PKTUSBAPI

Interface between the user C# application and PrehKeyTec USB devices

USER MANUAL

AUTHOR: THOMAS MEUSERT

Content

1. Disclaimer

PrehKeyTec GmbH reserves the right to make changes in specifications and other information contained in this document without prior notice. The reader should consult PrehKeyTec whether any such changes have been made. The information in this manual does not represent a commitment on the part of PrehKeyTec.

Whilst every care has been taken in producing this manual PrehKeyTec shall not be liable for technical or editorial errors or omissions contained herein, nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be copied without the prior written consent of PrehKeyTec.
© PrehKeyTec GmbH. All rights reserved.

Product names or marks mentioned in this document may be trademarks or registered trademarks of other companies and are the property of their respective owners.

2. History

Version	Date	Description
1.0	2017-07-14	Document created
1.0.0.5	2017-12-13	Updated all functions, sorted document content
5.1.0.0	2018-01-12	Updated copyright and hardware information Updated OnDeviceStatusChanged event Added SetLEDAcceptOnly function Updated example source code
5.1.0.1	2018-04-18	Added WriteLog() Method Updated Constants Added methods and constants for backlight control (only SIK,VC,MWX810)
5.1.0.2	2018-05-04	Added Concentrate Mode with Example Code
5.1.0.3	2018-08-30	Updated Constants, added device methods to dis/enable scanning and barcodes, added device method to take snapshot (PKT4000)
5.1.0.8	2019-04-30	Updated PKTUSB.ini section. Updated OCR-Data Structure
5.1.0.11	2019-09-26	Added LCD Display Methods
5.1.0.13	2020-01-07	Optimization Keytable download. Official release.
5.2.0.00	2023-05-03	Discontinued devices PKT4000/Hand scanner were removed.
5.2.0.01	2023-07-06	Optimizations regarding ini file: Local pktusb.ini has priority. Official release.

3. Description

The ‘PKTUSBAPI.dll’ helps to communicate with PrehKeyTec USB keyboards. It is the interface between the hardware and the customer application. This library is written in C# with Microsoft® .NET Framework 4.0. The following chapters describe all functions of the API and how to implement them in customer’s source code.

4. Requirements

4.1. Software

Currently the only specific requirements are as detailed below, however please feel free to check with Technical Support if you have any additional questions requiring these or any other system requirements.

System Requirements

Computer/Processor	32 Bit (x86) or 64 Bit (x64)
Operating System	Windows® 7 or later
Related Software	Microsoft® .NET Framework 4.x or later

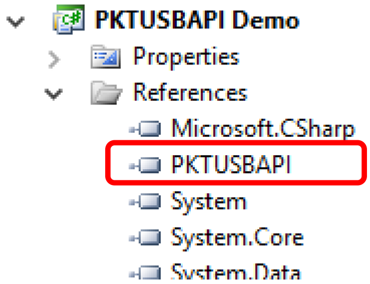
4.2. Hardware

The DLL supports all PrehKeyTec USB devices with the Firmware 605/3xxx and 605/5xxx or later.

5. API Description

5.1. Implementation

Start by adding the PKTUSBAPI into your application, the following data should be seen in the properties of your application:



Select

```
using PKTUSBAPI;
```

to enable you to have access to the features, methods and parameters of the PKTUSBAPI.

IMPORTANT NOTE:

In addition of including the PKTUSBAPI.dll to your application please make sure that the two C++ DLLs (PKTUSB32.dll and PKTUSB64.dll) are located in the same directory as the PKTUSBAPI.dll in your application.

5.2. INI File

Some API settings can be changed and set up with an INI file.
By default, the INI file called '*PKTUSB.ini*' will be located in following directory:

%APPDATA%\PrehKeyTec\PKTUSB\YourExeName\PKTUSB.ini

The INI file will be created by the API e.g. if you call the method *myAPI.StartLog(…)*. You can also create the INI file by yourself.

Important note: Starting with API Version 5.2, a local pktusb.ini has priority. In case such config file pktusb.ini is found in the application´s folder, this one will be used!

5.2.1. Example File

The code below shows an example of the 'PKTUSB.ini' file.

```
 [Logging]
File=D:\temp\MyLogFile.txt
Level=15
FileAppend=On
[Device0]
VendorID=0x053a
ProductID=0xb09
[Device1]
DevicePath=0b08
[Device2]
SearchText=MCI 3000
UsagePage = 0xFFFA
```

5.2.2. Section [Logging]

Key	Description
`File`	Name of the log file. If value exists logging is activated, if not logging is deactivated.
`Level`	Log level to log the different activities of the PKTUSBAPI.dll and its PKTUSBxx.dll
`FileAppend`	Append all log messages to the existing log file or overwrite existing file. (On/Off)

5.2.3. Section [DeviceX]

The X in the section name is the identifier of the alias device. This value of X has to be passed to the *myAPI.OpenDeviceAlias(X)* method to open a device with the attributes of the [DeviceX] section. This method opens only a device if all attributes of this section matches with the attributes of a connected device.

Key	Description
`VendorID`	Define a vendor ID (hex format, e.g. '0x0b08')
`ProductID`	Define a product ID (hex format, e.g. '0x053a')
`UsagePage`	Define a usage page (hex format, e.g. '0xfffa')
`DevicePath`	Define a string that must be contained in the device path.
`SearchText`	Define a string that must be contained in the device version string.

5.3. Initialize the PKTUSBAPI

The class ‘PKTUSBAPI_Class’ is the main class of the DLL. Create an object of the class ‘PKTUSBAPI_Class’ at the start of your application to initialize the PKTUSBAPI and the native PKTUSB(32Bit/64Bit) C++ DLL.

```
PKTUSBAPI_Class myAPI = new PKTUSBAPI_Class();
```

By default the PKTUSBAPI_Class shows all PrehKeyTec USB devices with the standard usage page 0xfffa. If you have to operate with other PrehKeyTec USB devices (e.g. the airline devices) you can change the device filter as shown below.

```
//Show all existing PrehKeyTec USB Devices
PKTUSBAPI_Class myAPI = new PKTUSBAPI_Class(null);
```

or:

```
//Show all PrehKeyTec USB devices with customized usage page filter
//(e.g. standard devices(0xfffa) and airline devices(0xfff9))
ushort[] filter = new ushort[] { 0xfffa, 0xfff9 };
PKTUSBAPI_Class myAPI = new PKTUSBAPI_Class(filter);
```

In case you do not want to explicitly communicate with a specific device, Concentrate Mode is available.
This receives all data from all connected devices and outputs them in a common event listener.

For more details see the chapter ***Concentrate Mode***.

5.3.1. Log File

It is possible to activate a log file. There are a few log levels to log the different activities of the PKTUSBAPI.dll and its PKTUSBxx.dll in one log file. The logging settings can be set up via methods programmatically or directly in the APIs INI-file. It is also possible to add your own messages to this Log-File.

5.3.1.1.Start Logging

```
//Name of my log file (text file)
String LogFileName = "D:\\temp\\MyLogFile.txt";

//Log only errors, warnings and inforamtion
int LogLevel = Constants.LOG_LEVEL_ERROR_WARNING_INFO;

//Append new messages to the log file if it already exists
bool LogAppend = true;

//Activate log file
int ret = myAPI.StartLog(LogFileName, LogAppend, LogLevel);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Logging successful started");
}
else
{
   Console.WriteLine("Starting logging failed");
}
```

Log Level:

Name	Description
`Constants.LOG_LEVEL_ALL`	Log all activities.
`Constants.LOG_LEVEL_ERROR_WARNING_INFO_DATA`	Log only errors, warnings, information and data.
`Constants.LOG_LEVEL_ERROR_WARNING_INFO`	Log only errors, warnings and information.
`Constants.LOG_LEVEL_ERROR_WARNING`	Log only errors and warnings.
`Constants.LOG_LEVEL_ERROR`	Log only errors.

LogAppend

Value	Description
`true`	Append all log messages to the existing log file.
`false`	Overwrite existing log file.

5.3.1.2.Stop Logging

```
//Deactivate log file
int ret = myAPI.StopLog();
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Logging successful stopped");
}
else
{
   Console.WriteLine("Stopping logging failed");
}
```

5.3.1.3. Write to Log-File

```
//Write your message to Log-File
int ret = myAPI.WriteLog("This message is from Customer.", Constants.LOG_INFO);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Writing to Log-File successful");
}
else
{
   Console.WriteLine("Writing to Log-File failed");
}
```

5.3.2. Event Handler: OnDeviceStatusChanged

The class ‘PKTUSBAPI_Class’ has an event handler that triggers an event if a device was connected, disconnected, opened or closed. The event handler represents the method that will handle an event.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on device status changed event
myAPI.OnDeviceStatusChanged += new
EventHandler<EventArguments.OnDeviceStatusChanged>(myAPI_OnDeviceStatusChanged);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnDeviceStatusChanged(object sender,
EventArguments.OnDeviceStatusChanged e)
{
    // The connected status of the device has changed
    if (e.ConnectedStatusChanged)
    {
       Console.WriteLine("Device " + e.ChangedDevice.DevicePath + " was " +
       (e.ChangedDevice.IsConnected?"connected":"disconnected"));
    }

    // The opened status of the device has changed
    if (e.OpenedStatusChanged)
    {
       Console.WriteLine("Device " + e.ChangedDevice.DevicePath + " was " +
       (e.ChangedDevice.IsOpened ? "opened" : "closed"));
    }

    //Get all currently connected devices
    List<PKTUSBAPI_Class.DeviceInfo> listDevices = e.NewDeviceList;
}
```

If the *OnDeviceStatusChanged* is triggered, the defined method *myAPI_OnDeviceStatusChanged* is called.

5.3.2.1. Event Arguments: OnDeviceStatusChanged

Type	Name	Description
`List<PKTUSBAPI_Class.DeviceInfo>`	NewDeviceList	The new list of connected devices.
`PKTUSBAPI_Class.DeviceInfo`	ChangedDevice	Device information of changed device.
`bool`	ConnectedStatusChanged	The device was connected or disconnected.
`bool`	OpenedStatusChanged	The device was opened or closed.

5.3.3. Get connected Device List

As shown above you can get a list of all connected devices from the '*OnDeviceStatusChanged'* Event Handler. A second possibility to get the list is to call the 'ConnectedDevices' property of the PKTUSBAPI_Class.

