlich, da das Gamepad für zwei Hände konzipiert ist und auch ein Steuerkreuz und die Tasten A, B, +, - und Home besitzt. Weiterhin besitzen die Controller zwei Analogsticks, zwei Schultertasten, zwei druckintensive Schultertasten und zwei Tasten in der Nähe der A und B Tasten. Der Unterschied zwischen Classic Controller und Classic Controller Pro besteht nur in ihrer Form und Größe.

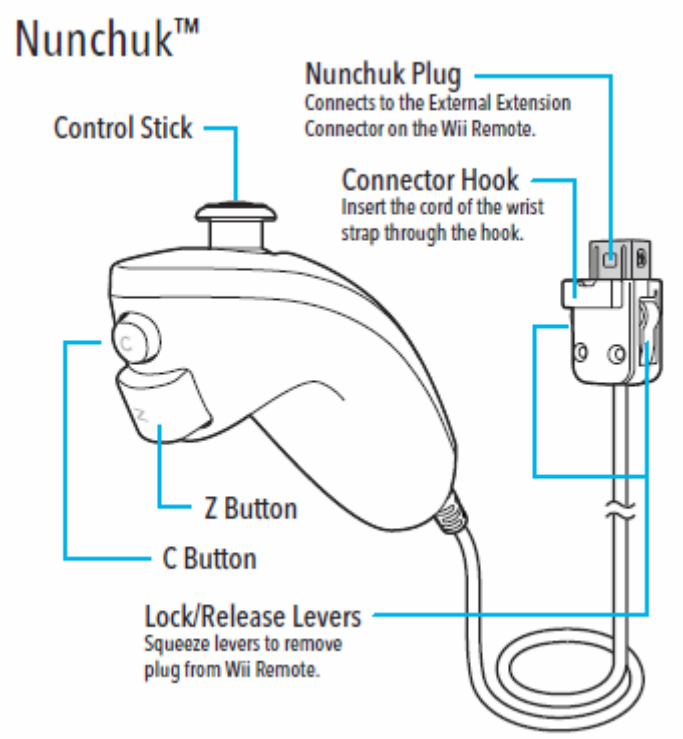

Abbildung 4-2: Der Nunchuk mit Beschriftung[Nint2009, Seite 8]

4.4.2 Technik

Beim Verbindungsaufbau zwischen einer Wii Remote und einem Host ist das Pairing, d.h. die Eingabe eines PINs, optional[Wiib2013]. Wird das Pairing ausgeführt, speichert die Wii Remote bis zu drei Host Adressen[Wiib2013]. Ist die Wii Remote ausgeschaltet und wird eine Taste gedrückt, welche nicht den Discovery Modus startet, versucht sie automatisch die Verbindung zum letzten Host wieder herzustellen[Wiib2013]. Diese Funktion wird zum Einschalten der Konsole per Wii Remote benutzt.

Für die Kommunikation wird das HID-Protokoll in Form des Bluetooth HID-Profiles benutzt[Wiib2013]. Dabei wird der L2CAP-Channel mit dem PSM 0x11 als Control-Channel und PSM 0x13 wird für den Interrupt-Channel benutzt[Wiib2013]. Die Wii Remote ist jedoch nicht gänzlich HID konform, was der Grund ist, weshalb sie nicht mit dem Standard HID-Gerätetreiber von Windows kompatibel ist und nicht ohne Zusatzsoftware zur Steuerung von Spielen an PCs eingesetzt werden kann. So wird ein unvollständiger Report Descriptor vom Gerät zurückgegeben. Dieser beschreibt zwar die Größe und Anzahl der Reports, nicht aber deren Inhalt und Struktur[Wiib2013]. Außerdem wird nur der Interrupt-Channel benutzt, was darin resultiert, dass nur Input und Output Reports gesendet werden können[Wiib2013]. Die Wii Remote benutzt jedoch einige Output Reports um Konfigurationen und Kalibrierungen auszuführen, für welche der HID-Standard Feature Reports vorsieht. Es existiert auch ein Output Report mit dem sich ein Report anfordern lässt, für diesen Fall ist laut HID-Profile die GET_REPORT Nachricht vorgesehen. Wird eine Erweiterung an den Controller angesteckt oder entfernt, muss ein bestimmter Output Report an die Wii Remote gesendet werden, dass diese weiterhin Input Reports sendet[Wiib2013].

Die Tabelle 4-2 zeigt die verschiedenen Reports der Wii Remote. Jeder Report beginnt mit seiner ID, gefolgt von der in der Spalte Size angegebenen Anzahl an Bytes. Bei Output Reports gibt Bit 0 des ersten Bytes nach der ID immer an, ob die Vibration ein oder ausgeschaltet ist[Wiib2013]. Nachfolgend werden nicht alle Reports erklärt, sondern nur die grundlegendsten in kurzer Form.

Tabelle 4-2: Reports der Wii Remote mit ID, Größe und Funktion[Wiib2013]

I/O	ID(s)	Size	Function
O	0x10	1	Unkown
O	0x11	1	Player LEDs
O	0x12	2	Data Reporting Mode
O	0x13	1	IR Camera Enable
O	0x14	1	Speaker Enable
O	0x15	1	Status Information Request
O	0x16	21	Write Memory and Registers
O	0x17	6	Read Memory and Registers
O	0x18	21	Speaker Data
O	0x19	1	Speaker Mute
O	0x1a	1	IR Camera Enable 2
I	0x20	6	Status Information
I	0x21	21	Read Memory and Registers Data
I	0x22	4	Acknowledge output report, return function result
I	0x30 – 0x3f	2 - 21	Data reports

Die Reports mit den IDs 0x30 bis 0x3F sind die Input Reports, welche die Gamecontroller Eingaben übertragen[Wiib2013]. Die Core Buttons, also die elf Tasten, die auf der Wii Remote vorhanden sind, werden immer mit zwei Bytes angegeben[Wiib2013]. Die Core Buttons sind mit Ausnahme von 0x3D in jedem 0x3 Report an zweiter Stelle vorhanden[Wiib2013]. Die Daten der Beschleunigungssensoren bestehen aus drei Bytes und vier ungenutzten Bits des ersten Bytes eines Reports[Wiib2013]. Die Ausnahme sind die Reports 0x3E und 0x3F dort werden die Daten des Beschleunigungssensors in einem Interlaced Verfahren übertragen[Wiib2013]. Der Infrarot Sensor unterstützt verschiedene Modi und benötigt je nach Modus 10, 12 oder 36 Byte[Wiib2013]. Für die Daten der Erweiterungen werden zwischen 6 und 21 Bytes benutzt[Wiib2013]. Die Größe der Daten von verschiedenen Erweiterungen ist unterschiedlich, sodass nicht genutzte

Bytes mit Nullen gefüllt werden[Wiib2013]. Die Reports übertragen nicht immer Daten von allen Eingabequellen, sodass sich ihre Anzahl aus den verschiedenen Kombinationen und Längen erklärt.

Mit dem Report 0x12 lässt sich der Report Mode verändern. Es wird bestimmt mit welchem Input Report die Wii Remote ihre Eingaben übertragen soll. Zudem kann man angeben, ob Input Reports in Intervallen gesendet werden oder nur wenn eine Änderung der Eingaben erfolgt[Wiib2013]. Der Standardwert des Report Modes, wenn die Wii Remote eingeschaltet wird, ist Input Report 0x30. Dieserenthält nur die Core Buttons und wird nur bei Änderung der Eingabe gesendet[Wiib2013].

Die vier LEDs lassen sich mit dem Report 0x11 steuern. Bit 4 bis Bit 7 des Bytes steuert jeweils eine LED[Wiib2013].

Ein Status Information Report mit der ID 0x20 besteht aus allgemeinen Informationen über den Status der Wii Remote. Er wird entweder per 0x15 Output Report angefordert oder jedes Mal gesendet, wenn eine Erweiterung an- oder abgesteckt wird. Sollte der Report aufgrund einer Änderung der Erweiterung gesendet werden, so sendet die Wii Remote keine weiteren Input Reports, bis der Report Mode mit dem 0x15 Report erneut gesetzt wurde. Der Report beinhaltet neben den Daten der Core Buttons auch Informationen über den aktuellen Batteriestand, den Status der LEDs, ob eine Erweiterung angesteckt ist und ob der Lautsprecher oder der Infrarotsensor eingeschaltet ist[Wiib2013].

5. Grundlegendes Konzept

Damit die DirectInput API von Windows 7 und Windows 8 einwandfrei mit der Wii Remote arbeiten kann, muss die Kommunikation zwischen Wii Remote und API abgefangen und angepasst werden, sodass diese aus Sicht der DirectInput API HID-konform ist. Um eine minimale Funktionalität zu erreichen muss mindestens der Report Discriptor so angepasst werden, dass dieser dem Standard Input Report 0x30 der Wii Remote entspricht. Damit wären die Core Buttons der Wii Remote verfügbar, bis eine Erweiterung an- oder abgesteckt wird. Für eine optimale Funktionalität des Gamecontrollers sollten jedoch alle Eigenschaften der Wii Remote beachtet werden. So muss der Report Mode jedes Mal gesetzt werden, wenn sich die Erweiterung ändert. Um den Nunchuk und den Classic Controller zu unterstützen müssen die Input Reports entsprechend verarbeitet werden und in einen allgemeinen Input Report umgewandelt werden, welcher alle möglichen Inputs vereint. Mit der Verarbeitung der Report lassen sich so auch die Beschleunigungssensoren umrechnen, damit diese die Rotation der Wii Remote angeben oder Schläge als Eingabe registrieren. Die LEDs sollten natürlich auch entsprechend gesetzt werden.

Da DirectInput Gamecontrollern keine Spielernummer zugewiesen werden, können die LEDs ihren ursprünglichen Zweck, das Anzeigen der Spielernummer, nicht erfüllen. Stattdessen sollen sie den Batteriestatus wiedergeben, sodass bei vollen Batterien alle vier LEDs leuchten und je leerer die Batterie wird desto Weniger.

Ist kein Nunchuk oder Classic Controller an der Wii Remote angeschlossen, so soll sie, wie in Abbildung 5-1 rechts oben gezeigt, Horizontal gehalten werden, sodass die primär die Tasten 1, 2, A, B und das Steuerkreuz benutzt werden. Zudem soll die Drehung um die X- und Y-Achse, sowie Schläge entlang der Z-Achse erkannt werden.

Wird der Nunchuk angeschlossen, soll die Wii Remote wie in Abbildung 5-1 links unten gehalten werden. Neben allen Tasten und dem Analogstick des Nunchuks, soll auch die Rotation der Wii Remote und des Nunchuks um die X- und Y-Achse und Schläge entlang der Z-Achse erkannt werden.

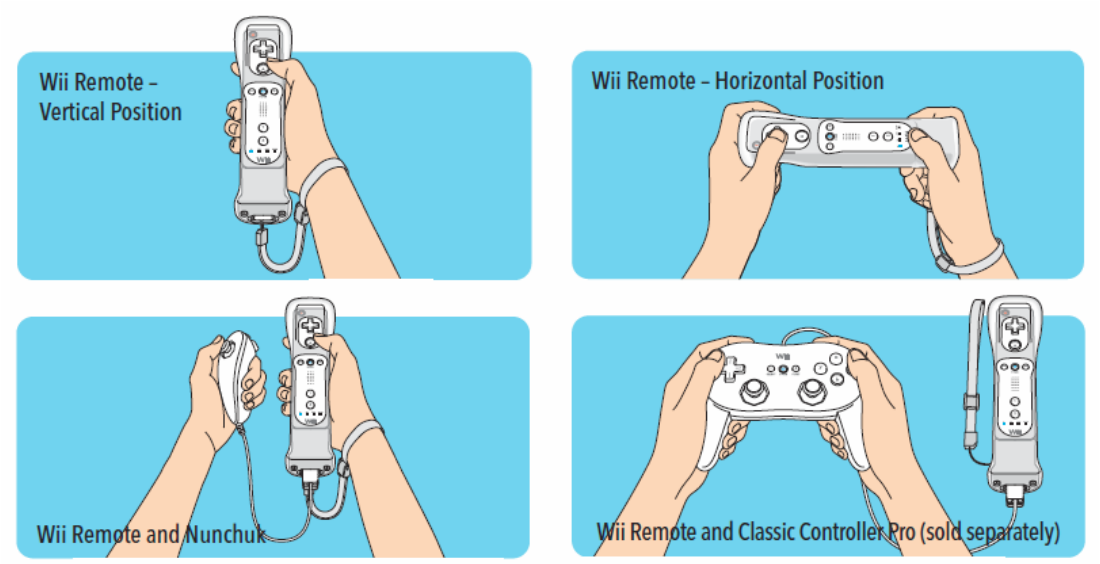

Abbildung 5-1: Die verschiedenen Arten eine Wii Remote mit oder ohne Erweiterung zu halten [Nint2009, Seite 28]

Ein Classic Controller soll wie vorgesehen mit beiden Händen gehalten werden, siehe Abbildung 5-1. Es sollen alle auf ihm vorhandenen Tasten und Analosticks benutzt werden. Da die Wii Remote nicht in der Hand gehalten wird und somit auch nicht benutzt wird, sollen nur die Eingaben auf dem Classic Controller erkannt werden.

