Interfacing
pei tel USB Audio Devices

PS12 USB family

PS20 USB family

HA11 USB family

Application Note 1002

Revision: 1.2

February 2016

pei tel Communications GmbH
A member of the peiker group
www.peitel.de

TABLE OF CONTENTS

0 History

Date	Revision	Author	Comments
APR 2010	1.0	TS	First Draft
MAY 2010	1.1	TS	First Release
FEB 2016	1.2	TS	Update to new USB PCB

Table 1: History

0.1 Related Documents

Nr.	Name	Revision/Date	Filename	Remarks
-	-	-	-	-

Table 2: Related Documents

1 General

1.1 Disclaimer

All information and data contained in this data sheet are without any commitment, are not to be considered as an offer for conclusion of a contract, nor shall they be construed as to create any liability.

Any new issue of this Application Note invalidates previous issues. Further, pei tel Communications GmbH reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of pei tel Communications GmbH.

1.2 References

[1] **USB specification, USB HID specification, USB HID usages tables, etc.**
www.usb.org

[2] **MSDN Developer Center/Library**
http://msdn.microsoft.com/en-us/library

[3] **USB Complete, Jan Axelson**
www.lvr.com

1.3 Introduction

This document gives a reference of how to interface a **pei tel USB Audio device** of the type **PS12 USB family, PS20 USB family** and **HA11 USB family** by a user's application.
It is assumed that the reader is familiar with basic USB knowledge.
Due to the discontinuation of the previously used USB chip a redesign of hardware and software became necessary. The interface was kept as close as possible to previous USB products.

1.4 Getting on the USB

All required drivers will be installed automatically when plugged into a free USB port the first time. You will be notified when installation has been completed.
In case that the driver is not found automatically, the driver can be downloaded from our website: www.peitel.de.
pei tel USB Audio devices will be recognized by Windows® Operating Systems as a **Composite Device** which consists of an **USB Sound Device** and a **USB HID Device** (Human Interface Device).
pei tel USB Audio devices use the **Vendor ID 0x2B2E.** The device name (product name) of all pei tel USB Audio devices will always be read as "**PTC USB**".
The Product ID is related to the type of product that is used. Refer to table 1 in section **1.5. Type of Product.**
All pei tel USB Audio devices do fully support Chapter 9 of the USB specification.
Since pei tel USB Audio devices are equipped differently, parts of this document may not be relevant. For instance, a PS12 USB does not have a speaker, hence adjusting speaker volume does not have an effect.

1.5 Type of Product

Currently the following types of product are defined:

Product	Product ID	Article Number
PS12	0x0120	6401-001-001-40
PS12	0x0110	6401-002-001-41
PS12	0x0100	6401-003-001-40
PS12	0x0130	6401-003-001-41
PS20	0x0220	6402-008-001-40
PS20	0x0230	6402-008-001-41
HA11	0x0400	6506-011-000-51
HA11	0x0410	6506-011-001-51

Table 1: Type of Product

The list may be subject to be extended or shortened.

Multiple devices with the same Product ID connected to one host system can be differentiated by the Serial Number String.

The Serial Number String can be read via a call to ***HidD_GetSerialNumberString()***. The structure of the Serial Number String is as follows:

aaa.aaa.bbb.bbb.ccc.xxxxxx

where

aaa.aaa	equals Hardware Version, 6 digits
bbb.bbb	equals Main Firmware Version, 6 digits
ccc	equals Type of Product, 3 digits
xxxxxx	equals Serial Number, 6 digits

2 USB Sound Device

Since the **WINDOWS AUDIO MIXER** is very complex, it is way beyond to describe all possibilities in this document. The pei tel USB Sound Device appears as an external sound card, which is usually automatically selected when plugged into the USB port.
So changing its volume levels (microphone and speaker) can be achieved by

- using standard Windows® API functions or
- manually through the Windows® Mixer Panel(s)

Please refer to the MSDN (Reference 2) for more information on the Windows® API functions and procedures for mixer and audio control.