```
//Get all currently connected devices
List<PKTUSBAPI_Class.DeviceInfo> listDevices = myAPI.ConnectedDevices;
```

5.3.4. Structure 'DeviceInfo'

The following structure contains all USB and product information of a PrehKeyTec device. With this structure you can open the device you are searching for.

Type	Name	Description
String	DevicePath	USB device path.
ushort	VendorID	USB vendor ID.
ushort	ProductID	USB product ID.
ushort	UsagePage	USB usage page.
bool	IsConnected	Device is connected.
bool	IsOpened	Device is opened.
String	VersionString	Complete PrehKeyTec version string.
String	Copyright	PrehKeyTec copyright information.
String	FirmwareVersion	PrehKeyTec firmware version.
String	FirmwareDate	PrehKeyTec firmware date.
String	BootloaderVersion	PrehKeyTec bootloader version.
String	ProductCode	PrehKeyTec product code.
String	SerialNumber	PrehKeyTec serial number.
String	ProductNumber	PrehKeyTec product number.

5.4. Open a Device

There are different ways to open a device. All of these methods returns an USB_Device class with the successful opened device or `null` if the device you tried to open does not exist or open device failed.

The first is to open the device that was found first by the API. That way can be used if you only have connected one device at the same time.

```
//Open the device that was found first from the API
USB_Device myDevice = myAPI.OpenDevice();
if (myDevice != null)
{
   Console.WriteLine("Device was opened:");
   Console.WriteLine(myDevice.DevicePath);
   Console.WriteLine(myDevice.VersionString);
}
```

The second way is to open a specified device. As shown above you can get a list of all connected devices. If you find your device in this list you can open your device as shown below:

```
//Get all currently connected devices
List<PKTUSBAPI_Class.DeviceInfo> listDevices = myAPI.ConnectedDevices;

//Find my device
foreach (PKTUSBAPI_Class.DeviceInfo device in listDevices)
{
   if (device.ProductID == 0x0b06)
   {
       //Open my device
       USB_Device myDevice = myAPI.OpenDevice(device);
       if (myDevice != null)
       {
           Console.WriteLine("Connected to device:");
           Console.WriteLine(myDevice.DevicePath);
           Console.WriteLine(myDevice.VersionString);
       }
       break;
    }
}
```

Another way to open a device is to open an alias device that was specified in the APIs INI-File. For more information about the INI-File please read the chapter ‘INI-File’.

```
//Open the device alias number 2 from INI-File
USB_Device myDevice = myAPI.OpenDeviceAlias(2);
if (myDevice != null)
{
   Console.WriteLine("Connected to device:");
   Console.WriteLine(myDevice.DevicePath);
   Console.WriteLine(myDevice.VersionString);
}
```

For more open device methods see capture ‘Class: PKTUSBAPI_Class’.

5.5. Send Data

To communicate with the device there are a few methods to send specified commands to the device. Below you find an example how to use these methods.

5.5.1. Example: Command play 'Error' tone

```
int volume = 50; //Tone Volume: 0-100 %

//Send command to play 'Error' tone
int ret = myDevice.PlayTone(Constants.TONE_ERROR, volume);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Played 'Error' tone successful");
}
else
{
   Console.WriteLine("Playing 'Error' tone failed");
}
```

5.5.2. Example: Command control device LEDs

```
int acceptColor = Constants.ACCEPT_ORANGE;      //Accept LED color: Orange
int acceptFlash = Constants.ACCEPT_FLASH_600ms;//Accept LED speed: flashing 600ms
bool num = false;                               //Switch Off Num-Lock LED
bool scroll = true;                             //Switch On  Scroll-Lock LED
bool caps = true;                               //Switch On  Caps-Lock LED

//Send command to control the device LEDs
int ret = myDevice.SetLEDs(num, caps, scroll, acceptColor, acceptFlash);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Controlled device LEDs successful");
}
else
{
   Console.WriteLine("Controlling device LEDs failed");
}
```

5.5.3. Example: Send Byte Array to Device

```
byte[] data = new byte[] { 0xef, 0x2b }; //Command: Play 'Beep' tone

//Send data to device
int ret = myDevice.WriteData(data);
if (ret > 0)
{
   Console.WriteLine(ret + " byte(s) sent to device successful");
}
else
{
   Console.WriteLine("Sending data to device failed");
}
```

5.6. Receive Data

There are a few event handlers receive data from the device modules.

The following code shows how to implement and use one of the devices event handlers.

5.6.1. Event Handler: OnReceivedData

The following code shows how to receive any data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received any data event for my device
myDevice.OnReceivedData += new
EventHandler<EventArguments.OnReceivedData>(myDevice_OnReceivedData);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnReceivedData(object sender, EventArguments.OnReceivedData e)
{
   Console.WriteLine("[Data]");
   Console.WriteLine(e.RawData.Length + " byte(s) received");
}
```

5.6.1.1. Event Arguments: OnReceivedData

Type	Name	Description
`byte[]`	RawData	Received binary data.

5.6.2. Event Handler: OnReceivedMSR

The following code shows how to receive MSR data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received MSR data event for my device
myDevice.OnReceivedMSR +=new
EventHandler<EventArguments.OnReceivedMSR>(myDevice_OnReceivedMSR);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnReceivedMSR(object sender, EventArguments.OnReceivedMSR e)
{
   Console.WriteLine("[MSR]");
   Console.WriteLine("Track 1: " + e.MSRtrack1);
   Console.WriteLine("Track 2: " + e.MSRtrack2);
   Console.WriteLine("Track 3: " + e.MSRtrack3);
}
```

5.6.2.1. Event Arguments: OnReceivedMSR

Type	Name	Description
`byte[]`	RawData	Raw binary MSR data.
`String`	MSRtrack1	MSR track 1 as string.
`String`	MSRtrack2	MSR track 2 as string.
`String`	MSRtrack3	MSR track 3 as string.

5.6.3. Event Handler: OnReceivedOCR

The following code shows how to receive OCR data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received OCR data event for my device
myDevice.OnReceivedOCR += new
EventHandler<EventArguments.OnReceivedOCR>(myDevice_OnReceivedOCR);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnReceivedOCR(object sender, EventArguments.OnReceivedOCR e)
{
   Console.WriteLine("[OCR]");
   Console.WriteLine("MRZ line 1: " + e.MRZline1);
   Console.WriteLine("MRZ line 2: " + e.MRZline2);
   Console.WriteLine("MRZ line 3: " + e.MRZline3);
}
```

5.6.3.1. Event Arguments: OnReceivedOCR

Type	Name	Description
`byte[]`	RawData	Raw binary OCR data.
`String`	Type	Passport type.
`String`	Code	Country code.
`String`	PassportNo	Passport number.
`String`	Surname	Surname.
`String`	Givenname	Givenname
`String`	Nationality	Nationality
`String`	NationalityCode	Nationality 3-Letter-Code.
`String`	DateOfBirth	Date of birth. (YYMMDD)
`String`	DateOfExpiry	Date of expiry. (YYMMDD)
`String`	Sex	Sex.
`String`	OptionalData1	Optional data 1.
`String`	OptionalData2	Optional data 2.
`String`	MRZline1	MRZ line 1.
`String`	MRZline2	MRZ line 2.
`String`	MRZline3	MRZ line 3.

5.6.4. Event Handler: OnReceivedAUX

The following code shows how to receive AUX port data from the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received AUX data event for my device
myDevice.OnReceivedAUX += new
EventHandler<EventArguments.OnReceivedAUX>(myDevice_OnReceivedAUX);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnReceivedAUX(object sender, EventArguments.OnReceivedAUX e)
{
   Console.WriteLine("[AUX]");
   Console.WriteLine(e.RawData.Length + " byte(s) received");
   Console.WriteLine("Data: " + e.DataString);
}
```

5.6.4.1. Event Arguments: OnReceivedAUX

Type	Name	Description
`byte[]`	RawData	Raw binary AUX port data.
`String`	DataString	AUX port data as string.

5.6.5. Event Handler: OnReceivedPOSkey

The following code shows how to receive POS key data from the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received POS key event for my device
myDevice.OnReceivedPOSkey += new
EventHandler<EventArguments.OnReceivedPOSkey>(myDevice_OnReceivedPOSkey);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnReceivedPOSkey(object sender, EventArguments.OnReceivedPOSkey e)
{
   Console.WriteLine("[POS Key]");
   Console.WriteLine("Key:    " + e.Key);
   Console.WriteLine("Status: " + (e.Pressed?"pressed":"released"));
}
```

5.6.5.1. Event Arguments: OnReceivedPOSkey

Type	Name	Description
`int`	Key	The POS key number.
`bool`	Pressed	Was POS key pressed (`true`) or released (`false`).