Da beim Anstecken des Nunchuks die Wii Remote um 90 Grad gedreht wird, muss beachtet werden, dass sich damit auch die Ausrichtung des Steuerkreuzes ändert. So sollen sich die Eingaben des Steuerkreuzes ebenfalls um 90 Grad drehen. Außerdem sollen die X- und Y-Achsen, welche die Rotation darstellen, getauscht werden. Dies soll sicherstellen, dass auch während des Spielens der Nunchuk angeschlossen werden kann, ohne die Steuerung anzupassen.

Zusammenfassend lässt sich sagen, dass der vom Treiber dargestellte Gamecontroller 15 Tasten, zwei Analogsticks und vier Achsen besitzt.

6. Grundlagen der Treiberentwicklung

6.1 User Mode und Kernel Mode

Code kann unter Windows in zwei verschiedenen Modi ausgeführt werden, einmal im User Mode und im Kernel Mode[MSDN2013d]. Anwendungen werden unter Windows immer im User Mode ausgeführt, während Teile des Betriebssystems im Kernel Mode ausgeführt werden[MSDN2013d]. Treiber werden häufig im Kernel Mode ausgeführt, es gibt jedoch auch welche, die im User Mode ausgeführt werden[MSDN2013d]. Der Hauptunterschied besteht im Zugriffsrecht auf den Speicher. Anwendungen werden in Prozessen ausgeführt, diese laufen in einem virtuellen Adressraum, haben nur Zugriff auf ihren zugewiesenen Speicher und sind somit von anderen Prozessen und dem Betriebssystem isoliert[MSDN2013d]. Reagiert ein Prozess nicht mehr oder hat eine andere Art von Fehler, wird lediglich der Prozess beendet und seine Ressourcen wiederfreigegeben[MSDN2013d]. Das Betriebssystem und andere Prozesse laufen dabei ohne Einschränkung weiter[MSDN2013d]. Code, welcher im Kernel Mode ausgeführt wird, teilt sich dagegen mit anderem im Kernel Mode ausgeführten Code den Adressraum[MSDN2013d]. Ein Kernel Mode Treiber kann somit aus Versehen Daten eines anderen Kernel Mode Treibers oder des Betriebssystems verändern und so einen Absturz hervorrufen[MSDN2013d]. Tritt ein Fehler im Kernel Mode auf, wird das komplette System angehalten und ein Blue Screen angezeigt.

6.2 Driver Stack

Jedes Gerät unter Windows wird als Device Node im Plug and Play-, kurz PnP-, Baum dargestellt, siehe Abbildung 6-1 für einen Beispielbaum[MSDN2013e]. Solch eine Device Node kann entweder das physikalische Gerät oder eine Funktion des Gerätes repräsentieren, sodass ein physikalisches Gerät auch durch mehrere Knoten repräsentiert wird[MSDN2013e]. Eine Device Node kann aber auch Software repräsentieren ohne eine Verbindung zu einem physikalischen Gerät zu besitzen[MSDN2013e]. Die Abbildung 6-1 zeigt auch die Verbindung der einzelnen Knoten untereinander. Sind an einem Gerät weitere Geräte angeschlossen, so entsteht eine Eltern-Kind-Beziehung zwischen den Device Nodes. Jede Device Node

besteht im Inneren aus Listen von Device Objects, jedes dieser Device Objects ist wiederrum mit einem Treiber verknüpft[MSDN2013e]. Sucht man sich einen beliebigen Device Node aus dem Baum aus und folgt der Verknüpfung mit den jeweiligen Elternknoten zur Wurzel erhält man eine Liste von Device Objects und deren Treibern. Diese Liste nennt man Driver Stack und kann wie in Abbildung 6-2 dargestellt werden[MSDN2013f].

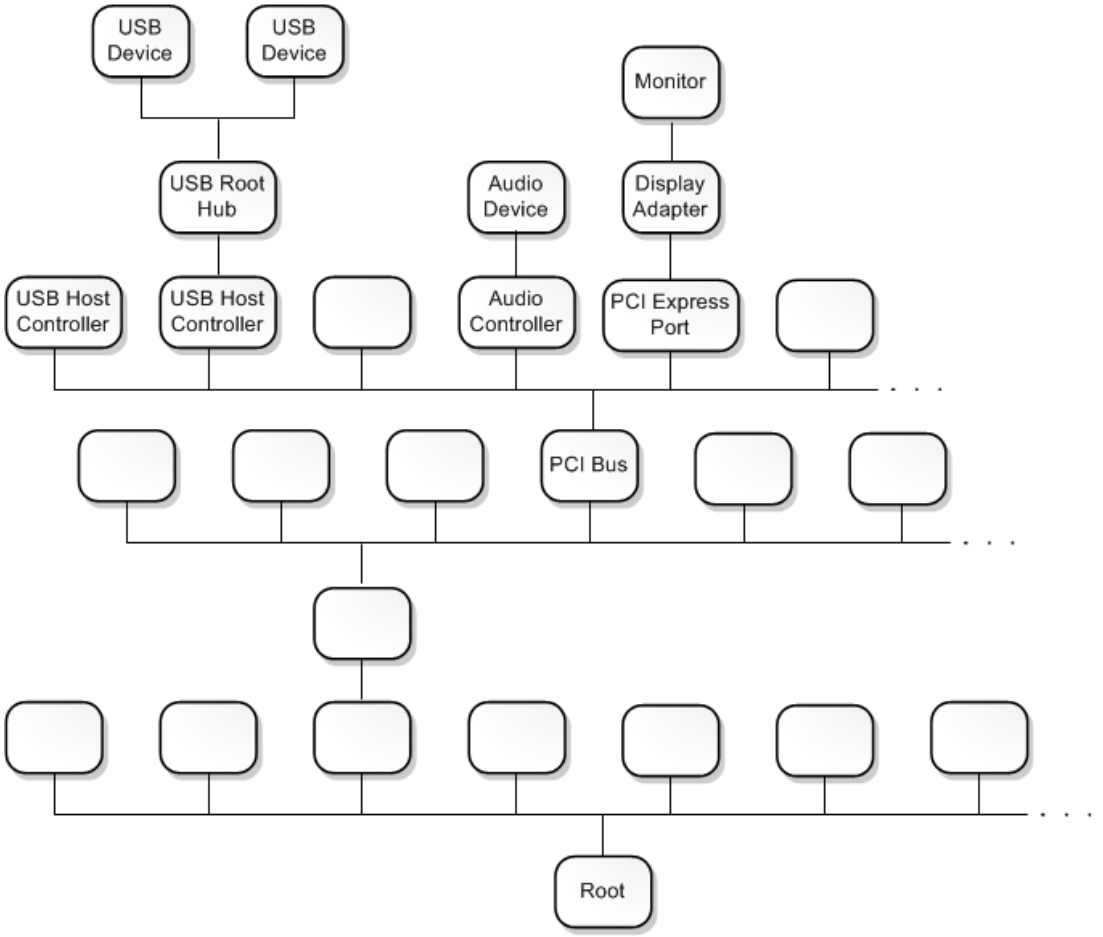

Abbildung 6-1: Beispiel eines PnP-Baum[MSDN2013e]

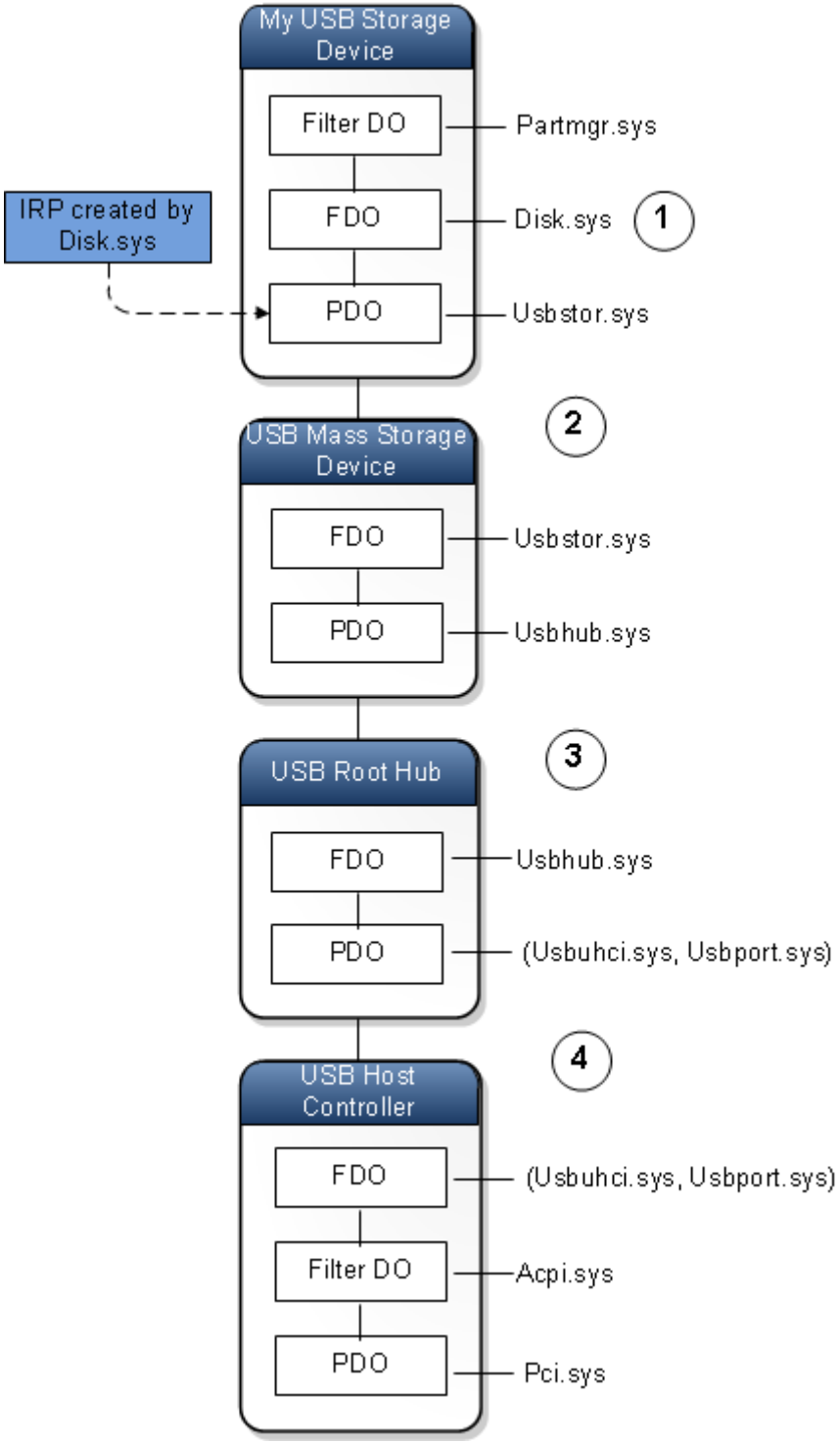

Abbildung 6-2: Beispiel eines Driver Stacks[MSDN2013f]

Es gibt verschiedene Arten von Device Objects, welche verschiedene Aufgaben haben. Eine Device Node besteht immer aus einem Function Device Objects, kurz FDO, und der damit verbundene Treiber ist der für diesen Knoten hauptverantwortliche Treiber[MSDN2013e]. Sollten Geräte von diesem Knoten aus gehen, erzeugt der Treiber für jedes Kindgerät jeweils ein Physical Device Object, kurz PDO[MSDN2013e]. Das PDO ist somit die Verbindung zwischen zwei Device Nodes. Zu solch einem PDO wird dann der entsprechende Function Driver geladen und ein FDO erzeugt[MSDN2013e]. Zusätzlich werden noch die Filter Driver geladen und deren Filter Device Objects erzeugt[MSDN2013e]. Filter Treiber sind optionale untergeordnete Hilfstreiber welche die Kommunikation zwischen den Device Objects verändern und anpassen können[MSDN2013e]. Es kann mehr als ein Filter Device Object pro Device Node existieren[MSDN2013e]. Sie sitzen entweder über oder unter dem FDO und heißen dann entsprechen Lower Filter Driver oder Upper Filter Driver[MSDN2013e].