3 USB HID device

For Windows® operating systems, the following tasks are required by the application to communicate with a pei tel USB Audio device:

- Opening the device
- Reading and Writing Reports
- Closing the device

All these procedures are described in detail in the book **USB Complete** (Reference 3), so here we will keep it short.

3.1 Opening the Device

The process of opening a device consists of seven steps as described below.

(1) Obtain the Windows® Globally Unique ID (GUID) for HID devices via a call to **HidD_GetHidGuid()**.

(2) Get an array of structures that contain information about all attached HIDs via a call to ***SetupDiGetClassDevs()***. This uses the previously obtained HID GUID to specify that the list should only contain HID devices

(3) Use the Windows® function ***SetupDiEnumDeviceInterfaces()*** to get information about a particular device in the list. We need to step through each index of device information until we find one with the correct VID & PID. If this function returns FALSE, then we have gone to the end of the list without finding the desired device.

(4) A call to ***SetupDiGetDeviceInterfaceDetail()*** returns detailed data about the particular device indexed in the previous step. We want to use the device path to open the device in the next step.

(5) Call ***CreateFile()*** to open the device using the path obtained in the previous step. If a valid handle is returned, then we can examine the VID & PID to determine if this is the device we want.

(6) Compare the open device's VID & PID to determine if this is the device we want. If so, then we should return the device handle and a TRUE condition.

(7) If the VID & PID are not correct, then we need to close the device handle with a call to ***CloseHandle()*** and return to step 3 to check the next device indexed in the list.

3.2 Reading and Writing Reports

Obviously the next functionality that the application needs to have is the ability to read input reports and write output reports.

Applications can use the following API functions to send and receive reports:

API Function	Purpose
WriteFile()	Send an Output Report
ReadFile()	Read an Input Report

Table 4: API Functions for reading and writing reports

To send and receive reports an application should use the ***ReadFile()*** and ***WriteFile()*** functions respectively. Each of these functions use the device handle returned by the ***CreateFile()*** function which need to be called prior to an Read/Write operation.

3.2.1 Writing Output Report

Sending an Output Report is done via the function ***WriteFile()*** and is mainly used for setting and switching external output components like LEDs and to prompt the firmware to provide an Input Report for the following ***ReadFile()*** operation.

The Output Report consists of 2 Bytes:

ReportID	OutputByte

- **ReportID**
  The ReportID is always 0x00
- **OutputByte**
  The OutputByte is used for the setting of external output components and request to provide an input report. Check **Appendix Section A** for coding of OutputByte for appropriate pei tel USB Audio device.

3.2.2 Reading Input Report

Windows' HID driver causes the host controller to request Input Reports periodically. The driver stores received reports in a buffer. ***ReadFile()*** retrieves one or more reports from the buffer. If the buffer is empty, ***ReadFile()*** waits for a report to arrive. ***ReadFile()*** thus does not cause the device to send a report but just reads reports that the driver has requested.
When called without overlapped flag, ***ReadFile()*** is a blocking call and the application's thread waits until either a new report is available, the user closes the application via the Taskmanager or the device is removed from the bus.
There are two main approaches to avoid this obviously bad condition:

(1) Use of a separate thread (multitasking) for the ***ReadFile()*** operation that notifies the main thread when a report is available
(2) Prompt the firmware to provide a report by sending a special output report via ***WriteFile()*** prior to each ***ReadFile()***

The Input Report consists of 3 Bytes:

ReportID	InputByte0	InputByte1

- **ReportID**
  The ReportID is always 0x00
- **InputByte0**
  The InputByte0 is reserved for future purpose and is always 0x00
- **InputByte1**
  The InputByte1 gives the current state of buttons and external input components. Check **Appendix Section B** for coding of InputByte1 for appropriate pei tel USB Audio device.

3.3 Closing the Device

The closing of the device is fortunately much easier than opening the device. Make a call to ***CloseHandle()*** with the handle obtained by the ***CreateFile()*** during the opening procedure.