5.6.6. Event Handler: OnReceivedKeyLockChanged

The following code shows how to receive key lock position on change from the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received key lock changed event for my device
myDevice.OnKeyLockChanged += new
EventHandler<EventArguments.OnKeyLockChanged>(myDevice_OnKeyLockChanged);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnKeyLockChanged(object sender, EventArguments.OnKeyLockChanged e)
{
  Console.WriteLine("[KeyLock]");
  Console.WriteLine("Position: " + e.Position);
}
```

5.6.6.1. Event Arguments: OnReceivedKeyLockChanged

Type	Name	Description
`Int`	Position	New key lock position.

5.6.7. Event Handler: OnReceivedProgressChanged

The following code shows how to receive the current progress of a file up-/download from/to the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received progress changed event for my device
myDevice.OnProgressChanged += new
EventHandler<EventArguments.OnProgressChanged>(myDevice_OnProgressChanged);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myDevice_OnProgressChanged(object sender, EventArguments.OnProgressChanged e)
{
  Console.WriteLine("[Progress]");
  Console.WriteLine(e.Step + "%");
  Console.WriteLine(e.StepInfo);
}
```

5.6.7.1. Event Arguments: OnReceivedProgressChanged

Type	Name	Description
`int`	Step	Current up-/download progress (0-100%).
`String`	StepInfo	Description of the current up-/download step.

5.7. Read/Write Files

It is possible to read or write binary files (e.g. keytable file [*.mwx]) from/to the device.
Please note that the read/write file process will take some time. If you do not want that the process blocks your application please call the read/write file methods inside a thread.
While the read/write file process is running, the OnProgressChange event keeps your application informed about the progress of the read/write file process.

5.7.1. Example: Read keytable file from Device

Following example shows how to read the binary key table file out of a device.

```
String FileName = "D:\\temp\\MyKeytable.mwx"; //File name to store the keytable

//Start reading keytable file from device
int ret = myDevice.KeytableRead(FileName);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Reading keytable file finished successful");
}
else
{
   Console.WriteLine("Reading keytable file failed");
}
```

5.7.2. Example: Write keytable file to Device

Following example shows how to write a binary keytable file into a device.

```
String FileName = "D:\\temp\\MyKeytable.mwx"; //File name of the keytable to write

//Start writing keytable file from device
int ret = myDevice.KeytableWrite(FileName);
if (ret == ErrorStatus.Succeeded)
{
   Console.WriteLine("Writing keytable file finished successful");
}
else
{
   Console.WriteLine("Writing keytable file failed");
}
```

5.8. Close a Device

The example below shows how to close a connected and opened device. The device close method closes the device and all its background threads.

```
//Check if the device instance already exists
if (myDevice != null)
{
   //Close the device
   myDevice.Close();
   myDevice = null;
}
```

5.9. Concentrate Mode

In case you do not want to explicitly communicate with a specific device, Concentrate Mode is available.
This receives all data from all connected devices and outputs them in a common event listener.
The opening and closing of each device is automatically handled by the API and does not have to be done manually.

To activate this concentrate mode just add the Boolean value ‘true’ to the initialization of the PKTUSBAPI_Class.

```
//Initialize the PKTUSBAPI (Concentrate Mode)
PKTUSBAPI_Class myAPI = new PKTUSBAPI_Class(null, true);
```

5.9.1. Event Handler: OnReceivedCommonPOSkey

The following code shows how to receive POS key data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received POS key event for all devices
myAPI.OnReceivedCommonPOSkey += new
EventHandler<EventArguments.OnReceivedCommonPOSkey>(myAPI_OnReceivedCommonPOSkey);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnReceivedCommonPOSkey(object sender,
EventArguments.OnReceivedCommonPOSkey e)
{
	Console.WriteLine("\n[POS-Key] {" + e.Device.SerialNumber + "}");
	Console.WriteLine("Key:    " + e.Data.Key);
	Console.WriteLine("Status: " + (e.Data.Pressed ? "pressed" : "released"));
}
```

5.9.1.1. Event Arguments: OnReceivedCommonPOSkey

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure ‘DeviceInfo’)
`OnReceivedPOSkey`	Data	Received data. (Event Arguments: OnReceivedPOSkey)

5.9.2. Event Handler: OnReceivedCommonOCR

The following code shows how to receive OCR data from the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received OCR data event for all devices
myAPI.OnReceivedCommonOCR += new
EventHandler<EventArguments.OnReceivedCommonOCR>(myAPI_OnReceivedCommonOCR);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnReceivedCommonOCR(object sender,
EventArguments.OnReceivedCommonOCR e)
{
        Console.WriteLine("\n[OCR] {" + e.Device.SerialNumber + "}");
        Console.WriteLine("MRZ line 1: " + e.Data.MRZline1);
        Console.WriteLine("MRZ line 2: " + e.Data.MRZline2);
        Console.WriteLine("MRZ line 3: " + e.Data.MRZline3);
}
```

5.9.2.1. Event Arguments: OnReceivedCommonOCR

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure 'DeviceInfo')
`OnReceivedOCR`	Data	Received data. (Event Arguments: OnReceivedOCR)

5.9.3. Event Handler: OnReceivedCommonMSR

The following code shows how to receive MSR data from the device.

Create and add the event with the plus '+=' operator to invoke the method.

```
//Set up on received OCR data event for all devices
myAPI.OnReceivedCommonOCR += new
EventHandler<EventArguments.OnReceivedCommonOCR>(myAPI_OnReceivedCommonOCR);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnReceivedCommonMSR(object sender,
EventArguments.OnReceivedCommonMSR e)
{
        Console.WriteLine("\n[MSR] {" + e.Device.SerialNumber + "}");
        Console.WriteLine("Track 1: " + e.Data.MSRtrack1);
        Console.WriteLine("Track 2: " + e.Data.MSRtrack2);
        Console.WriteLine("Track 3: " + e.Data.MSRtrack3);
}
```

5.9.3.1. Event Arguments: OnReceivedCommonMSR

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure 'DeviceInfo')
`OnReceivedMSR`	Data	Received data. (Event Arguments: OnReceivedMSR)

5.9.4. Event Handler: OnReceivedCommonData

The following code shows how to receive data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received data event for all devices
myAPI.OnReceivedCommonData += new
EventHandler<EventArguments.OnReceivedCommonData>(myAPI_OnReceivedCommonData);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnReceivedCommonData(object sender,
EventArguments.OnReceivedCommonData e)
{
    Console.WriteLine("\n" + e.Data.RawData.Length + " Bytes received from {" +
    e.Device.SerialNumber + "}");
}
```

5.9.4.1. Event Arguments: OnReceivedCommonData

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure ‘DeviceInfo’)
`OnReceivedData`	Data	Received data. (Event Arguments: OnReceivedData)

5.9.5. Event Handler: OnReceivedCommonAUX

The following code shows how to receive AUX data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received AUX data event for all devices
myAPI.OnReceivedCommonAUX += new
EventHandler<EventArguments.OnReceivedCommonAUX>(myAPI_OnReceivedCommonAUX);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnReceivedCommonAUX(object sender,
EventArguments.OnReceivedCommonAUX e)
{
       Console.WriteLine("\n[AUX] {" + e.Device.SerialNumber + "}");
       Console.WriteLine("Data: " + e.Data.auxData.AUX_String);
}
```

5.9.5.1. Event Arguments: OnReceivedCommonAUX

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure ‘DeviceInfo’)
`OnReceivedAUX`	Data	Received data. (Event Arguments: OnReceivedAUX)

5.9.6. Event Handler: OnCommonKeyLockChanged

The following code shows how to receive Key Lock data from the device.

Create and add the event with the plus ‘+=’ operator to invoke the method.

```
//Set up on received POS key event for all devices
myAPI.OnReceivedCommonPOSkey += new
EventHandler<EventArguments.OnReceivedCommonPOSkey>(myAPI_OnReceivedCommonPOSkey);
```

Definition of the method assigned to the EventHandler delegate.

```
static void myAPI_OnCommonKeyLockChanged(object sender,
EventArguments.OnCommonKeyLockChanged e)
{
        Console.WriteLine("\n[KeyLock] {" + e.Device.SerialNumber + "}");
        Console.WriteLine("Position: " + e.Data.Position);
}
```

5.9.6.1. Event Arguments: OnCommonKeyLockChanged

Type	Name	Description
`DeviceInfo`	Device	Device that sent the data. (Structure ‘DeviceInfo’)
`OnKeyLockChanged`	Data	Received data. (Event Arguments: OnReceivedKeyLockChanged)

6. Class: PKTUSBAPI_Class

6.1. Properties

The class ‘PKTUSBAPI_Class’ provides following properties:

Type	Name	Description
`String`	VersionDLL	Get the version of the C++ DLL ‘PKTUSBxx.dll’.
`String`	VersionAPI	Get the version of the API.
`List<DeviceInfo>`	ConnectedDevices	Get a list of all connected PrehKeyTec devices and the information about them.
`int`	NumberOfConnectedDevices	Get the number of all connected PrehKeyTec devices.