Ein Function Driver kann auch aus zwei Treibern bestehen, dies wird dann als Driver Pair bezeichnet[MSDN2013g]. Einer der Treiber ist für allgemeine Aufgaben zuständig, welche geräteübergreifend sind[MSDN2013g]. Der Andere ist für spezielle Aufgaben zuständig, die das Gerät betreffen. Dieser wird dann als Miniport driver oder Minidriver bezeichnet[MSDN2013g].

Soll mit einem Gerät interagiert werden, also zum Beispiel von diesem gelesen werden, wird der Device Node ein I/O Request Packet, kurz IRP, gesendet[MSDN2013f]. Dieses wird dann nacheinander von denen im Driver Stack vorhandenen Treibern verarbeitet und weiter nach unten gereicht, bis es entsprechend abgearbeitet wurde. Danach wird es wieder nach oben gereicht, bis es an der anfänglichen Device Node ankommt. In Abbildung 6-2 wird zum Beispiel von Disk.sys ein IRP erzeugt und an das PDO weitergereicht, siehe Nummer 1. Ein IRP ist jedoch nicht im Besitzt des Device Objects, sondern im Besitz des verknüpften Treiber[MSDN2013f]. Da PDO und FDO der nächsten Device Node mit dem gleichen Treiber verknüpft sind, siehe Nummer 2 in Abbildung 6-2, wurde damit das IRP von der oberen Device Node in die Untere transportiert[MSDN2013f].

Dort wird es vom Treiber weiterverarbeitet und wenn nötig mit dem gleichen Verfahren weitergereicht, siehe Nummer 3 und Nummer 4 in Abbildung 6-2. In der untersten Device Node wird das IRP vom FDO an das Filter Device Object weitergereicht und von diesem erst nach der Verarbeitung and das PDO weitergereicht.

6.3 Treiber und Geräte Installation

Bei der Installation eines Treibers, muss neben der Binär Datei des Treibers auch eine Informationsdatei, kurz INF Datei, vorhanden sein[MSDN2013h]. In dieser sind alle Information enthalten, die zur Installation benötigt werden[MSDN2013h]. Sie enthält kein Installationskript, sondern ist eine Resource für den Microsoft Win32 Installer[MSDN2013h]. Neben dem Treibernamen enthält die Datei auch Informationen über die Art des Treibers, für welche Geräte der Treiber ist und ob Registry Werte hinzuzufügen sind[MSDN2013h].

Zur Identifikation welcher Treiber für welches Gerät geladen werden soll benutzt Windows Hardware IDs[MSDN2013i]. Eine Hardware ID ist eine vom Hersteller vorgegebene Zeichenkette und besteht meistens aus einer Vendor ID und einer Product ID[MSDN2013i]. Ein Gerät hat zwei Listen, eine Hardware ID Liste und eine Compatible ID Liste. Beide Listen sind der Genauigkeit abnehmend sortiert[MSDN2013i]. Während die Hardware ID Liste Einträge enthält, welche dem Gerät entsprechen, enthält die Compatible ID Liste unter anderem generische Einträge[MSDN2013j].

In den INF Dateien werden die Geräte, für die der Treiber geladen werden, mit ihren Hardware IDs angegeben[MSDN2013j]. Wird nun ein neues Gerät angeschlossen, wird ein neues PDO erzeugt. Außerdem werden die Hardware IDs und Compatible IDs des neuen Gerätes ermittelt[MSDN2013j]. Der PnP Manager sucht nun anhand der IDs nach der INF Datei mit der passendsten HardwareID und lädt den entsprechenden Treiber für das Gerät[MSDN2013j].

6.4 Windows Driver Model

Das Windows Driver Model, kurz WDM, wurde mit Windows 2000 eingeführt[MSDN2013k]. Es ermöglicht das Entwickeln von Source-Code kompatiblen Treibern für alle Windows Betriebssysteme[MSDN2013k]. Es unterscheidet zwischen Bus Driver, Function Driver und Filter Driver[MSDN2013k]. Während ein Function Driver ein einzelnes Gerät steuert und Filter Driver optionale Zusatztreiber sind, steuert ein Bus Driver ein I/O Bus Device, an welchem weitere Geräte angeschlossen werden können. Ein WDM Treiber muss immer Plug and Play und Power Management unterstützen und wird in der Programmiersprache C geschrieben[MSDN2013k].

Plug and Play, kurz PnP, bedeutet, dass ein Betriebssystem Änderungen an der Hardware selbstständig erkennt und diese selbstständig oder mit nur wenig Hilfe des Benutzers konfigurieren und in Betrieb nehmen kann[MSDN2013l]. Dies ermöglicht Benutzern Geräte und Komponenten anzuschließen und zu entfernen ohne genaue Kenntnis von Computer Hardware und deren Konfiguration zu haben[MSDN2013l]. PnP muss sowohl von der Hardware, dem Betriebssystem, als auch von den Treibern unterstütz werden[MSDN2013l]. Der PnP Manager kann so den passenden Treiber für eine Hardware erkennen und laden[MSDN2013l].

Power Management beschreibt unter anderem verschiedene Power States die ein Gerät haben kann, im Sinne der Energieverwaltung und Energiesparmodi[MSDN2013m]. Power States werden mit D0, D1, D2 und D3 bezeichnet[MSDN2013m]. D0 beschreibt dabei den höchsten Power State den ein Gerät haben kann und bedeutet dass das Gerät voll funktionsfähig ist[MSDN2013m]. D1, D2 und D3 sind Energiesparmodi in verschiedenen Abstufungen. Die Abstufungen und welche Modi ein Gerät unterstützt ist Geräte abhängig[MSDN2013m]. Viele Geräte nutzen nur die Power States D0 und D3, welche unterstützt werden müssen[MSDN2013m].

Für jeden installierten und geladenen WDM Treiber wird ein Driver Object erzeugt[MSDN2013n]. Dieses Driver Object wird dem Treiber in seiner DriverEntry Routine übergeben[MSDN2013n]. Der Treiber muss nun in das Driver Object seine weiteren Standardroutinen mittels Funktionszeigern eintragen, dass diese, wenn benötigt, aufgerufen werden können[MSDN2013n]. Ein WDM Treiber muss alle Standardroutinen implementieren, unabhängig ob diese Geräte spezifisch sind oder Standardaufgaben übernehmen.

Die Aufgabe eines IRPs wird mit Major Function Codes, mit dem Prefix IRP_MJ_XXX gekennzeichnet, und wenn benötigt Minor Function Codes, mit dem Prefix IRP_MM_XX gekennzeichnet, angegeben[MSDN2013o]. Ein WDM Treiber muss für jeden dieser Major Function Codes eine Routine zur Verarbeitung bereit stellen[MSDN2013n]. Diese Codes sind vordefiniert und werden innerhalb eines IRP in I/O Stack Location Structures vermerkt[MSDN2013o]. Für jeden Treiber im Driver Stack wird ein I/O Stack Location Structure erzeugt und mittels Zeigern mit dem Device Object verknüpft[MSDN2013o]. Wird ein IRP von einem Treiber an den nächsten weitergereicht, kann über die I/O Stack Location das Device Object erlangt werden.

6.5 Windows Driver Framework

Das Windows Driver Framework, kurz WDF, ist ein Framework auf Basis von WDM. Während das WDM sehr eng mit dem Betriebssystem arbeitet, stellt das WDF viele Objekte und Funktionen zu Verfügung, welche die direkte Interaktion mit dem Betriebssystem übernehmen[MSDN2013p]. Das WDF besteht aus dem Kernel Mode Driver Framework, kurz KMDF, und dem User Mode Driver Framework, kurz UMDF. Während mit dem UMDF User Mode Treiber in C++ geschrieben werden können, werden Kernel Mode Treiber weiterhin in C geschrieben. KMDF ist ein Objekt basierendes und Event gesteuertes Framework[MSDN2013p]. Es soll die benötigte Anzahl an Codezeilen reduzieren und die Struktur eines Treibers vereinfachen und abstrahieren. So liefert das KMDF viele Standard Implementierungen von Funktionen und Routinen, sodass nur die Geräte spezifischen Routinen implementiert werden müssen[MSDN2013p].

Ein KMDF Treiber muss eine DriverEntry implementieren[MSDN2013q]. In der DriverEntry Routine wird ein Driver Object erzeugt und die EvtDriverDeviceAdd Routine als Callback gesetzt[MSDN2013q]. In der EvtDriverDeviceAdd Routine werden die für das Gerät benötigten Callbacks gesetzt[MSDN2013p]. Es gibt unteranderem verschiedene Callbacks für PnP und das Power Management[MSDN2013p]. IRPs werden in Request Objekte gepackt, um deren Handhabung zu vereinfachen. I/O Queue Objekte werden benutzt, um eingehende Request Objekte zu empfangen[MSDN2013q]. Sie können auch benutzt werden um treiberintern Request Objekte weiterzureichen und zu speichern. In der EvtDriverDeviceAdd Routine muss mindestens eine I/O Queue erzeugt werden, die alle eingehende Request Objekte für das Gerät von anderen Treibern oder dem Betriebssystem entgegennimmt[MSDN2013q]. Per Callback wird der Treiber benachrichtigt, wenn ein Request Objekt empfangen wurde[MSDN2013q]. KMDF liefert viele weitere Objekte die allgemeine Funktionen implementieren und abstrahieren, so wie zum Beispiel ein Memory Objekt oder ein Timer Objekt[MSDN2013p]. Jeder Instanz eines KMDF Objektes kann ein oder mehrere Object Context Spaces beigefügt werden[MSDN2013r]. Ein Object Context Space ist ein Speicherblock mit variabler Größe, welcher mit der Instance des KDMF Objektes verknüpft wird[MSDN2013r]. Man kann so eigene Variablen und Daten einem KMDF Objekt hinzufügen[MSDN2013r]. Dies wird zum Beispiel dazu genutzt die I/O Queue eines Gerätes in dessen Device Object Context zu speichern, um so in anderen Callbacks wieder auf die I/O Queue zugreifen zu können.

6.6 Windows Driver Kit

Das Windows Driver Kit, kurz WDK, bezeichnet das Packet welches die nötigen Mittel zu Treiber Entwicklung enthält. Neben den Headern und Libraries des WDM und WDF, enthält es auch alle benötigten Tools und Templates. Das WDK integriert sich in die Microsoft Visual Studio Installation und bietet somit seine meisten Funktionen direkt in Visual Studio an.

Beim Debuggen von Kernel Mode Treibern wird empfohlen mit zwei Windows Computern zu arbeiten[MSDN2013s]. Auf dem Target Computer, auch Test Computer genannt, wird der Treiber installiert und ausgeführt[MSDN2013s]. Der Host Computer führt dabei den Debugger aus und kann den Debug Prozess steuern[MSDN2013s]. Der Grund ist, da bei einem Fehler des Treibers oder dem Anhalten des Treibers durch einen Breakpoint das gesamte System des Test Computers betroffen ist. Bevor ein Test Computer zum Debuggen benutzt werden kann, muss dieser konfiguriert werden[MSDN2013s]. Mit Visual Studio lässt sich diese Konfiguration automatisch durchführen und wird Provisioning a Test Computer genannt[MSDN2013t]. Für das Provisioning wird eine LAN-Verbindung benötigt, ab Windows 8 kann auch über diese LAN-Verbindung das Remote Debugging ausgeführt werden[MSDN2013t]. Bei früheren Windows Versionen wird für das Debuggen eine USB-, Firewire- oder Serielleverbindung mittels spezieller Debug Kabel benötigt[MSDN2013t]. Nach dem Provisioning kann Visual Studio den kompilierten Treiber automatisch auf dem Test Computer installieren und den Debugger verbinden[MSDN2013u].