Appendix A-1: OutputByte PS12

Type of Product: 1xx

Layout: The OutputByte for the PS12 USB is coded as shown below:

	7	6	5	4	3	2	1	0	Bit
PS12	**RIR**	**MUTE**	**DMIC**	**-**	**-**	**-**	**CLED**	**-**	**OutputByte**
Default	-	0	1	-	-	-	0	-	

- **"-": Reserved**
  These Bits are reserved and should be set to "0"

- **CLED: Center LED**
  Sets or unsets the center LED, if equipped

  0 Turn off center LED

  1 Turn on center LED

- **MUTE, DMIC: Mute and Microphone select**
  Selects the microphone and mute state according to the following table:

MUTE	DMIC	Active Microphone	Mute State
0	0	ignored (no change)	ignored (no change)
0	1	Desktop Microphone	No Mute
1	X	Don't care	Mute

X = don't care

Remark: The microphone can also be muted manually via the Mute Checkbox in the Recording Panel of the Windows ® Mixer Panel.

- **RIR: Request Input Report**
  Prompt the device to provide Input Report for following *ReadFile()* operation.

  0 Input Report is not requested => normal Set operation

  1 Device is requested to provide Input Report

Range:	0x00 – 0x7F	Set operation only
	0x80	Prompt to provide Input Report only (**no change of outputs**)
	0x81 – 0xFF	Set operation **and** prompt to provide Input Report

Appendix A-2: OutputByte PS20

Type of Product: 2xx

Layout: The OutputByte for the PS20 USB is coded as shown below:

	7	6	5	4	3	2	1	0	Bit
PS12	**RIR**	**MUTE**	**DMIC**	**HMIC**	**SPK**	**RLED**	**CLED**	**LLED**	**OutputByte**
Default	-	0	1	0	1	0	0	0	

- **LLED: Left LED**
  Sets or unsets the left LED, if equipped

  0 Turn off left LED

  1 Turn on left LED

- **CLED: Center LED**
  Sets or unsets the center LED, if equipped

  0 Turn off center LED

  1 Turn on center LED

- **RLED: Right LED**
  Sets or unsets the right LED, if equipped

  0 Turn off right LED

  1 Turn on right LED

- **SPK: Speaker**
  Activates or deactivates the speaker in the device

  0 Deactivate speaker

  1 Activate speaker

  Remark: The speaker can also be muted manually via the Mute Checkbox in the Playback Panel of the Windows ® Mixer Panel.

- **MUTE, DMIC, HMIC: Mute and Microphone select**
  Selects the active microphone and mute state according to the following table:

MUTE	DMIC	HMIC	Active Microphone	Mute State
0	0	0	ignored (no change)	ignored (no change)
0	0	1	Headset Microphone	No Mute
0	1	0	Desktop Microphone	No Mute
0	1	1	ignored (no change)	ignored (no change)
1	X	X	Don't care	Mute

X = don't care

Remark: The microphone can also be muted manually via the Mute Checkbox in the Recording Panel of the Windows ® Mixer Panel.

- **RIR: Request Input Report**
  Prompt the device to provide Input Report for following *ReadFile()* operation.

  0 Input Report is not requested => normal Set operation

  1 Device is requested to provide Input Report

Range:

0x00 – 0x7F	Set operation only
0x80	Prompt to provide Input Report only (**no change of outputs**)
0x81 – 0xFF	Set operation **and** prompt to provide Input Report

Remark(s): When a headset is connected, the device automatically selects the Headset Microphone as the active microphone and the microphone will be unmuted. The application does not have to do this, but should keep it in mind upon Headset detection. The application may select the Desktop Microphone on demand afterwards.

When the Headset is disconnected, the device automatically selects the Desktop Microphone, the microphone is unmuted and the Speaker is activated, even if it was deactivated before. The application does not have to do this, but should keep it in mind upon Headset removal detection. The reason is that many people just

remove their Headset at the end of the shift and leave. The successor then might not know that the microphone is muted and the speaker is deactivated. The speaker of the Headset cannot be deactivated.