6.2. Static Functions

The class ‘PKTUSBAPI_Class’ provides following static methods:

Name	Description
`public static int GetFileChecksum(String FileName)`	Get the checksum of an external file.
`public static int GetFileLength(String FileName)`	Get the length of an external file.
`public static String GetApplicationVersion(String AppName)`	Get version of an external application.
`public static String ByteArrayToHexString(byte[] Data)`	Convert a byte array to hex string.

6.2.1. GetFileChecksum

Syntax:	`public static int GetFileChecksum(String FileName)`
Parameter:	`String FileName`: The name of the file
Return Value:	The checksum of the file or `ErrorStatus.Failed`
Description:	Get the checksum of an external file.

6.2.2. GetFileLength

Syntax:	`public static int GetFileLength(String FileName)`
Parameter:	`String FileName`: The name of the file
Return Value:	The length of the file or `ErrorStatus.Failed`
Description:	Get the length of an external file.

6.2.3. GetApplicationVersion

Syntax:	`public static String GetApplicationVersion(String AppName)`
Parameter:	`String AppName`: The name of the application.
Return Value:	The version of the application.
Description:	Get version of an external application.

6.2.4. ByteArrayToHexString

Syntax:	`public static String ByteArrayToHexString(byte[] Data)`
Parameter:	`byte[] Data`: The data array to convert.
Return Value:	The converted byte array as Hex string.
Description:	Convert a byte array to hex string.

6.3. Public Functions

The class ‘PKTUSBAPI_Class’ provides following public methods:

Name	Description
`public USB_Device OpenDevice()`	Open the first device that is found.
`public USB_Device OpenDevice(DeviceInfo Info)`	Open a device with a specified device information structure.
`public USB_Device OpenDevice(ushort UsagePage)`	Open a device with a specified usage page.
`public USB_Device OpenDevice(String SearchText)`	Open a device that version string contains a specified text sequence.
`public USB_Device OpenDevice(ushort UsagePage, String SearchText)`	Open a device with a specified usage page and that version string contains a specified text sequence.
`public USB_Device OpenDeviceAlias(int Index)`	Open a device that is defined in the INI File.
`public ushort[] GetUsagePageFilter()`	Get the current usage page filter.
`public void SetUsagePageFilter(ushort filter)`	Set usage page filter. Only show devices with this usage page.
`public void SetUsagePageFilter(ushort[] filter)`	Set usage page filter. Only show devices with one of this usage pages.
`public int StartLog(String LogFile, bool Append, int LogLevel)`	Start logging of the activities of the API and the C++ dll.
`public int WriteLog(String Message, ushort Level)`	Write extern message to the Log-File of the C++ dll.
`public int StopLog()`	Stop logging of the activities of the API and the C++ dll.
`public String GetLogFile()`	Get Current Log File.
`public int GetLogLevel()`	Get Current Log Level.
`public bool GetLogFileAppend()`	Get Current LogFileAppend status.

6.3.1. Device Methods

6.3.1.1. OpenDevice

Syntax:	`public USB_Device OpenDevice()`
Description:	Open the first device that is found.
Parameter:	None
Return Value:	The opened device class or `null` if device not found.

6.3.1.2. OpenDevice

Syntax:	`public USB_Device OpenDevice(ushort UsagePage)`
Description:	Open a device with a specified usage page.
Parameter:	`ushort UsagePage`: The specified usage page.
Return Value:	The opened device class or null if device not found.

6.3.1.3. OpenDevice

Syntax:	`public USB_Device OpenDevice(String SearchString)`
Description:	Open a device that version string contains a specified text sequence.
Parameter:	`String SearchString`: The specified text sequence.
Return Value:	The opened device class or null if device not found.

6.3.1.4. OpenDevice

Syntax:	`public USB_Device OpenDevice(ushort UsagePage, String SearchString)`
Description:	Open a device with a specified usage page and that version string contains a specified text sequence.
Parameter:	`ushort UsagePage`: The specified usage page. `String SearchString`: The specified text sequence.
Return Value:	The opened device class or null if device not found.

6.3.1.5. OpenDevice

Syntax:	`public USB_Device OpenDevice(DeviceInfo Info)`
Description:	Open a device with a specified device information structure.
Parameter:	`DeviceInfo Info`: The device information structure
Return Value:	The opened device class or null if device not found.

6.3.1.6. OpenDeviceAlias

Syntax:	`public USB_Device OpenDeviceAlias(int Index)`
Description:	Open a device that is defined in the INI File.
Parameter:	`int Index`: INI file device number
Return Value:	The opened device class or null if device not found.

6.3.1.7. GetUsagePageFilter

Syntax:	`public ushort[] GetUsagePageFilter()`
Description:	Get the current usage page filter.
Parameter:	None
Return Value:	The current usage page filter array or `null` if no filter is active.

6.3.1.8. SetUsagePageFilter

Syntax:	`public void SetUsagePageFilter(ushort filter)`
Description:	Set usage page filter. Only show devices with this usage page.
Parameter:	`ushort filter: Usage page to searching for`
Return Value:	None

6.3.1.9. SetUsagePageFilter

Syntax:	`public void SetUsagePageFilter(ushort[] filter)`
Description:	Set usage page filter. Only show devices with one of this usage pages.
Parameter:	`ushort[] filter: Usage pages to searching for`
Return Value:	None

6.3.2. Logging Methods

6.3.2.1. StartLog

Syntax:	`public int StartLog(String LogFile, bool Append, int LogLevel)`
Description:	Start logging of the activities of the API and the C++ dll.
Parameter:	`String LogFile`: Name of the log file `bool Append`: Append log messages to existing log file `int LogLevel`: Log level
Return Value:	`ErrorStatus.Succeeded` if succeeded, `ErrorStatus.InvalidValue` if failed.

6.3.2.2. StopLog

Syntax:	`public int StopLog()`
Description:	Stop logging of the activities of the API and the C++ dll.
Parameter:	None
Return Value:	`ErrorStatus.Succeeded` if succeeded

6.3.2.3. WriteLog

Syntax:	`public int WriteLog(String Message, ushort Level)`
Description:	Write extern message to the Log-File of the C++ dll.
Parameter:	`String Message`: Message to log `ushort Level`: Message Log level
Return Value:	`ErrorStatus.Succeeded` if succeeded

6.3.2.4. GetLogFile

Syntax:	`public String GetLogFile()`
Description:	Get current log file name.
Parameter:	None
Return Value:	The current log file name.

6.3.2.5. GetLogLevel

Syntax:	`public int GetLogLevel()`
Description:	Get current log level.
Parameter:	None
Return Value:	The current log level.

6.3.2.6. GetLogFileAppend

Syntax:	`public bool GetLogFileAppend()`
Description:	Get Current LogFileAppend status.
Parameter:	None
Return Value:	`true`: Messages will be appended to existing file `false`: Existing file will be overwritten

7. Class: USB_Device

The class ‘USB_Device’ is an instance of a PrehKeyTec USB device and provides different properties, event handles and method to communicate with the device.

To create a new USB_Device, please use the ‘Open…’ methods of the PKTUSBAPI_Class as described above.

7.1. Properties

The class ‘USB_Device’ provides following properties:

Type	Name	Description
`String`	DevicePath	Get USB device path.
`ushort`	VendorID	Get USB vendor ID.
`ushort`	ProductID	Get USB product ID.
`ushort`	UsagePage	Get USB usage page.
`PKTUSBAPI_Class.DeviceInfo`	Information	Get the device information structure of the device.
`String`	VersionString	Get the complete PrehKeyTec version string.
`String`	Copyright	Get the PrehKeyTec copyright information.
`String`	FirmwareVersion	Get the PrehKeyTec firmware version.
`String`	FirmwareDate	Get the PrehKeyTec firmware date.
`String`	BootloaderVersion	Get the PrehKeyTec bootloader version.
`String`	ProductCode	Get the PrehKeyTec product code.
`String`	ProductNumber	Get the PrehKeyTec product number.
`String`	SerialNumber	Get the PrehKeyTec serial number.
bool	IsBootloaderActive	Is Devices bootloader active?
bool	MSRSentinels	De-/Activate receive MSR data include sentinels.
bool	MSRChecksum	De-/Activate receive MSR data include checksum.