Während ein Treiber Projekt in Visual Studio nur die Binary des Treibers erzeugt, benötigt man noch ein Driver Package Projekt, welches alle benötigten Dateien, wie unter andrem Treiber Binaries, INF-Datei und Zertifikat, als Treiber Paket zusammenbringt[MSDN2013u]. Dieses Treiber Packet wird wiederrum benutzt, um den Treiber auf dem Test Computer oder einem anderen Windows Computer zu installieren[MSDN2013u].

Um einen Treiber zu veröffentliche muss dieser signiert werden[MSDN2013v]. Dazu wird ein Zertifikat von einer vertrauenswürdigen Zertifizierungsstelle benötigt[MSDN2013v]. Zum Testen wird automatisch ein Testzertifikat erstellt und zum signieren benutzt[MSDN2013v].

7. Ansätze

Auf Basis der Grundlagen und den per Geräte-Manager verfügbaren Daten wurde der Driver Stack der Wii Remote so weit wie möglich rekonstruiert. Abbildung 7-1 zeigt wie der Driver Stack in etwa aussieht. Er ist nicht vollständig da nicht alle Informationen einsehbar sind.

Es wurden vier Ansätze ausgearbeitet, was für eine Art von Treiber wo in den Driver Stack integriert werden könnte. Diese wurden dem Aufwand zunehmend nach einander kurz getestet, ob mit der Wii Remote kommuniziert und der Eigenschaften Test Dialog manipuliert werden kann.

7.1 Ansatz 1

Der erste Ansatz war einen UMDF HID Minidriver für den Function Driver des HID-konformen Gamecontroller zu entwickeln, siehe Nummer 1 in Abbildung 7-1. Es wurde angenommen, dass der Function Driver die Schnittstelle zur DirectInput API ist. Beim Test konnte jedoch weder mit IOCTL_HID_READ_REPORT noch mit IOCTL_HID_GET_INPUT_REPORT von der Wii Remote gelesen werden, da beide Requests mit dem Status Code STATUS_NOT_SUPPORTED beantwortet wurden.

7.2 Ansatz 2

Der zweite Ansatz war ähnlich dem Ersten. Es sollte ein UMDF Filter Treiber entwickelt werden, welcher unterhalb des Function Driver des HID-konformen Gamecontrollers sitzt, siehe Nummer 1 in Abbildung 7-1. Es sollte die Kommunikation beobachtet und abgeändert werden. Beim Weitergeben der Request wurden diese jedoch mit dem Status Code STATUS_NO_SUCH_PRIVILEGE zurückgegeben.

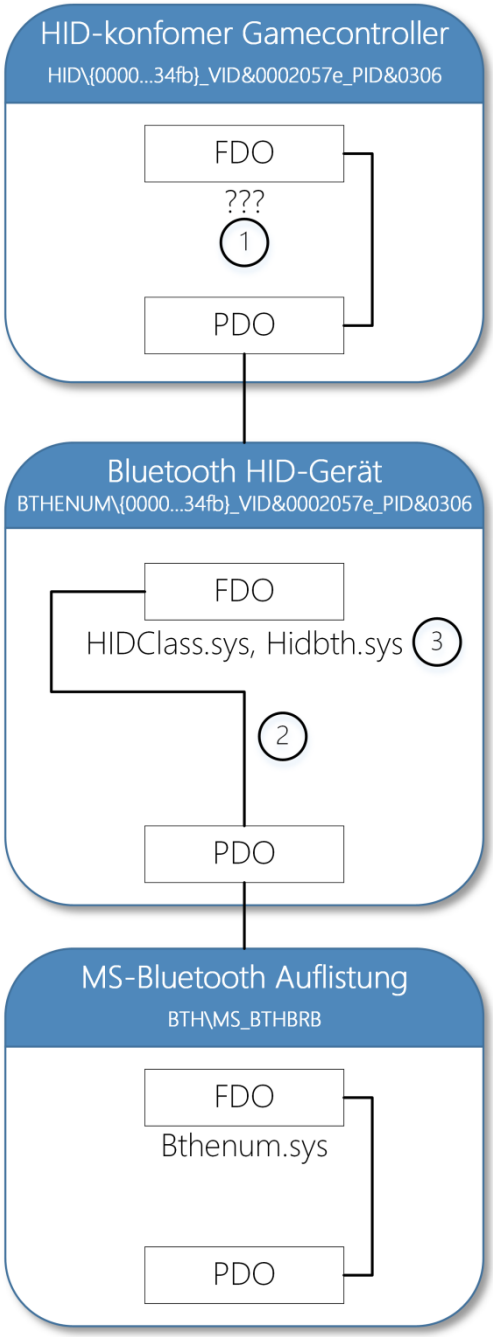

Abbildung 7-1: Teilweise rekonstruierter Driver Stack der Wii Remote

7.3 Ansatz 3

Der dritte Ansatz war ein KMDF Filter Treiber unterhalb des HIDClass Treibers des Bluetooth HID-Gerätes, siehe Nummer 2 in Abbildung 7-1. Der Treiber sollte die Kommunikation des HIDClass Treibers mit dem Bluetooth Treiber beobachten. Daraufhin sollte überprüft werden, ob es möglich sei die Kommunikation ohne großen Aufwand abzuändern. Es konnte allerdings keine erfolgreiche Installation des Treibers durchgeführt werden. Entweder wurde der Filter Treiber geladen, nicht aber das HIDClass-HidBth-Treiberpaar, oder das Treiberpaar wurde geladen, jedoch nicht der Filter Treiber.

7.4 Ansatz 4

Der letzte Ansatz war den HidBth Minitreiber, Nummer 3 in Abbildung 7-1, zu ersetzen und eine eigene Implementation zu liefern. Der Minitreiber implementiert die Geräte spezifische Kommunikation und abstrahiert diese für den HIDClass Treiber. Im Falle der Wii Remote ist dies der Bluetooth HID-Profile Treiber. Bei einem HID Gerät, welches über USB kommuniziert, würde der HIDUSB Minitreiber geladen werden. Dieser Ansatz wurde als sehr Aufwendig geschätzt da die direkte Kommunikation mit der Wii Remote per L2CAP implementiert werden muss. Der Vorteil wäre jedoch die volle Kontrolle über die Kommunikation des HIDClass Treibers mit der Wii Remote. Da es aufgrund von Überschneidungen in der Verarbeitung von einigen I/O Request zu Kompatibilitätsproblemen kommt, kann kein KMDF Minitreiber mit den HIDClass Treiber zusammen arbeiten[MSDN2013w]. Stattdessen liefert Microsoft einen Minitreiber, welcher zwischen den HIDClass Treiber und dem KMDF Minitreiber platziert wird[MSDN2013w]. Dieser MsHidKmdf gennante Treiber wird dabei zum eigentlichen Minidriver des HIDClass Treibers, während der zu entwickelnde Treiber ein Filter Treiber ist[MSDN2013w]. Der MsHidKmdf Treiber leitet sämtliche I/O Request an den Filter Treiber weiter, sodass dieser dann die Funktion eines Minitreibers übernehmen kann[MSDN2013w]. Bei Test konnte so erfolgreich eine Verbindung zur Wii Remote aufgebaut und Reports ausgetauscht werden. Außerdem konnte dem HIDClass Treiber einen eigens definierten Report Discriptor übergeben werden und auch dazu passende Reports. Dieser Ansatz ist somit für die Entwicklung eines Treibers geeignet.

8. Konzept

Auf Basis des vierten Ansatzes wurde ein Konzept für den Wii Remote Treiber erstellt.

8.1 Allgemeine Struktur

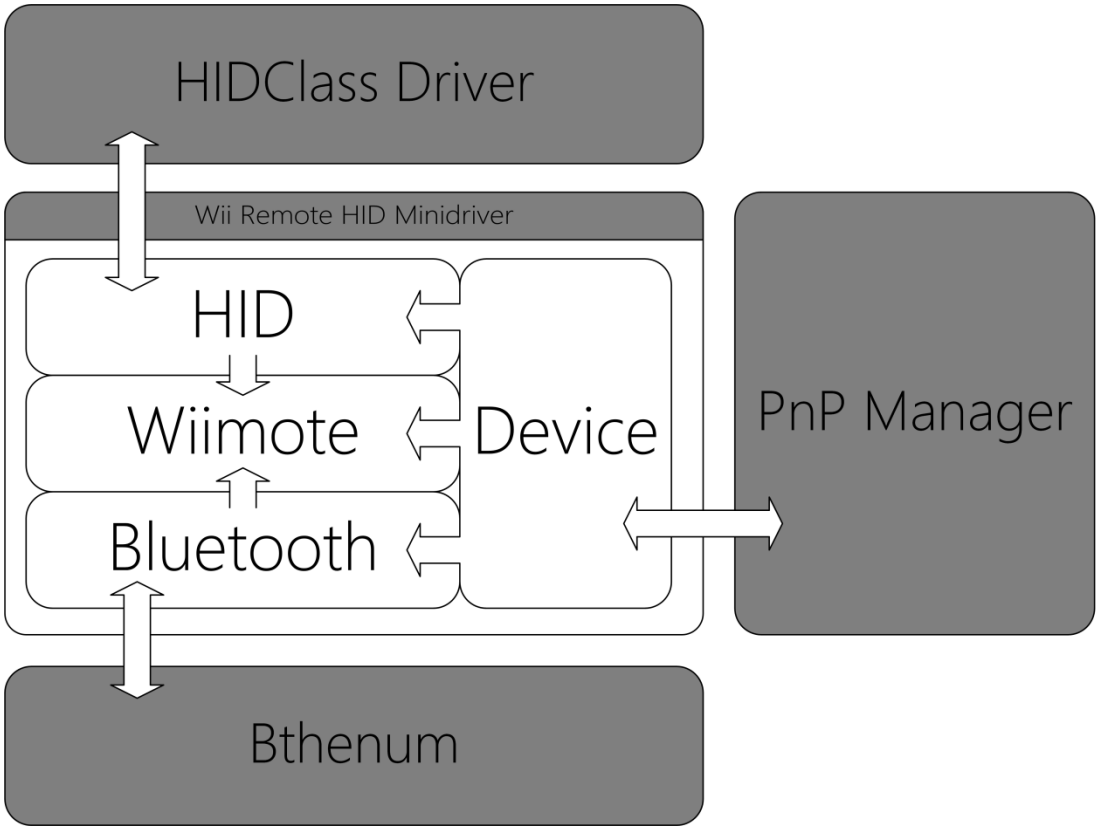

Abbildung 8-1: Allgemeine Struktur des Treibers für die Wii Remote

Wie in Abbildung 8-1 dargestellt, besteht der Treiber aus drei Ebenen. Die Unterste, Bluetooth genannt, abstrahiert die Kommunikation per Bluetooth und stellt Funktionen zur Verfügung, um Daten an die Wii Remote zu senden und Daten von ihr zu empfangen. In der Mitte ist die Wiimote genannte Ebene. Diese Ebene ist für die Wii Remote spezifische Kommunikation zuständig und speichert ihren aktuellen Status zwischen. Darüber ist die HID Ebene, welche die Kommunikation mit dem HIDClass Treiber übernimmt und den Wiimote Status in HID Reports umwandelt. Ein weiterer Teil des Treibers ist Device genannt. Dieser kommuniziert mit dem PnP Manager und beinhaltet die PnP und Power Management Callbacks. Er steuert den

PnP und Power Status des Gerätes und leitet Änderungen am Geräte Status an die anderen Ebenen weiter.