Appendix A-3: OutputByte HA11

Type of Product: 4xx

Layout: The OutputByte for the HA11 USB is coded as shown below:

	7	6	5	4	3	2	1	0	Bit
PS12	**RIR**	**MUTE**	**DMIC**	**-**	**SPK**	**-**	**-**	**-**	**OutputByte**
Default	-	0	1	-	-	-	0	-	

- **"-": Reserved**
  These Bits are reserved and should be set to "0"

- **SPK: Speaker**
  Activates or deactivates the speaker in the device

  0 Deactivate speaker

  1 Activate speaker

- **MUTE, DMIC: Mute and Microphone select**
  Selects the microphone and mute state according to the following table:

MUTE	DMIC	Active Microphone	Mute State
0	0	ignored (no change)	ignored (no change)
0	1	Desktop Microphone	No Mute
1	X	Don't care	Mute

X = don't care

Remark: The microphone can also be muted manually via the Mute Checkbox in the Recording Panel of the Windows ® Mixer Panel.

- **RIR: Request Input Report**
  Prompt the device to provide Input Report for following *ReadFile()* operation.

  0 Input Report is not requested => normal Set operation

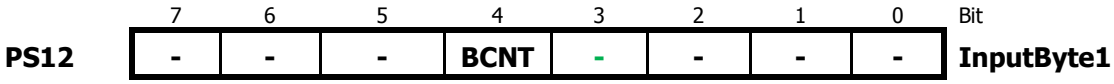
  1 Device is requested to provide Input Report

Range:	0x00 – 0x7F	Set operation only
	0x80	Prompt to provide Input Report only (**no change of outputs**)
	0x81 – 0xFF	Set operation **and** prompt to provide Input Report

Appendix B-1: InputByte1 PS12

Type of Product: 1xx

Layout: The InputByte1 for the PS12 USB is coded as shown below:



- **"-": Reserved**
  These Bits are reserved and read as "1"

- **BCNT: Center Button**
  Determines current state of the center button

  0 Center button pressed (active)

  1 Center button released (inactive)

Appendix B-2: InputByte1 PS20

Type of Product: 2xx

Layout: The InputByte1 for the PS20 USB is coded as shown below:

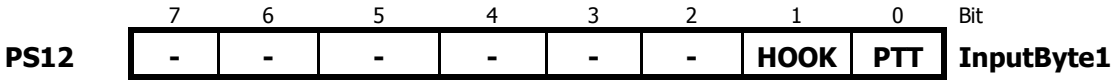
	7	6	5	4	3	2	1	0	Bit
PS20	**BRG**	**BEXT**	**BLFT**	**BCNT**	**-**	**EHS**	**-**	**-**	**InputByte1**

- **"-": Reserved**
  These Bits are reserved and read as "1"
- **EHS: External Headset**
  Determines whether there is an external headset connected or not.
  - 0 External headset connected
  - 1 No external headset connected
- **BCNT: Center Button**
  Determines current state of the center button
  - 0 Center button pressed (active)
  - 1 Center button released (inactive)
- **BLFT: Left Button**
  Determines current state of the left button
  - 0 Left button pressed (active)
  - 1 Left button released (inactive)
- **BEXT: External Button**
  Determines current state of the external button (usually external PTT switch)
  - 0 External button pressed (active)
  - 1 External button released (inactive)
- **BRG: Right Button**
  Determines current state of the right button
  - 0 Right button pressed (active)
  - 1 Right button released (inactive)

Appendix B-3: InputByte1 HA11

Type of Product: 4xx

Layout: The InputByte1 for the PS12 USB is coded as shown below:



- **"-": Reserved**
  These Bits are reserved and read as "1"

- **PTT: PTT Button**
  Determines current state of the PTT button

  0 Center button pressed (active)

  1 Center button released (inactive)

- **HOOK: HOOK Contact**
  Determines current state of the HOOK contact

  1 Handset off Hook

  0 Handset on Hook

END OF DOCUMENT