7.2. Public Functions

The class ‘USB_Device provides following public methods:

Name	**Description**
`public void Close()`	Close the device and its background threads.
`public int WriteData(byte[] data)`	Write data to the device.
`public int KeytableWrite(String FileName)`	Download a keytable to keyboard.
`public int KeytableRead(String FileName)`	Upload a keytable from keyboard.
`public byte[] KeytableReadStart()`	Read the first 8 bytes from devices keytable.
`public int KeytableReadChecksum()`	Read the device keytable checksum.
`public int KeytableReadLength()`	Read the device keytable length.
`public int EnableKeyLock(bool Enabled)`	Activate/Deactivate the automatically output of the key lock position on position change.
`public int GetKeyLockPosition()`	Get the current key lock position.
`public byte[] KeytableReadStart()`	Read the first 8 bytes from devices keytable.
`public int KeytableReadChecksum()`	Read the device keytable checksum.
`public int KeytableReadLength()`	Read the device keytable length.
`public int EnableKeyLock(bool Enabled)`	Activate/Deactivate the automatically output of the key lock position on position change.
`public int GetKeyLockPosition()`	Get the current key lock position.
`public int EnableMSR(bool Enabled)`	Activate/Deactivate the automatically output of the MSR data.
`public EventArguments.OnReceivedMSR ReadMSR()`	Get the last or the next MSR data. (Only if MSR is disabled)
`public int ResetMSR()`	Reset the MSR data.
`public int SetFATFingerAlarm(bool Active)`	Activate/Deactivate the FAT finger alarm.
`public int SetEasyLayer(int LayerNumber)`	Set an easy layer.
`public int SetFNLayer(bool Active)`	Activate/Deactivate FN layer.
`public int ReadTCOCounter(int CounterNumber)`	Get TCO information.
`public String ReadTCOVersionString()`	Get the TCO information: VersionString
`public String ReadTCOFirmwareLevel()`	Get the TCO information: FirmwareLevel
`public String ReadTCOSerialNumber()`	Get the TCO information: SerialNumber
`public String ReadTCOManufactureDate()`	Get the TCO information: ManufactureDate
`public String ReadTCOProductCode()`	Get the TCO information: ProductCode
`public String ReadTCOProductNumber()`	Get the TCO information: ProductNumber
`public int InitSound(int Tone1_Frequency, int Tone2_Frequency, int Tone3_Frequency, int Tone1_Volume, int Tone2_Volume, int Tone3_Volume)`	Initialize the three tones of the device.
`public int StopTone()`	Stop playing a tone.
`public int PlayTone1_Endless()`	Play tone 1 endless.
`public int PlayTone1_Duration(int Duration)`	Play tone 1 with fix duration.
`public int PlayTone1_OnOffEndless()`	Play tone 1 endless with 100ms on off.
`public int PlayTone2_Endless()`	Play tone 2 endless.
`public int PlayTone2_Duration(int Duration)`	Play tone 2 with fix duration.
`public int PlayTone2_OnOffEndless()`	Play tone 2 endless with 100ms on off.
`public int PlayTone3_Endless()`	Play tone 3 endless.
`public int PlayTone3_Duration(int Duration)`	Play tone 3 with fix duration.
`public int PlayTone3_OnOffEndless()`	Play tone 3 endless with 100ms on off.
Name	**Description**

`public int PlayBeep()`	Play 'Beep' tone.
`public int PlayTone(int Tone, int Volume)`	Play an example tone.
`public int SetLEDAcceptOnly(int AcceptColor, int AcceptFlash)`	Set only the device accept LED. Caps, Num and Scroll LED will stay synchronous to normal keyboard state.
`public int SetLEDs(bool NumLock, bool CapsLock, bool ScrollLock, int AcceptColor, int AcceptFlash)`	Set the device LEDs.
`public int ResetLEDs()`	Reset the device LEDs.
`public int GetBacklightLevel()`	Get current backlight level.
`public int GetBacklightTimeout()`	Get current backlight timeout.
`public int SetBacklightLevelHigher()`	Increase backlight level.
`public int SetBacklightLevelLower()`	Decrease backlight level.
`public int SetBacklightLevel(int Level)`	Set backlight level.
`public int SetBacklightLevelTemporary(int Level)`	Set backlight level temporary.
`public int SetBacklightTimeoutHigher()`	Increase backlight timeout.
`public int SetBacklightTimeoutLower()`	Decrease backlight timeout.
`public int SetBacklightTimeout(int Timeout)`	Set backlight timeout.
`public int CaptureImage()`	Take a snapshot. (Only PKT4000)
`public int EnableAllBarcodeTypes(bool Enable)`	Enable/Disable all barcode types.
`public int EnableBarcodeType(int Type, bool Enable)`	Enable/Disable scanning a barcode type.
`public int SetScannerEnabled(bool enable)`	De-/Activate Scanner Engine.
`public int ClearLCDText()`	Clear text on LCD display.
`public int SetLCDTextLine1(String text)`	Set text line 1 on LCD display.
`public int SetLCDTextLine2(String text)`	Set text line 2 on LCD display.
`public int SetLCDText(String line1, String line2)`	Set text on LCD display
`public int SetLCDTextAt(String text, int position)`	Set LCD text at position.
`public int SetLCDContrast(int value)`	Set LCD Display contrast.
`public int SetLCDBrightness(int value)`	Set LCD Display brightness.
`public int SendLCDCommand(byte[] cmd)`	Send a LCD Display command.

7.2.1. Common Device Methods

7.2.1.1. Close

Syntax:	`public void Close()`
Description:	Close the device.
Parameter:	None
Returns:	None

7.2.1.2. WriteData

Syntax:	`public int WriteData(byte[] data)`
Description:	Write data to the device.
Parameter:	`byte[] data`: Data byte array to write.
Returns:	Number of written devices or `ErrorStatus.WriteFailed`

7.2.1.3. SetFATFingerAlarm

Syntax:	`public int SetFATFingerAlarm(bool Active)`
Description:	Activate/Deactivate the FAT finger alarm.
Parameter:	`bool Active`: Activate FAT finger alarm.
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.1.4. SetEasyLayer

Syntax:	`public int SetEasyLayer(int LayerNumber)`
Description:	Set an easy layer.
Parameter:	`int LayerNumber`: Number of the layer. (0 - 127)
Returns:	`ErrorStatus.Succeeded`, `ErrorStatus.InvalidValue` or `ErrorStatus.WriteFailed`

7.2.1.5. SetFNLayer

Syntax:	`public int SetFNLayer(bool Active)`
Description:	Activate/Deactivate FN layer.
Parameter:	`bool Active`: Activate FN Layer
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.2. File Transfer Methods

7.2.2.1. KeytableWrite

Syntax:	`public int KeytableWrite(String FileName)`
Description:	Download a keytable to keyboard.
Parameter:	`String FileName`: Keytable file (*.mwx) to download.
Returns:	Keytable checksum on success, or `ErrorStatus` if failed.

7.2.2.2. KeytableRead

Syntax:	`public int KeytableRead(String FileName)`
Description:	Upload a keytable from keyboard.
Parameter:	`String FileName`: The file name to store the uploaded keytable.
Returns:	Keytable checksum on success, or `ErrorStatus` if failed.

7.2.2.3. KeytableReadStart

Syntax:	`public byte[] KeytableReadStart()`
Description:	Read the first 8 bytes from devices keytable.
Parameter:	None
Returns:	The first 8 bytes of device keytable, or `null` if failed.

7.2.2.4.KeytableReadChecksum

Syntax:	`public int KeytableReadChecksum()`
Description:	Read the device keytable checksum.
Parameter:	None
Returns:	The device keytable checksum, `ErrorStatus.InvalidDevice`, or `ErrorStatus.Failed`

7.2.2.5.KeytableReadLength

Syntax:	`public int KeytableReadLength()`
Description:	Read the length of device keytable.
Parameter:	None
Returns:	The device keytable length, `ErrorStatus.InvalidDevice`, or `ErrorStatus.Failed`

7.2.3. KeyLock Methods

7.2.3.1.EnableKeyLock

Syntax:	`public int EnableKeyLock(bool Enabled)`
Description:	Activate/Deactivate the automatically output of the key lock position on position change.
Parameter:	`bool Enabled`: Activate key lock
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.3.2.GetKeyLockPosition

Syntax:	`public int GetKeyLockPosition()`
Description:	Get the current key lock position.
Parameter:	None
Returns:	Current key lock position or `ErrorStatus.WriteFailed`

7.2.4. MSR Methods

7.2.4.1.EnableMSR

Syntax:	`public int EnableMSR(bool Enabled)`
Description:	Activate/Deactivate the automatically output of the MSR data.
Parameter:	`bool Enabled`: Activate MSR
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.4.2. ReadMSR

Syntax:	`public EventArguments.OnReceivedMSR ReadMSR()`
Description:	Get the last or the next MSR data. (Only if MSR is disabled)
Parameter:	None
Returns:	The MSR data or `null` if failed.

7.2.4.3. ResetMSR

Syntax:	`public int ResetMSR()`
Description:	Reset the MSR data.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.5. TCO Counter Methods

7.2.5.1. ReadTCOCounter

Syntax:	`public int ReadTCOCounter(int CounterNumber)`
Description:	Read out TCO counter.
Parameter:	`int CounterNumber`: Number of the TCO Counter.
Returns:	Counter value as string.

7.2.5.2. ReadTCOVersionString

Syntax:	`public String ReadTCOVersionString()`
Description:	Get the TCO information: VersionString
Parameter:	None
Returns:	Counter value as string.

7.2.5.3. ReadTCOFirmwareLevel

Syntax:	`public String ReadTCOFirmwareLevel()`
Description:	Get the TCO information: FirmwareLevel
Parameter:	None
Returns:	Counter value as string.

7.2.5.4. ReadTCOSerialNumber

Syntax:	`public String ReadTCOSerialNumber()`
Description:	Get the TCO information: SerialNumber
Parameter:	None
Returns:	Counter value as string.