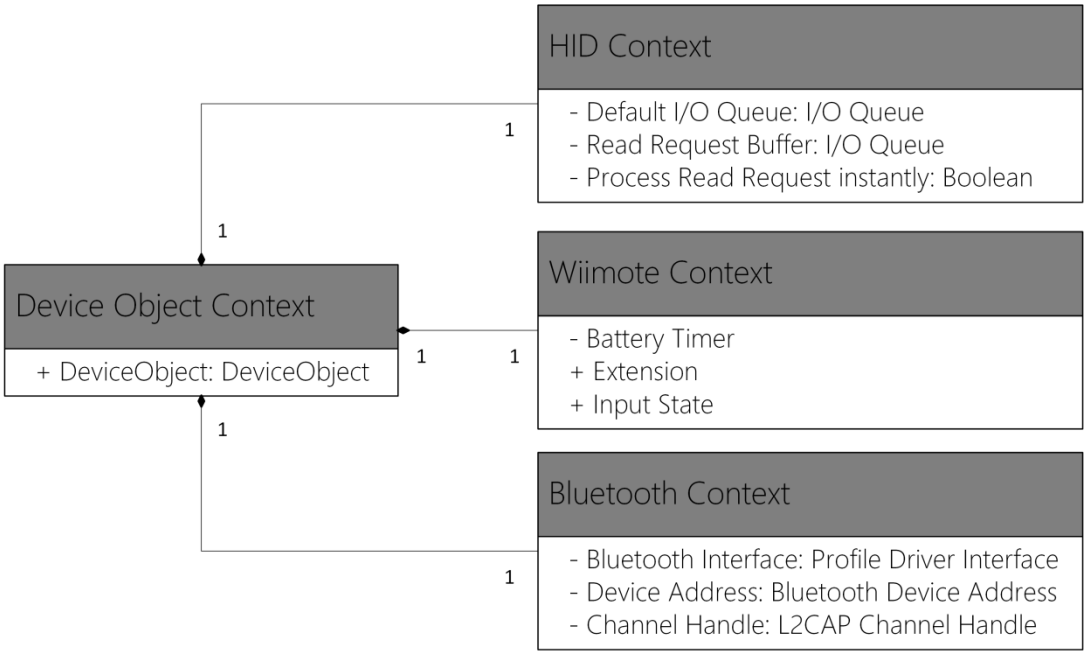

Abbildung 8-2: Struktur des Object Context Spaces für das Device Object

Das Device Object hat ein Object Context Space zum Speichern von Geräte abhängigen Daten. Jede Ebene hat einen eigenen Speicher für ihre Daten und Objekte. Diese Speicherblöcke befindet sich, wie in Abbildung 8-2 zu sehen, innerhalb des Device Object Context.

8.2 HID

Die HID-Ebene verarbeitet die eingehenden HID I/O Control Requests vom HIDClass Treiber. I/O Control Codes, kurz IOCTL, werden für die Kommunikation zwischen Treibern innerhalb eines Driver Stacks benutzt und werden von IRPs mit dem Major Function Code IRP_MJ_INTERNAL_DEVICE_CONTROL transportiert[MSDN2013x]. HIDClass benutzt zum Beispiel IOCTL_HID_GET_REPORT_DESCRIPTOR um den Report Descriptor zu erhalten oder IOCTL_HID_GET_READ_REPORT um nach einem Input Report zu fragen[MSDN2013y; MSDN2013z].

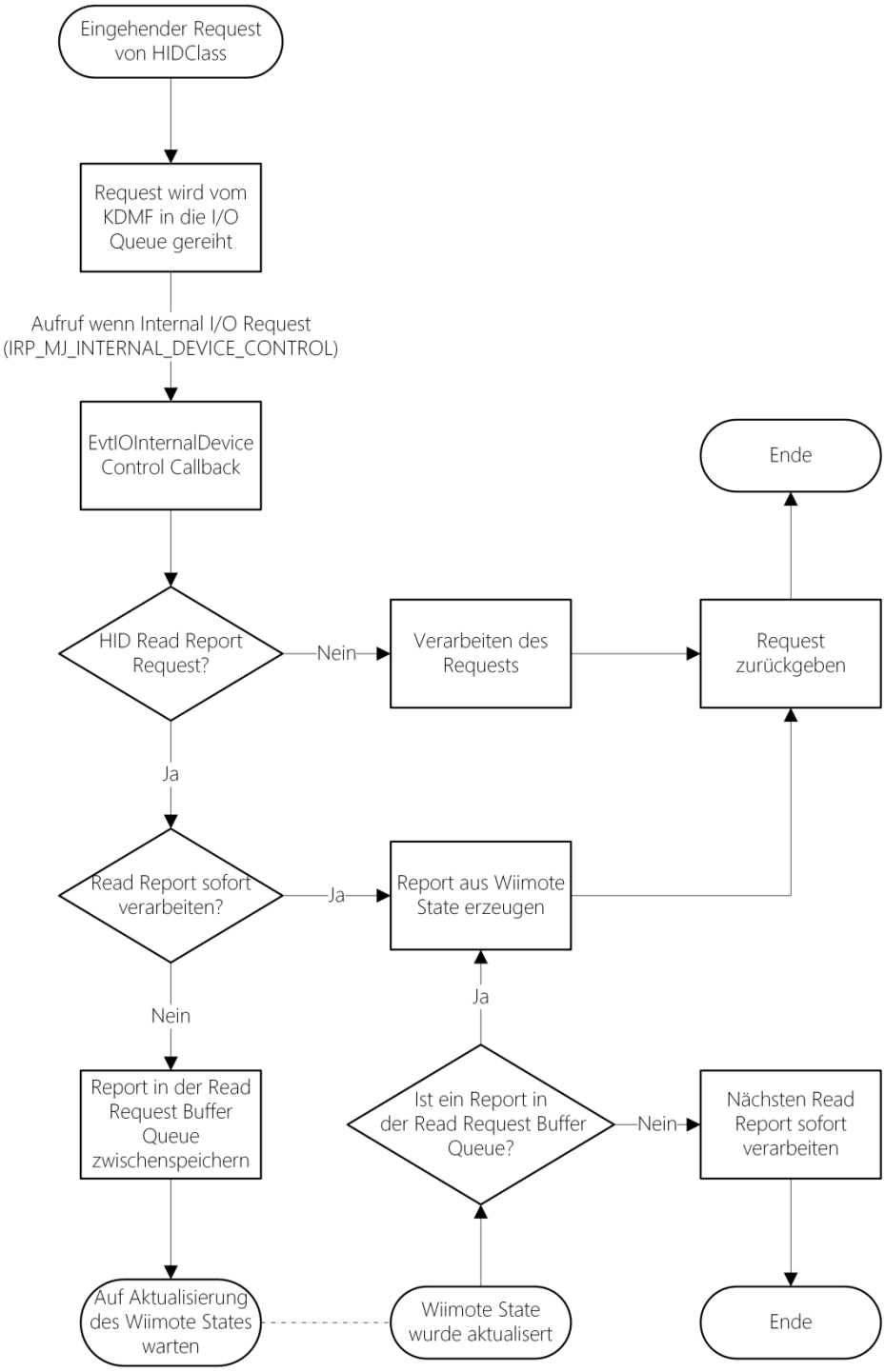

Abbildung 8-3: Ablaufdiagramm zum Verarbeiten der Requests

In eine Default I/O Queue werden eingehende Requests automatisch vom KMDF eingereiht[MSDN2013aa]. Per Callbacks lassen sich diese Request aus der Queue verarbeiten[MSDN2013ab]. Dazu bietet das I/O Queue Object verschieden Callbacks an[MSDN2013ab]. Die EvtIOInternalDeviceControl Callback wird aufgerufen, wenn der Request einen IOCTL enthält[MSDN2013ab]. Abbildung 8-3 zeigt als Ablaufdiagramm die grobe Verarbeitung eines Requests. Viele Requests vom HIDClass Treiber verlangen lediglich einfache Daten und werden direkt verarbeitet. Besondere Aufmerksamkeit ist dem IOCTL_HID_READ_REPORT, in der Abbildung als HID Read Report Request bezeichnet, zuzuwenden, da dieser einen Input Report vom Gerät lesen soll. Er wird in einer weiteren I/O Queue, der Read Request Buffer Queue, zwischengespeichert und erst weiterverarbeitet, wenn eine Änderung des Inputs vorliegt. Es gibt außerdem ein Flag für den Fall, dass sich der Input verändert, ohne dass ein HID Read Report in der Read Request Buffer Queue befindet. Um die Änderung schnellstmöglich an den HIDClass Treiber weiterzugeben, gibt deswegen das Flag an, ob ein HID Report Request sofort verarbeitet werden soll. So ist sichergestellt, dass immer alle Änderungen des Gamecontrollers an den HIDClass Treiber weitergegeben werden. Beim Weiterverarbeiten eines HID Report Requests, wird aus den Daten der Wiimote Ebene, welche den aktuellen Stand des Gamecontrollers wiederspiegeln, ein Input Report erzeugt.

Somit benötigt die HID-Ebene die in Abbildung 8-2 gezeigten drei Geräte abhängigen Variablen. Die zwei I/O Queues und den Boolean als Flag.

8.3 Bluetooth

Die Kommunikation per Bluetooth Verbindung verwaltet die Bluetooth Ebene. Sie stellt abstrahierte Funktionen zu Verfügung um Reports an die Wii Remote zu senden und einen Continuous Reader, um Reports von ihr zu lesen.

Für die Interaktion mit einem Bluetooth Gerät werden von Profile Treibern Bluetooth Request Blocks, kurz BRBs, benutzt[MSDN2013ac]. Diese werden per IOCTL_BTH_INTERNAL_SUBMIT_BRB an den Bluetooth Treiber gesendet und werden zum Beispiel dazu genutzt, um einen L2CAP Channel zu öffnen und Daten über einen L2CAP Channel zu senden[MSDN2013ac]. Abbildung 8-4 zeigt die Verwendung eines solchen BRBs. BRBs müssen vom Profile Driver Interface initialisiert werden, optional kann das Interface auch direkt initialisierte BRBs allozieren und muss diese dann auch wieder freigeben[MSDN2013ac]. Das KDMF stellt die Funktion WdfFdoQueryForInterface zur Verfügung, welche per GUID ein Interface automatisch besorgt und zur Verfügung stellt[MSDN2013ad]. Weiterhin wird noch die Bluetooth Adresse der Wii Remote zum Adressieren der BRBs benötigt. Die Bluetooth Adresse kann per IOCTL_INTERNAL_BTHENUM_GET_DEVINFO ermittelt werden.

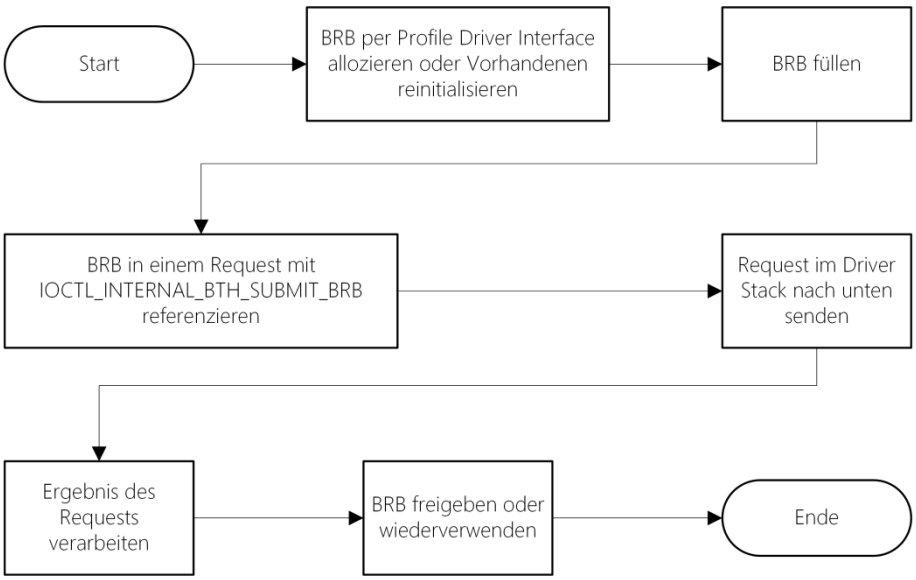

Abbildung 8-4: Zyklus eines Bluetooth Request Blocks

Um einen L2CAP-Channel zu öffnen, wird ein BRB_L2CA_OPEN_CHANNEL mit dem Profile Driver Interface erzeugt. Dieses BRB wird dann mit der Bluetooth Adresse, dem PSM des Channels und weiterer Konfiguration gefüllt. Anschließend wird es mit einem Request an den Bluetooth Treiber gesendet. Wurde es erfolgreich vom

Bluetooth Treiber verarbeitet, enthält es einen L2CAP Channel Handle. Dieser Handle wird zwischengespeichert und für spätere Lese- und Schreibaktionen über dem Channel benötigt. Zum Schluss kann der BRB freigegeben werden oder für eine andere Interaktion mit dem Bluetooth Geräts wiederverwendet werden. Zum Lesen oder Schreiben wird ein BRB_L2CA_ACL_TRANSFER benötigt. Dieses wird mit der Bluetooth Adresse und dem L2CAP Channel Handle konfiguriert. Zudem wird per Flag angegeben, ob gelesen oder geschrieben werden soll. In dem BRB wird ein vom Treiber allozierter Buffer referenziert. Dieser enthält die Schreibdaten oder wird später die gelesenen Daten enthalten. Beim Lesen wird der Request solange vom Bluetooth Treiber nicht zurückgegeben, bis endlich Daten vom Bluetooth Gerät zur Verfügung stehen oder ein Fehler auftritt.