7.2.5.5. ReadTCOManufactureDate

Syntax:	`public String ReadTCOManufactureDate()`
Description:	Get the TCO information: ManufactureDate
Parameter:	None
Returns:	Counter value as string.

7.2.5.6. ReadTCOProductCode

Syntax:	`public String ReadTCOProductCode()`
Description:	Get the TCO information: ProductCode
Parameter:	None
Returns:	Counter value as string.

7.2.5.7. ReadTCOProductNumber

Syntax:	`public String ReadTCOProductNumber()`
Description:	Get the TCO information: ProductNumber
Parameter:	None
Returns:	Counter value as string.

7.2.6. Sound Methods

7.2.6.1. InitSound

Syntax:	`public int InitSound(int Tone1_Frequency, int Tone2_Frequency, int Tone3_Frequency, int Tone1_Volume, int Tone2_Volume, int Tone3_Volume)`
Description:	Initialize the three tones of the device.
Parameter:	`int Tone1_Frequency`: Frequency of the tone 1. (265 - 5000) `int Tone2_Frequency`: Frequency of the tone 2. (265 - 5000) `int Tone3_Frequency`: Frequency of the tone 3. (265 - 5000) `int Tone1_Volume`: Volume of the tone 1. (0 - 100) `int Tone2_Volume`: Volume of the tone 2. (0 - 100) `int Tone3_Volume`: Volume of the tone 3. (0 - 100)
Returns:	`ErrorStatus.Succeeded`, `ErrorStatus.InvalidValue` or `ErrorStatus.WriteFailed`

7.2.6.2. StopTone

Syntax:	`public int StopTone()`
Description:	Stop playing a tone.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.3. PlayTone1_Endless

Syntax:	`public int PlayTone1_Endless()`
Description:	Play tone 1 endless.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.4. PlayTone1_Duration

Syntax:	`public int PlayTone1_Duration(int Duration)`
Description:	Play tone 1 with fix duration.
Parameter:	`int Duration`: Duration of the tone.
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.5. PlayTone1_OnOffEndless

Syntax:	`public int PlayTone1_OnOffEndless()`
Description:	Play tone 1 endless with 100ms on off.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.6. PlayTone2_Endless

Syntax:	`public int PlayTone2_Endless()`
Description:	Play tone 2 endless.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.7. PlayTone2_Duration

Syntax:	`public int PlayTone2_Duration(int Duration)`
Description:	Play tone 2 with fix duration.
Parameter:	`int Duration`: Duration of the tone.
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.8. PlayTone2_OnOffEndless

Syntax:	`public int PlayTone2_OnOffEndless()`
Description:	Play tone 2 endless with 100ms on off.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.9. PlayTone3_Endless

Syntax:	`public int PlayTone3_Endless()`
Description:	Play tone 3 endless.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.10. PlayTone3_Duration

Syntax:	`public int PlayTone3_Duration(int Duration)`
Description:	Play tone 3 with fix duration.
Parameter:	`int Duration`: Duration of the tone.
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.11. PlayTone3_OnOffEndless

Syntax:	`public int PlayTone3_OnOffEndless()`
Description:	Play tone 3 endless with 100ms on off.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.12. PlayBeep

Syntax:	`public int PlayBeep()`
Description:	Play 'Beep' tone.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.6.13. PlayBeep

Syntax:	`public int PlayTone(int Tone, int Volume)`
Description:	Play an example tone.
Parameter:	`int Tone`: Example tone (Values see chapter ‘Class Constants’) `int Volume`: Volume of the tone. (0 - 100)
Returns:	`ErrorStatus.Succeeded`, `ErrorStatus.InvalidValue` or `ErrorStatus.WriteFailed`

7.2.7. LED Methods

7.2.7.1. SetLEDAcceptOnly

Syntax:	`public int SetLEDAcceptOnly(int AcceptColor, int AcceptFlash)`
Description:	Set only the device accept LED. Caps, Num and Scroll LED will stay synchronous to normal keyboard state.
Parameter:	`int Accept`: Color of the accept LED (Values see chapter 'Class Constants') `int AcceptFlash`: Flash speed of the accept LED (Values see chapter 'Class Constants')
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.7.2. SetLEDs

Syntax:	`public int SetLEDs(bool NumLock, bool CapsLock, bool ScrollLock, int AcceptColor, int AcceptFlash)`
Description:	Set the device LEDs.
Parameter:	`bool NumLock`: Switch on num lock LED `bool CapsLock`: Switch on caps lock LED `bool ScrollLock`: Switch on scroll lock LED `int AcceptColor`: Color of the accept LED (Values see chapter 'Class Constants') `int AcceptFlash`: Flash speed of the accept LED (Values see chapter 'Class Constants')
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.7.3. ResetLEDs

Syntax:	int ResetLEDs()
Description:	Reset the device LEDs.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8. Backlight Methods

7.2.8.1. GetBacklightLevel

Syntax:	`public int GetBacklightLevel()`
Description:	Get current backlight level.
Parameter:	None
Returns:	The current backlight level or `ErrorStatus.WriteFailed`

7.2.8.2. GetBacklightTimeout

Syntax:	`public int GetBacklightTimeout()`
Description:	Get current backlight timeout.
Parameter:	None
Returns:	The current backlight timeout or `ErrorStatus.WriteFailed`

7.2.8.3. SetBacklightLevelHigher

Syntax:	`public int SetBacklightLevelHigher()`
Description:	Increase backlight level.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.4. SetBacklightLevelLower

Syntax:	`public int SetBacklightLevelLower()`
Description:	Decrease backlight level.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.5. SetBacklightLevel

Syntax:	`public int SetBacklightLevel(int Level)`
Description:	Set backlight level.
Parameter:	`int Level:` The new backlight level
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.6. SetBacklightLevelTemporary

Syntax:	`public int SetBacklightLevelTemporary(int Level)`
Description:	Set backlight level temporary.
Parameter:	`int Level:` The new temporary backlight level
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.7. SetBacklightTimeoutHigher

Syntax:	`public int SetBacklightTimeoutHigher()`
Description:	Increase backlight timeout.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.8. SetBacklightTimeoutLower

Syntax:	`public int SetBacklightTimeoutLower()`
Description:	Decrease backlight timeout.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.8.9. SetBacklightTimeout

Syntax:	`public int SetBacklightTimeout(int Timeout)`
Description:	Set backlight timeout.
Parameter:	`int Timeout:` The new backlight timeout
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.9. Barcode/PKT4000 Methods

7.2.10. LCD Display Methods

7.2.10.1. ClearLCDText

Syntax:	`public int ClearLCDText()`
Description:	Clear displayed text on LCD display.
Parameter:	None
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.2. SetLCDTextLine1

Syntax:	`public int SetLCDTextLine1(String text)`
Description:	Set text line 1 on LCD Display
Parameter:	`String text:` Text to display in line 1
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.3. SetLCDTextLine1

Syntax:	`public int SetLCDTextLine1(byte[] data)`
Description:	Set text line 1 on LCD Display
Parameter:	`byte[] data:` Data to display in line 1
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.4. SetLCDTextLine2

Syntax:	`public int SetLCDTextLine2(String text)`
Description:	Set text line 2 on LCD Display
Parameter:	`String text:` Text to display in line 2
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.5. SetLCDTextLine2

Syntax:	`public int SetLCDTextLine2(byte[] data)`
Description:	Set text line 2 on LCD Display
Parameter:	`byte[] data:` Data to display in line 2
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.6. SetLCDText

Syntax:	`public int SetLCDText(String line1, String line2)`
Description:	Set text on LCD Display
Parameter:	`String line1:` Text to display in line 1 `String line2:` Text to display in line 2
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.7. SetLCDText

Syntax:	`public int SetLCDText(byte[] line1, byte[] line2)`
Description:	Set text on LCD Display
Parameter:	`byte[] line1:` Data to display in line 1 `byte[] line2:` Data to display in line 2
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.8. SetLCDTextAt

Syntax:	`public int SetLCDTextAt(String text, int position)`
Description:	Set LCD text at position
Parameter:	`String text:` Text to display `int position:` Position to display the text
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.9. SetLCDTextAt

Syntax:	`public int SetLCDTextAt(byte[] text, int position)`
Description:	Set LCD text at position
Parameter:	`byte[] text:` Text to display `int position:` Position to display the text
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.10. SetLCDContrast

Syntax:	`public int SetLCDContrast(int value)`
Description:	Set LCD display contrast
Parameter:	`int value:` Contrast value (1 – 50, Default: 40)
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.11. SetLCDBrightness

Syntax:	`public int SetLCDBrightness(int value)`
Description:	Set LCD display brightness
Parameter:	`int value:` Brightness value (1 – 8, Default: 8)
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

7.2.10.12. SendLCDCommand

Syntax:	`public int SendLCDCommand(byte[] cmd)`
Description:	Send a command to LCD display
Parameter:	`byte[] cmd:` Command bytes to send
Returns:	`ErrorStatus.Succeeded` or `ErrorStatus.WriteFailed`

8. Class 'Constants'

In this class there are some constant values that are used as parameter for some methods above.