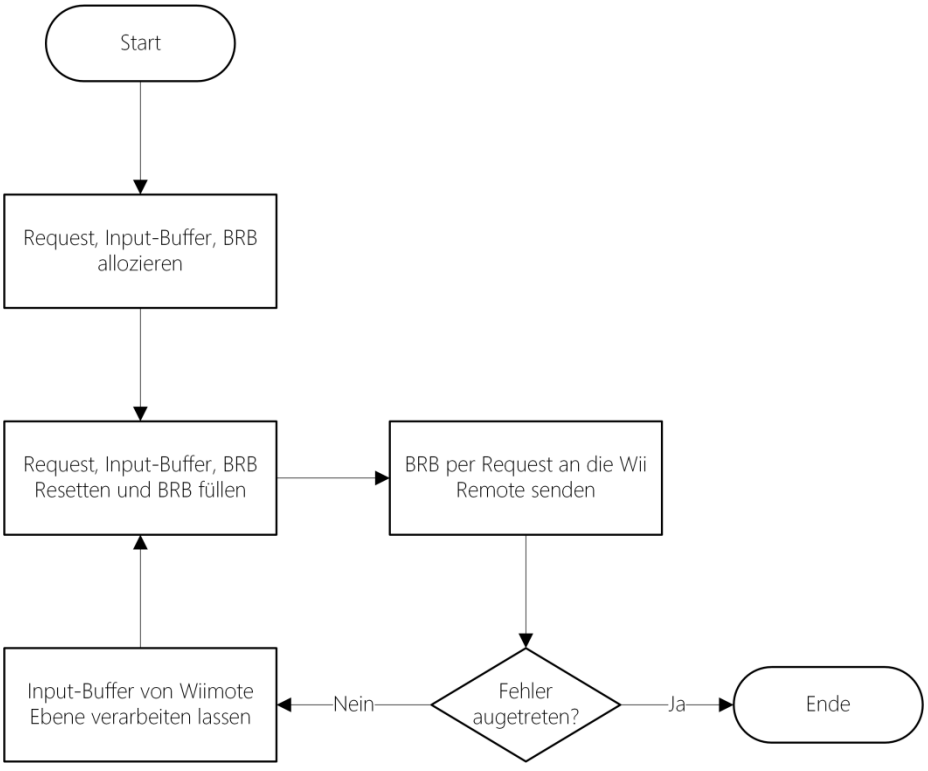

Abbildung 8-5: Funktionsweise des Continuous Readers

Die Bluetooth Ebene liefert Funktionen um einen Channel zur Kommunikation zu öffnen und durch diesen Daten an die Wii Remote zusenden. Zudem enthält die Ebene einen Continuous Reader, um ständig Input Reports von der Wii Remote zu lesen. Dieser liest ununterbrochen die Daten von der Wii Remote und stellt die der

Wiimote Ebene zum Verarbeiten zur Verfügung, siehe Abbildung 8-5. Einzig durch einen Fehler wird der Continuous Reader beendet. Ein Fehler tritt auf, wenn die Verbindung abbricht oder beendet wurde. Somit muss der Continuous Reader nicht referenziert werden und beendet sich selbst, wenn die Wii Remote nicht mehr mit dem System verbunden ist. Außerdem soll die Bluetooth-Ebene dem Device Bereich signalisieren, wenn die Wii Remote die Verbindung beendet hat, dass dieser wiederrum dem PnP-Manager signalisieren kann, dass die Wii Remote entfernt wurde. Wird die Wii Remote ausgeschaltet, aus der Bluetooth Reichweite entfernt oder gehen die Batterien während des Betriebs leer, wird vom PnP-Manager nicht erkannt, dass die Wii Remote nicht mehr Verfügbar ist. Lediglich der L2CAP Channel wird geschlossen, sodass der Profile Treiber dem PnP-Manager signalisieren muss, dass die Wii Remote entfernt wurde.

8.4 Wiimote

Die Wiimote Ebene enthält die Logik zur Steuerung der Wii Remote und speichert ihren aktuellen Status zwischen, damit die HID Ebene diesen zu Input Reports verarbeiten kann. Von der Bluetooth Ebene werden empfangene Input Reports zur Verarbeitung bereitgestellt. Diese werden wie in Abbildung 8-6 dargestellt verarbeitet. Data Reports, Report ID 0x30 bis 0x3F, werden je nach Report Mode und angesteckter Erweiterung verarbeitet und aktualisieren den Input State. Danach wird der HID Ebene signalisiert, dass der Wiimote State aktualisiert wurde, damit diese die Änderungen an den HIDClass Treiber weitergeben kann. Report ID 0x20 Status Information wird entweder gesendet, wenn sich die Erweiterung ändert oder wenn dieser Report per Output Report 0x15 Status Information Request angefordert wurde. Zunächst werden die LEDs dem aktuellen Batterie Level angepasst und der Batterie Timer wieder zurückgesetzt. Danach wird überprüft, ob sich die Erweiterung verändert hat. Hat sich die Erweiterung verändert, so muss der Report Mode entsprechend gesetzt und die Erweiterung, sofern vorhanden, initialisiert werden.

Der Batterie Timer soll jede Minute einen Status Information Report anfordern, um den Batterie Status zu überprüfen. Bei einem Intervall von einer Minute sollte die Kommunikation nicht allzu viel Bandbreite und Strom verbrauchen, jedoch dennoch Änderungen des Batterie Status rechtzeitig angeben. Wenn der Batterie Status niedrig ist, sodass nur noch eine LED leuchtet, wird der Timer abgeschaltet und nicht mehr benötigt.

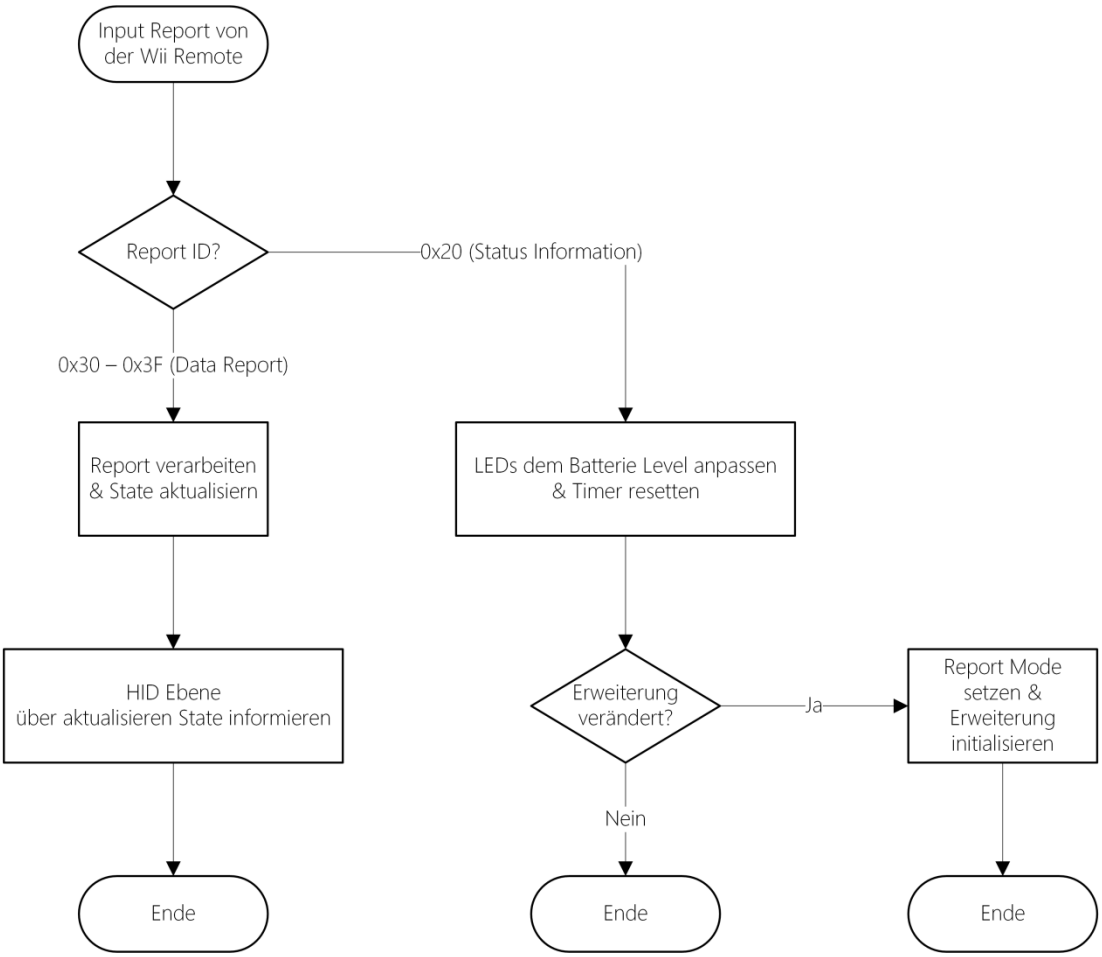

Abbildung 8-6: Verarbeitung der Input Reports von der Wii Remote

8.5 Hinzufügen einer Wii Remote

Wird ein Treiber für ein Gerät geladen, ruft der PnP-Manager die EvtDriverDeviceAdd Callback des Treibers auf[MSDN2013ae]. Abbildung 8-7 zeigt die Sequenz des Treibers bis die Wii Remote voll funktionsfähig ist. Es werden zunächst die weiteren PnP und Power Management Callbacks gesetzt, ein Framework Device Object und dessen Object Context erzeugt und die I/O Queues der HID Ebene. Die Funktion gibt dann einen NTSTATUS zurück, um dem PnP-Manager zu signalisieren, ob sie erfolgreich war oder ein Fehler auftrat[MSDN2013ae]. Danach wird die EvtDevicePrepareHardware Callback einmalig aufgerufen[MSDN2013af]. Der PnP-Manager hat bereits weitere Ressourcen für das Gerät geladen und es befindet sich in einem uninitialisierten D0 Power State[MSDN2013af]. Diese Funktion soll Informationen über das Gerät sammeln, Interfaces anderer Treiber holen und andere nur einmalige Operationen ausführen[MSDN2013af]. Im Falle der Wii Remote wird das Profile Driver Interface und die Bluetooth Adresse besorgt und gespeichert. Wie auch bei der EvtDriverDeviceAdd Funktion, wird auch wieder ein NTSTATUS zurückgegeben. Anschließend wird die EvtDeviceD0Entry Callback aufgerufen. Diese Callback wird jedes Mal aufgerufen, wenn das Gerät in den D0 Power State wechselt[MSDN2013ag]. Dies geschieht auch, wenn es zum Beispiel aus dem Ruhemodus wieder erwacht. Deshalb müssen von dieser Funktion aus alle Operationen ausgeführt werden, damit die Wii Remote voll funktionsfähig ist. Es wird zunächst von der Bluetooth Ebene der L2CAP Channel geöffnet. Da diese Operation zeitintensiv sein kann und das Betriebssystem weiter laufen muss, wird das Öffnen des L2CAP Channels asyncron ausgeführt. Ist der Channel geöffnet, so wird von der Wiimote Ebene der Continuous Reader und, sofern benötigt, der Batterie Timer gestartet. Außerdem wird der aktuelle Status der Wii Remote ausgelesen, um den Zwischenspeicher der Wiimote Ebene anzugleichen.