Type	Name	Description
`int`	LED_OFF	USB_Device.SetLEDs(): Switch off accept LED
`int`	LED_GREEN	USB_Device.SetLEDs(): Switch on green accept LED
`int`	LED_RED	USB_Device.SetLEDs(): Switch on red accept LED
`int`	LED_ORANGE	USB_Device.SetLEDs(): Switch on orange accept LED
`int`	LED_FLASH_OFF	USB_Device.SetLEDs(): Switch off flashing accept LED
`int`	LED_FLASH_400ms	USB_Device.SetLEDs(): Set flashing speed of accept LED to 400ms
`int`	LED_FLASH_600ms	USB_Device.SetLEDs(): Set flashing speed of accept LED to 600ms
`int`	LED_FLASH_800ms	USB_Device.SetLEDs(): Set flashing speed of accept LED to 800ms
`int`	TONE_ACCEPTANCE	USB_Device.PlayTone(): Set example tone 'Acceptance'
`int`	TONE_ATTENTION	USB_Device.PlayTone(): Set example tone 'Attention'
`int`	TONE_CALL	USB_Device.PlayTone(): Set example tone 'Call'
`int`	TONE_ERROR	USB_Device.PlayTone(): Set example tone 'Error'
`ushort`	LOG_ALWAYS	PKTUSBAPI.WriteLog(): Message Log Level 'Always'.
`ushort`	LOG_ERROR	PKTUSBAPI.WriteLog(): Message Log Level 'Error'.
`ushort`	LOG_WARNING	PKTUSBAPI.WriteLog(): Message Log Level 'Warning'.
`ushort`	LOG_INFO	PKTUSBAPI.WriteLog(): Message Log Level 'Info'.
`ushort`	LOG_DATA	PKTUSBAPI.WriteLog(): Message Log Level 'Data'.
`ushort`	LOG_DEBUG	PKTUSBAPI.WriteLog(): Message Log Level 'Debug'.
`ushort`	LOG_LEVEL_ERROR	PKTUSBAPI.StartLog(): Log only errors.
`ushort`	LOG_LEVEL_ERROR_WARNING	PKTUSBAPI.StartLog(): Log only errors and warnings.
`ushort`	LOG_LEVEL_ERROR_WARNING_INFO	PKTUSBAPI.StartLog(): Log only errors, warnings and info.
`ushort`	LOG_LEVEL_ERROR_WARNING_INFO_DATA	PKTUSBAPI.StartLog(): Log only errors, warnings, info and data.
`ushort`	LOG_LEVEL_ALL	PKTUSBAPI.StartLog(): Log all messages.
`int`	BACKLIGHT_LEVEL0	Backlight level 0.
`int`	BACKLIGHT_LEVEL1	Backlight level 1.
`int`	BACKLIGHT_LEVEL2	Backlight level 2.
`int`	BACKLIGHT_LEVEL3	Backlight level 3.
`int`	BACKLIGHT_LEVEL4	Backlight level 4.
`int`	BACKLIGHT_LEVEL5	Backlight level 5.
`int`	BACKLIGHT_TIMEOUT_NO	Backlight Timeout always on.
`int`	BACKLIGHT_TIMEOUT_1MIN	Backlight Timeout 1 minutes.
`int`	BACKLIGHT_TIMEOUT_5MIN	Backlight Timeout 5 minutes.
`int`	BACKLIGHT_TIMEOUT_10MIN	Backlight Timeout 10 minutes.
`int`	BACKLIGHT_TIMEOUT_30MIN	Backlight Timeout 30 minutes.
`int`	BACKLIGHT_TIMEOUT_1H	Backlight Timeout 1 hour.
`int`	BACKLIGHT_TIMEOUT_4H	Backlight Timeout 4 hours.

9. Class 'ErrorStatus'

In this class the methods return values are defined.

Type	Name	Description	Value (dec)
`int`	Succeeded	Method was finished successful.	0
`int`	InvalidValue	Method was called with an invalid parameter.	-1
`int`	WriteFailed	Write data to device failed.	-2
`int`	ReadFailed	Read data from device failed.	-3
`int`	DLLAccessFailed	Unable to access the native C++ DLL.	-4
`int`	Failed	Common or unknown fault.	-5
`int`	InvalidDevice	Wrong device.	-6
`int`	DeviceClaimed	Device is claimed.	-7
`int`	Keytable_InvalidFileSize	Invalid keytable file size.	-10
`int`	Keytable_ReadFailed	Reading keytable from device failed.	-11
`int`	Keytable_InvalidChecksum	Invalid keytable checksum.	-12
`int`	Keytable_Failed	Unknown keytable error.	-13
`int`	Keytable_InvalidFile	Invalid keytable file.	-14
`int`	Keytable_InvalidFileExtension	Invalid keytable file extension.	-15
`int`	Keytable_ReadFileFailed	Reading keytable file failed.	-16
`int`	Keytable_WriteFailed	Writing keytable to device failed.	-17
`int`	Bootloader_Failed	Unknown bootloader error.	-20
`int`	Bootloader_InvalidFile	Invalid file.	-21
`int`	Bootloader_InvalidFilePID	Invalid file for the device.	-22
`int`	Bootloader_ReadFileFailed	Reading file failed.	-23
`int`	Bootloader_WriteFailed	Writing to device failed.	-24
`int`	Bootloader_BootloaderNotFound	No bootloader device found.	-25
`int`	Bootloader_InvalidFileExtension	Invalid file extension.	-26
`int`	Bootloader_ReadFailed	Reading from device failed.	-27
`int`	Bootloader_InvalidChecksum	Invalid checksum.	-28

10. Example Applications

To help you use the PKTUSBAPI in your own application we provide some example application with different usages of the PKTUSBAPI. In the chapters below the examples are explained.

10.1. PKTUSBAPI Console Demo

This example is a simple console application. In this application the first found device will be opened. If a device was found and opened the devices version string is shown. Furthermore the 'Error' tone is played and some LEDs are changed to identify the opened device.
Then the event handlers to receive MSR data, POS keys and the key lock position are set up.

This example is a console application that monitors incoming MSR, POS key and key lock data.

10.1.1. Source Code 'Programm.cs'

```
#define MY_OCR_DEVICE // You can uncomment this Definition for non OCR devices

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.ComponentModel;

//Reference to PKTUSBAPI.DLL
using PKTUSBAPI;


namespace PKTUSBAPI_Console_Demo
{

    class Program
    {
        private static PKTUSBAPI_Class myAPI = null;
        private static USB_Device myDevice = null;

#if MY_OCR_DEVICE
        private static USB_Device myOCRDevice = null;
#endif

        static void Main(string[] args)
        {
            try
            {
                //Initialize the PKTUSBAPI (Show all existing PrehKeyTec USB Devices)
                Console.WriteLine("\nInitialize the PKTUSBAPI");
                myAPI = new PKTUSBAPI_Class(null);

                Console.WriteLine("PKTUSBAPI.dll Version: " + myAPI.VersionAPI);
                Console.WriteLine("PKTUSBxx.dll Version: " + myAPI.VersionDLL);

                //Name of my log file (text file)
                String LogFileName = "MyLogFile.txt";




                //Log only errors, warnings and inforamtion
                int LogLevel = Constants.LOG_LEVEL_ERROR_WARNING_INFO;
                //Append new messages to the log file if it already exists
                bool LogAppend = true;

                //Activate log file
                int ret = myAPI.StartLog(LogFileName, LogAppend, LogLevel);
                if (ret == ErrorStatus.Succeeded)
                {
                    Console.WriteLine("Logging successful started");
                }
                else
                {
                    Console.WriteLine("Starting logging failed");
                }

                // Open already connected device
                OpenDevice();

#if MY_OCR_DEVICE
                // Open already connected OCR device
                OpenDeviceOCR();
#endif
                //Set up on device status changed event
                Console.WriteLine("\nSet up Event Handler: OnDeviceStatusChanged");
                myAPI.OnDeviceStatusChanged += new
EventHandler<EventArguments.OnDeviceStatusChanged>(myAPI_OnDeviceStatusChanged);

                while (true)
                    Thread.Sleep(5);     // wait forever limiting cpu resources
            }
            catch { }
        }

        private static void OpenDevice()
        {
            //Open the device that was found first from the API with the usage page of 0xfffa
            myDevice = myAPI.OpenDevice(0xfffa);
            if (myDevice != null)
            {
                //Display some device information
                Console.WriteLine("\n+  +  +  +  +  +  +  +  +  +  +  +  +  +");
                Console.WriteLine("Device opened:");
                Console.WriteLine(myDevice.DevicePath + "\n");
                Console.WriteLine(myDevice.VersionString);

                //Play 'Beep' tone on it
                myDevice.PlayBeep();

                //Set up on received MSR data event for my device
                Console.WriteLine("\nSet up Event Handler: OnReceivedMSR");
                myDevice.OnReceivedMSR += new
EventHandler<EventArguments.OnReceivedMSR>(myDevice_OnReceivedMSR);

                //Set up on received POS key event for my device
                Console.WriteLine("Set up Event Handler: OnReceivedPOSkey");
                myDevice.OnReceivedPOSkey += new
EventHandler<EventArguments.OnReceivedPOSkey>(myDevice_OnReceivedPOSkey);


                //Set up on received key lock changed event for my device
                Console.WriteLine("Set up Event Handler: OnReceivedKeyLockChanged");
                myDevice.OnKeyLockChanged += new
EventHandler<EventArguments.OnKeyLockChanged>(myDevice_OnKeyLockChanged);

                Console.WriteLine("+  +  +  +  +  +  +  +  +  +  +  +  +  +");
            }
        }

#if MY_OCR_DEVICE
        private static void OpenDeviceOCR()
        {
            //Open the device that was found first from the API with the usage page of 0xfff9 (OCR)
            myOCRDevice = myAPI.OpenDevice(0xfff9);
            if (myOCRDevice != null)
            {
                //Display some device information
                Console.WriteLine("\n+  +  +  +  +  +  +  +  +  +  +  +  +  +");
                Console.WriteLine("OCR Device opened:");
                Console.WriteLine(myOCRDevice.DevicePath + "\n");
                Console.WriteLine(myOCRDevice.VersionString);

                //Set up on received MSR data event for my device
                Console.WriteLine("\nSet up Event Handler: OnReceivedMSR");
                myOCRDevice.OnReceivedMSR += new
EventHandler<EventArguments.OnReceivedMSR>(myDevice_OnReceivedMSR);

                //Set up on received OCR data event for my device
                Console.WriteLine("Set up Event Handler: OnReceivedOCR");
                myOCRDevice.OnReceivedOCR += new
EventHandler<EventArguments.OnReceivedOCR>(myDevice_OnReceivedOCR);

                Console.WriteLine("+  +  +  +  +  +  +  +  +  +  +  +  +  +");
            }
        }
#endif

        static void myAPI_OnDeviceStatusChanged(object sender, EventArguments.OnDeviceStatusChanged e)
        {
            //Do something only if a device was attached or detached
            if (e.ConnectedStatusChanged)
            {
                //Check the usage page of the de-/attached device to see if it is a PrehKeyTec standard
device
                if (e.ChangedDevice.UsagePage == 0xfffa)
                {
                    //Check if myDevice was detached
                    if (myDevice != null && e.ChangedDevice.DevicePath.Equals(myDevice.DevicePath) &&
!e.ChangedDevice.IsConnected)
                    {
                        //Close myDevice instance
                        Console.WriteLine("\nDevice closed");
                        myDevice.Close();
                        myDevice = null;
                    }



                    //Searching for a device to open if myDevice is 'null'
                    if (myDevice == null)
                    {
                        OpenDevice();
                    }
                }


#if MY_OCR_DEVICE
                //Check the usage page of the de-/attached device to see if it is a PrehKeyTec OCR
device
                if (e.ChangedDevice.UsagePage == 0xfff9)
                {
                    //Check if myOCRDevice was detached
```

```
                        if (myOCRDevice != null &&
    e.ChangedDevice.DevicePath.Equals(myOCRDevice.DevicePath) && !e.ChangedDevice.IsConnected)
                        {
                            //Close myOCRDevice instance
                            Console.WriteLine("\nOCR Device closed");
                            myOCRDevice.Close();
                            myOCRDevice = null;
                        }

                        //Searching for an OCR device to open if myOCRDevice is 'null'
                        if (myOCRDevice == null)
                        {
                            OpenDeviceOCR();
                        }
                    }
    #endif
                }
            }

            static void myDevice_OnKeyLockChanged(object sender, EventArguments.OnKeyLockChanged e)
            {
                //Show received event data
                Console.WriteLine("\n[KeyLock]");
                Console.WriteLine("Position: " + e.Position);
            }

            static void myDevice_OnReceivedPOSkey(object sender, EventArguments.OnReceivedPOSkey e)
            {
                //Show received event data
                Console.WriteLine("\n[POS Key]");
                Console.WriteLine("Key: " + e.Key);
                Console.WriteLine("Status: " + (e.Pressed ? "pressed" : "released"));
            }

            static void myDevice_OnReceivedMSR(object sender, EventArguments.OnReceivedMSR e)
            {
                //Show received event data
                Console.WriteLine("\n[MSR]");
                Console.WriteLine("Track 1: " + e.MSRtrack1);
                Console.WriteLine("Track 2: " + e.MSRtrack2);
                Console.WriteLine("Track 3: " + e.MSRtrack3);
            }



    #if MY_OCR_DEVICE
            static void myDevice_OnReceivedOCR(object sender, EventArguments.OnReceivedOCR e)
            {
                //Show received event data
                Console.WriteLine("\n[OCR]");
                Console.WriteLine("MRZ line 1: " + e.MRZline1);
                Console.WriteLine("MRZ line 2: " + e.MRZline2);
                Console.WriteLine("MRZ line 3: " + e.MRZline3);
            }
    #endif
        }
    }
```

10.2. PKTUSBAPI Concentrater Demo

This example is a simple console application. In this application all connected devices are opened with the concentrate mode of the API.

This example is a console application that monitors every incoming data of all connected devices.

10.2.1. Source Code 'Programm.cs'

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using PKTUSBAPI;

namespace PKTUSBAPI_Concentrater_Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("\nInitialize the PKTUSBAPI");
                //Initialize the PKTUSBAPI (Show all existing PrehKeyTec USB Devices)
                PKTUSBAPI_Class myAPI = new PKTUSBAPI_Class(null, true);
                //Show the library versions
                Console.WriteLine("PKTUSBAPI.dll Version: " + myAPI.VersionAPI);
                Console.WriteLine("PKTUSBxx.dll Version: " + myAPI.VersionDLL);
                Console.WriteLine("\nSet up Event Handler: OnReceivedCommonData");
                //Set up on received data event for all devices
                myAPI.OnReceivedCommonData += new 
EventHandler<EventArguments.OnReceivedCommonData>(myAPI_OnReceivedCommonData);
                Console.WriteLine("\nSet up Event Handler: OnReceivedCommonOCR");
                //Set up on received OCR data event for all devices
                myAPI.OnReceivedCommonOCR += new 
EventHandler<EventArguments.OnReceivedCommonOCR>(myAPI_OnReceivedCommonOCR);
                Console.WriteLine("\nSet up Event Handler: OnReceivedCommonMSR");
                //Set up on received MSR data event for all devices
                myAPI.OnReceivedCommonMSR += new 
EventHandler<EventArguments.OnReceivedCommonMSR>(myAPI_OnReceivedCommonMSR);
                Console.WriteLine("\nSet up Event Handler: OnReceivedCommonAUX");
                //Set up on received AUX data event for all devices
                myAPI.OnReceivedCommonAUX += new 
EventHandler<EventArguments.OnReceivedCommonAUX>(myAPI_OnReceivedCommonAUX);
                Console.WriteLine("Set up Event Handler: OnReceivedCommonPOSkey");
                //Set up on received POS key event for all devices
                myAPI.OnReceivedCommonPOSkey += new 
EventHandler<EventArguments.OnReceivedCommonPOSkey>(myAPI_OnReceivedCommonPOSkey);

                Console.WriteLine("Set up Event Handler: OnCommonKeyLockChanged");
                //Set up on received key lock changed event for all devices
                myAPI.OnCommonKeyLockChanged += new 
EventHandler<EventArguments.OnCommonKeyLockChanged>(myAPI_OnCommonKeyLockChanged);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        static void myAPI_OnReceivedCommonData(object sender, EventArguments.OnReceivedCommonData e)
        {
            Console.WriteLine("\n" + e.Data.RawData.Length + " Bytes received from {" + e.Device.SerialNumber + 
"}");
        }

        static void myAPI_OnCommonKeyLockChanged(object sender, EventArguments.OnCommonKeyLockChanged e)
        {
            Console.WriteLine("\n[KeyLock] {" + e.Device.SerialNumber + "}");
                    Console.WriteLine("Position: " + e.Data.Position);
        }

        static void myAPI_OnReceivedCommonPOSkey(object sender, EventArguments.OnReceivedCommonPOSkey e)
        {
            Console.WriteLine("\n[POS-Key] {" + e.Device.SerialNumber + "}");
                    Console.WriteLine("Key:    " + e.Data.Key);
                    Console.WriteLine("Status: " + (e.Data.Pressed ? "pressed" : "released"));
        }

        static void myAPI_OnReceivedCommonMSR(object sender, EventArguments.OnReceivedCommonMSR e)
        {
            Console.WriteLine("\n[MSR] {" + e.Device.SerialNumber + "}");
                    Console.WriteLine("Track 1: " + e.Data.MSRtrack1);
            Console.WriteLine("Track 2: " + e.Data.MSRtrack2);
            Console.WriteLine("Track 3: " + e.Data.MSRtrack3);
        }

        static void myAPI_OnReceivedCommonOCR(object sender, EventArguments.OnReceivedCommonOCR e)
        {
            Console.WriteLine("\n[OCR] {" + e.Device.SerialNumber + "}");
                    Console.WriteLine("MRZ line 1: " + e.Data.MRZline1);
            Console.WriteLine("MRZ line 2: " + e.Data.MRZline2);
            Console.WriteLine("MRZ line 3: " + e.Data.MRZline3);
        }
        static void myAPI_OnReceivedCommonAUX(object sender, EventArguments.OnReceivedCommonAUX e)
        {
            Console.WriteLine("\n[AUX] {" + e.Device.SerialNumber + "}");
                    Console.WriteLine("Data: " + e.Data.auxData.AUX_String);
        }

    }
}
```

11. Support

PrehKeyTec GmbH Scheinbergweg 10 97638 Mellrichstadt Germany Phone: +49 9776 7046 0 Support: +49 9776 7046 222 Fax: +49 9776 7046 199 Email: support@prehkeytec.de