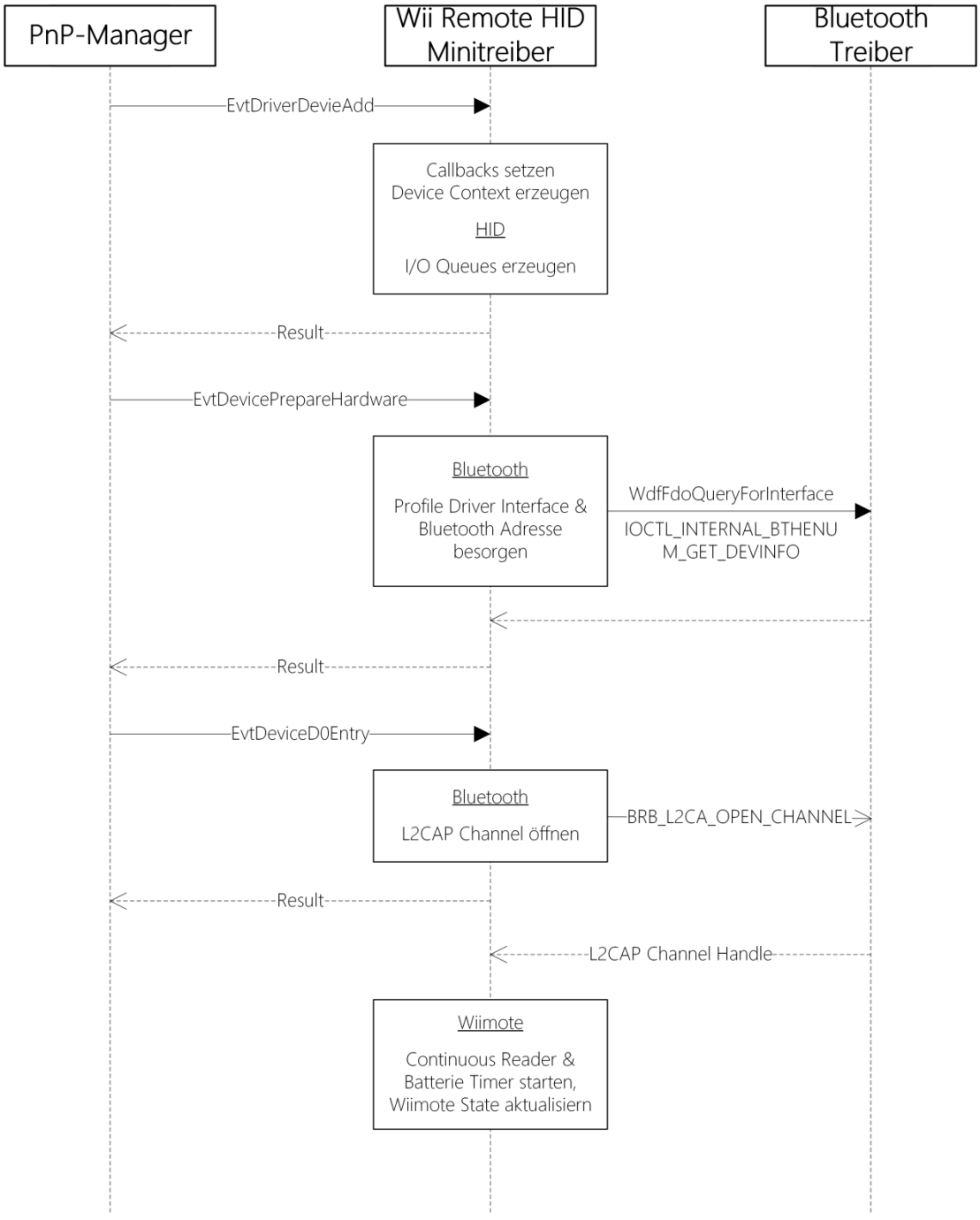

Abbildung 8-7: Sequenz beim Hinzufügen der Wii Remote

8.6 Entfernen einer Wii Remote

Die Wii Remote kann auf zwei verschiedenen Wegen entfernt werden. Entweder wird sie vom Benutzer aus dem Betriebssystem entfernt, oder die Verbindung wird von der Wii Remote aus unterbrochen. Im letzteren Fall signalisiert, wie bereits beschrieben, der Treiber dem PnP-Manager, dass die Wii Remote entfernt wurde. Unabhängig davon wie die Wii Remote entfernt wurde, wird während des Entfernen wie in Abbildung 8-8 gezeigt vorgegeangen. Zunächst wird vom PnP-Manager die Callback EvtDeviceD0Exit aufgerufen. Diese signalisiert, dass das Gerät aus dem vollfunktionsfähigen Power State in einen anderen Power State wechselt[MSDN2013ah]. Die Wiimote Ebene schickt der Wii Remote eine HID SUSPEND Nachricht, um diese auszuschalten und stoppt ihren Batterie Timer. Anschließend wird von der Bluetooth Ebene der L2CAP Channel geschloßen. Das Auschalten der Wii Remote und Schließen des Channels entfällt natürlich, wenn die Verbindung zur Wii Remote bereits abgebrochen ist. Danach wird vom PnP-Manager die EvtDeviceReleaseHardware Callback ausgeführt. In dieser Funtion sollen alle Operationen ausgeführt werden, wenn das Gerät nicht länger gebraucht wird[MSDN2013ai]. Es werden die noch vorhandenen HID Read Requests in der Read Request Buffer Queue der HID Ebene und der Timer der Wiimote Ebene gelöscht. Alle weiteren Resourcen werden, wie I/O Queues und Device Object, werden vom Framework automatisch freigegeben[MSDN2013aj].

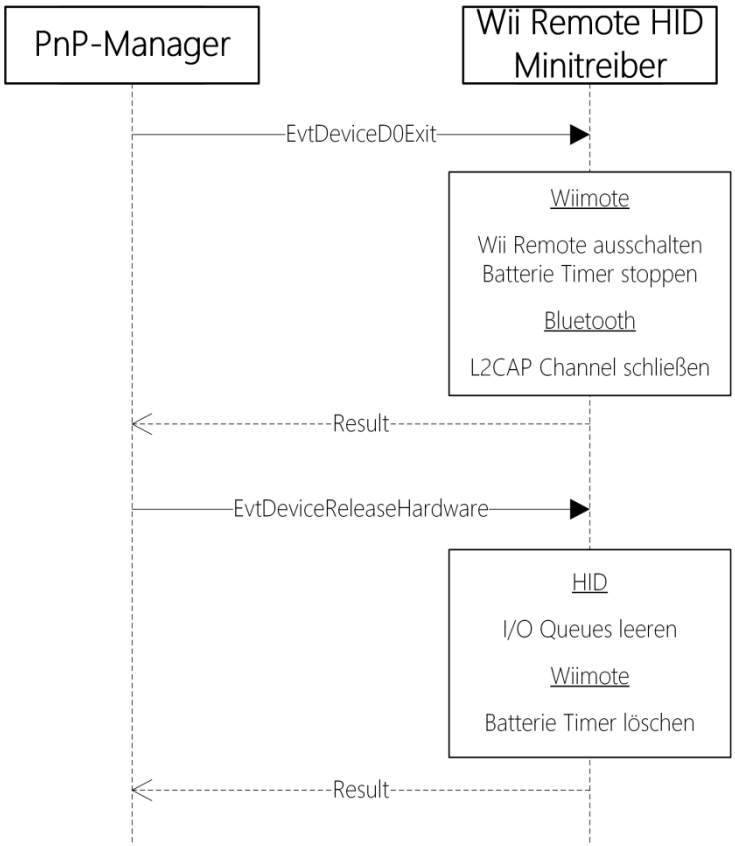

Abbildung 8-8: Sequenz beim Entfernen der Wii Remote

8.7 INF Datei

Damit der Treiber sowohl unter Windows 7, als auch unter Windows 8 installierbar ist, werden von der INF Datei des Treibers beide Betriebssysteme unterstützt. Der benötigte Hilfstreiber MsHidKmdf.sys ist seit Windows 7 im Betriebssystem integriert und muss nicht im Treiberpacket mitgeliefert werden[MSDN2013ah]. Dennoch muss in der INF Datei angegeben werden, dass dieser Treiber mit geladen werden muss. Außerdem müssen die beiden unterschiedlichen Hardware IDs der alten und neuen Wii Remote angegeben werden, damit der Treiber für alle Wii Remotes geladen wird.

9. Umsetzung

Aufgrund des Umfamgs konnte das Konzept leider nur teilweise umgesetzt werden. Die Grundzüge wurden jedoch soweit umgesetzt, um zu Beweisen, dass das technische Konzept funktioniert.

Der umgesetzte Treiber stellt der DirectInput API lediglich die Core Buttons der Wii Remote zur Verfügung. Weitere Eingabemöglichkeiten sind nicht implementiert. Auch die LEDs repräsentieren nicht den Batterie Status, sondern es werden einfach alle vier LEDs eingeschaltet. Auf die Erweiterungen wird soweit reagiert, dass die Core Buttons weiterhin funktioneren, wenn eine Erweiterung angeschlossen oder entfernt wird. Dem PnP-Manager wird vom Treiber nicht signalisiert, wenn die Wii Remote entfernt wurde und es wird nur die ältere Version der Wii Remote von der INF Datei unterstützt. Alle anderen Teile des Konzeptes wurden umgesetzt, sodass der Treiber beweisen kann, dass das Konzept funktioniert.

Die Implementation erfolgte mit dem WDK 8 und der Version 1.11 des WDFs.

10. Evaluation

Die Umsetzung wurde erfolgreich auf Windows 7 und Windows 8 installiert und getestet. Beim Test wurde der Eigenschaften Test Dialog der Wii Remote nach Treiber Installation herangezogen und überprüft, ob die Eingaben zeitnah angegeben wurden. Wie auf Abbildung 10-1 zu sehen werden die elf Tasten der Wii Remote Dialog angezeigt und auch beim Betätigen einer Taste wird dies sofort im Dialog wiedergegeben. Somit kann der Test als erfolgreich gewertet werden, was wiederrum beweist, dass das Konzept erfolgreich funktioniert. Lediglich die Handhabung und Umsetzung der verschiedenen Erweiterungen konnte nicht getestet werden.

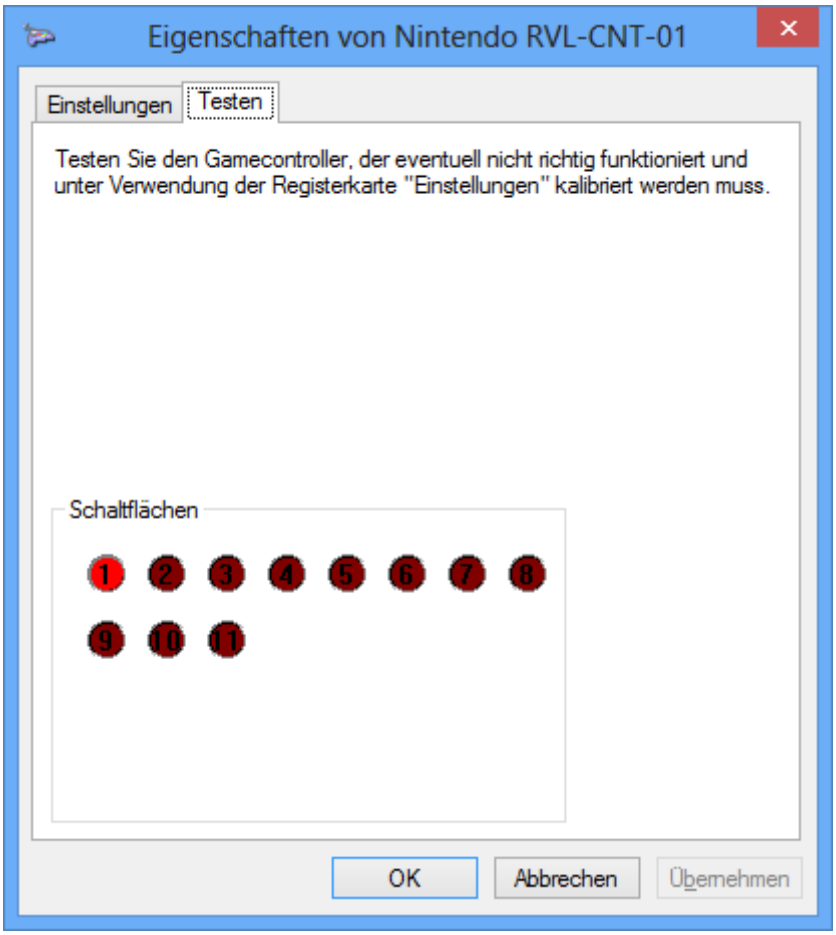

Abbildung 10-1: Eigenschaften Test Dialog der Wii Remote mit installiertem Treiber

Aufgrund der Unterstützung der DirectInput API ist die Wii Remote mit dem entwickelten Treiber für ältere Spiele einsetzbar. Neuere Spiele unterstützen zum Teil nur die neuere XInput API und sind somit dennoch nicht mit dem Treiber bedienbar.

Außerdem gestaltet sich die Installation des Treibers etwas kompliziert. Da ein Zertifikat zur Signierung und Veröffentlichung des Treibers nur kostenpflichtig verfügbar ist, muss der Treiber mit einem Test Zertifikat signiert werden. Dadurch kann der Treiber nur auf einer Windowsinstallation mit deaktivierter Prüfung der Treibersignatur ausgeführt werden. Diese Prüfung muss jedoch explizit bei jedem Start des Betriebssystems deaktiviert werden, was einfachen Anwendern zu kompliziert erscheinen könnte.

11. Bewertung & Ausblick

Mit einer kompletten Umsetzung des Konzeptes stellt die Wii Remote mit dem Treiber eine gute Alternative zu älteren Gamecontroller dar. Mit einem entsprechendem Veröffentlichungszertifikat und einer einfachen Installation sollte der Treiber auch für viele Benutzer geeignet sein, sodass durchaus Potential besteht, dass viele Spieler den Treiber nutzen werden.

Eine Idee, welche durch diese Arbeit enstand, wäre das Reverse Engineering des XUSB Treibers, bzw. die Erforschung, wie man die XInput API von der Seite eines Treibers anspricht. Die Ergebnisse könnten dann wiederrum mit dem Konzept dieser Arbeit verbunden werden, um die Wii Remote mit der XInput API kompatibel zu machen.

Außerdem soll diese Arbeit auch eine Basis für ähnliche Umsetzungen liefern. So kann man versuchen das Konzept für das Wii U Gamepad und den Wii U Pro Controller anzupassen. Besonders der Wii U Pro Controller ist durch sein ähnliches Layout als Xbox 360 Controller Ersatz geeignet.

I. Quellenverzeichnis

[BSIG2012] *Bluetooth SIG, Inc.*: HUMAN INTERFACE DEVICE PROFILE 1.1. Revision V11, 2012.

[BSIG2013a] *Bluetooth SIG, Inc.*: Bluetooth Developer Portal: Profiles Overview. https://developer.bluetooth.org/TechnologyOverview/Pages/Profiles.aspx, Abruf am 28.09.2013 22:58.

[BSIG2013b] *Bluetooth SIG, Inc.*: Bluetooth Developer Portal: Logical Link Control and Adaptation (L2CAP) Architecture. https://developer.bluetooth.org/TechnologyOverview/Pages/L2CAP.aspx, Abruf am 29.09.2013 17:30.

[Loeh2013] *Löhr, Julian*: Wii Remote am Windows PC: Analyse und Vergleich verschiedener Programme und Libraries. 2013.

[MSDN2013a] *Microsoft Corporation*: DirectInput and XUSB Devices. http://msdn.microsoft.com/en-us/library/windows/desktop/hh405052(v=vs.85).aspx, 26.07.2013, Abruf am 28.09.2013 11:46.

[MSDN2013b] *Microsoft Corporation*: XInput and DirectInput. http://msdn.microsoft.com/en-us/library/windows/desktop/ee417014(v=vs.85).aspx, 26.07.2013, Abruf am 28.09.2013 12:17.

[MSDN2013c] *Microsoft Corporation*: Introduction to HID Concepts. http://msdn.microsoft.com/en-us/library/windows/hardware/jj126202(v=vs.85).aspx, 26.07.2013, Abruf am 28.09.2013 12:58.

[MSDN2013d] *Microsoft Corporation*: User mode and kernel mode. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554836(v=vs.85).aspx, 02.09.2013, Abruf am 01.10.2013 23:41.

[MSDN2013e] *Microsoft Corporation*: Device nodes and device stacks. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554721(v=vs.85).aspx, 02.09.2013, Abruf am 02.10.2013 11:48.

[MSDN2013f] *Microsoft Corporation*: Driver stacks. http://msdn.microsoft.com/en-us/library/windows/hardware/hh439632(v=vs.85).aspx, 02.09.2013, Abruf am 02.10.2013 12:01.

[MSDN2013g] *Microsoft Corporation*: Minidrivers, Miniport drivers, and driver pairs. http://msdn.microsoft.com/en-us/library/windows/hardware/hh439643(v=vs.85).aspx, 02.09.2013, Abruf am 02.10.2013 13:15.

[MSDN2013h] *Microsoft Corporation*: Overview of INF Files. http://msdn.microsoft.com/en-us/library/windows/hardware/ff549520(v=vs.85).aspx, 05.09.2013, Abruf am 02.10.2013 16:34.

[MSDN2013i] *Microsoft Corporation*: Hardware IDs. http://msdn.microsoft.com/en-us/library/windows/hardware/ff546152(v=vs.85).aspx, 05.09.2013, Abruf am 02.10.2013 23:45.

[MSDN2013j] *Microsoft Corporation*: Step 1 The New Device is Identified. http://msdn.microsoft.com/en-us/library/windows/hardware/ff728852(v=vs.85).aspx, 05.09.2013, Abruf am 02.10.2013 23:45.

[MSDN2013k] *Microsoft Corporation*: Introduction to WDM. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548158(v=vs.85).aspx, 26.08.2013, Abruf am 03.10.2013 11:18.

[MSDN2013l] *Microsoft Corporation*: Introduction to Plug and Play. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548102(v=vs.85).aspx, 26.08.2013, Abruf am 03.10.2013 11:37.

[MSDN2013m] *Microsoft Corporation*: Device Power States. http://msdn.microsoft.com/en-us/library/windows/hardware/ff543162(v=vs.85).aspx, 26.08.2013, Abruf am 03.10.2013 11:37.

[MSDN2013n] *Microsoft Corporation*: Introduction to Driver Objects. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548034(v=vs.85).aspx, 26.08.2013, Abruf am 03.10.2013 13:10.

[MSDN2013o] *Microsoft Corporation*: I/O Stack Locations. http://msdn.microsoft.com/en-us/library/windows/hardware/ff551821(v=vs.85).aspx, 26.08.2013, Abruf am 03.10.2013 13:40.

[MSDN2013p] *Microsoft Corporation*: Differences Between WDM and KMDF. http://msdn.microsoft.com/en-us/library/windows/hardware/gg583838(v=vs.85).aspx, 05.09.2013, Abruf am 03.10.2013 14:00.

[MSDN2013q] *Microsoft Corporation*: Writing a Simple WDF Driver. http://msdn.microsoft.com/en-us/library/windows/hardware/ff553112(v=vs.85).aspx, 05.09.2013, Abruf am 04.10.2013 12:25.

[MSDN2013r] *Microsoft Corporation*: Framework Object Context Space. http://msdn.microsoft.com/en-us/library/windows/hardware/ff542873(v=vs.85).aspx, 05.09.2013, Abruf am 04.10.2013 12:30.

[MSDN2013s] *Microsoft Corporation*: Debugging a Driver. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554672(v=vs.85).aspx, 12.09.2013, Abruf am 04.10.2013 13:20.

[MSDN2013t] *Microsoft Corporation*: Configuring a Computer for Driver Deployment, Testing, and Debugging. http://msdn.microsoft.com/en-us/library/windows/hardware/hh698272(v=vs.85).aspx, 12.09.2013, Abruf am 04.10.2013 13:20.

[MSDN2013u] *Microsoft Corporation*: Creating a Driver Package. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554776(v=vs.85).aspx, 12.09.2013, Abruf am 04.10.2013 18:55.

[MSDN2013v] *Microsoft Corporation*: Signing a Driver. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554809(v=vs.85).aspx, 12.09.2013, Abruf am 04.10.2013 18:55.

[MSDN2013w] *Microsoft Corporation*: Creating KMDF HID Minidrivers. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540774(v=vs.85).aspx, 05.09.2013, Abruf am 03.10.2013 23:25.

[MSDN2013x] *Microsoft Corporation*: Introduction to I/O Control Codes. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548059(v=vs.85).aspx, 26.08.2013, Abruf am 05.10.2013 19:00.

[MSDN2013y] *Microsoft Corporation*: IOCTL_HID_GET_REPORT_DESCRIPTOR control code. http://msdn.microsoft.com/en-us/library/windows/hardware/ff541147(v=vs.85).aspx, 26.07.2013, Abruf am 06.10.2013 00:10.

[MSDN2013z] *Microsoft Corporation*: IOCTL_HID_READ_REPORT control code. http://msdn.microsoft.com/en-us/library/windows/hardware/ff541172(v=vs.85).aspx, 26.07.2013, Abruf am 06.10.2013 00:10.

[MSDN2013aa] *Microsoft Corporation*: Creating I/O Queues. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540783(v=vs.85).aspx, 05.09.2013, Abruf am 06.10.2013 00:10.

[MSDN2013ab] *Microsoft Corporation*: Request Handlers. http://msdn.microsoft.com/en-us/library/windows/hardware/ff544583(v=vs.85).aspx, 05.09.2013, Abruf am 06.10.2013 00:10.

[MSDN2013ac] *Microsoft Corporation*: Using the Bluetooth Driver Stack. http://msdn.microsoft.com/en-us/library/windows/hardware/ff536853(v=vs.85).aspx, 05.09.2013, Abruf am 06.10.2013 21:00.

[MSDN2013ad] *Microsoft Corporation*: WdfFdoQueryForInterface method. http://msdn.microsoft.com/en-us/library/windows/hardware/ff547289(v=vs.85).aspx, 05.09.2013, Abruf am 06.10.2013 21:07.

[MSDN2013ae] *Microsoft Corporation*: EvtDriverDeviceAdd function. http://msdn.microsoft.com/en-us/library/windows/hardware/ff541693(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 15:07.

[MSDN2013af] *Microsoft Corporation*: EvtDevicePrepareHardware function. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540880(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 15:07.

[MSDN2013ag] *Microsoft Corporation*: EvtDeviceD0Entry function. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540848(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 15:07.

[MSDN2013ah] *Microsoft Corporation*: EvtDeviceD0Exit function. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540855(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 15:07.

[MSDN2013ai] *Microsoft Corporation*: EvtDeviceReleaseHardware function. http://msdn.microsoft.com/en-us/library/windows/hardware/ff540890(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 15:07.

[MSDN2013aj] *Microsoft Corporation*: WdfObjectDelete method. http://msdn.microsoft.com/en-us/library/windows/hardware/ff548734(v=vs.85).aspx, 05.09.2013, Abruf am 07.10.2013 18:35.

[Nint2009] *Nintendo of America Inc.*: Wii Operations Manual: System Setup. http://www.nintendo.com/consumer/downloads/WiiOpMn_setup.pdf , Abruf am 30.09.2013 14:30.

[USBI2001] *USB Implementers Forum, Inc.*: Universal Serial Bus (USB): Device Class Definition for Human Interface Devices (HID). Version 1.11, 2001.

[Valv2013] *Valve Corporation*: Steam-Hard- & Software-Umfrage: August 2013. http://store.steampowered.com/hwsurvey, Abruf am 25.09.2013 12:46.

[Wiib2013] *WiiBrew*: Wiimote. http://wiibrew.org/wiki/Wiimote, 03.10.2012, Abruf am 29.09.2013 23:06

II. Abbildungsverzeichnis

III. Tabellenverzeichnis