Bridge PC/SC over BLE or TCP

Principles

Windows' driver architecture has no provision for BLE devices, that are expected to be handled by applications running in user land only.

To be able to use a BLE device as a PC/SC reader, SpringCard has developed an 'abstract' PC/SC and CCID-compliant device driver, that exposes itself to Windows as being a smartcard reader, yet does nothing more than sending CCID frames to a local communication pipe. At the other end of the communication pipe, a so-called "bridge" application runs in user land and is responsible for the actual communication with the BLE device.

This document is the specification of a new library, `SpringCard.SpringCore.BleBridge.dll`, that has three roles:

1. manage the driver,
2. implement the core feature of bridging the driver to the BLE device,
3. make provisions for controlling the application(s) to card(s) workflow, should it be required.

This library is written in C# and targets .NET framework v4.6.2. For compatibility reasons, the namespace of the library is `SpringCard.PCSC.Bridge`.

The library is accompanied by two executables:

- `SpringCoreBleBridge.exe` is a tool that allows to run most functions and methods of the library from the command-line,
- `SpringCoreBleBridgeTray.exe` is a windowed application that does the same but sits in the tray bar.

Static methods

Install the driver

Condition: must be elevated. Instance number is defined by Windows.

Synopsis

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Install the driver */
        public static bool DriverInstall(string DeviceName, string[] SlotNames, out
uint InstanceNumber, out uint ErrorReason);
    }
}
```

C# Code

```
Console.WriteLine("Installing the BLE PC/SC driver, please wait...");
if (!SpringCard.PCSC.Bridge.BLE.DriverInstall("AF Care Two/BT", new string[] { "Carte
vitale", "CPS" }, out uint InstanceNumber, out uint ErrorReason))
{
    Console.WriteLine("Failed to install the BLE PC/SC driver, error {0}",
ErrorReason);
}
else
{
    Console.WriteLine("Success, driver instance is {0}", InstanceNumber );
}
```

Error Reason

- `2 : Device Name is null or empty`
- `2 : Slot name is null or empty`
- `4 : Fails to install driver`
- `4 : Fails to create driver`
- `4 : No Driver ID found`

Command line

```
SpringCoreBleBridge.exe driver-install --device-name="AF Care Two/BT" --slot-
names="Carte vitale|CPS"
```

Remove the driver

Condition: must be elevated.

Synopsis

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Uninstall the driver */
        public static bool DriverUninstall(uint InstanceNumber, out uint ErrorReason);
    }
}
```

C# code

```
Console.WriteLine("Removing the BLE PC/SC driver, please wait...");
if (!SpringCard.PCSC.Bridge.BLE.DriverUninstall(0, out uint ErrorReason))
{
    Console.WriteLine("Failed to remove the BLE PC/SC driver, error {0}",
ErrorReason);
}
else
{
    Console.WriteLine("Success");
}
```

Error Reason

- `5 : unable to access registry`
- `4 : This driver instance does not exist`
- `4 : Fails to create driver`

Command line

```
SpringCoreBleBridge.exe driver-uninstall [--instance=0]
```

Check whether the driver is installed

Synopsis

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Check that the driver is installed */
        public static bool IsDriverInstalled(uint InstanceNumber, out string
DeviceName, out string[] SlotNames);
    }
}
```

C# Code

```
if (!SpringCard.PCSC.Bridge.BLE.IsDriverInstalled(0, out string DeviceName, out
string[] SlotNames))
{
    Console.WriteLine("No driver installed at instance number=0");
}
else
{
    Console.WriteLine("Driver is installed at instance number=0");
    Console.WriteLine("\tDeviceName={0}", DeviceName);
    foreach (string slotName in SlotNames)
        Console.WriteLine("\tSlot={0}", slotName);
}
```

Command line

```
SpringCoreBleBridge.exe driver-check [--instance=0]
```

Look for BLE devices in the nearby

Synopsis

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Scan for (compliant) BLE devices */
        public static bool Scan(uint DurationMs, out SpringCard.Bluetooth.Device[]
FoundDevices, out uint ErrorReason);
    }
}
```

C# code

```
Console.WriteLine("Looking for BLE devices, please wait 3s...");
if (!SpringCard.PCSC.Bridge.BLE.Scan(3000, out SpringCard.Bluetooth.Device[]
FoundDevices, out uint ErrorReason))
{
    Console.WriteLine("BLE scan failed, error {0}", ErrorReason);
}
else
{
    Console.WriteLine("{0} device(s) found", FoundDevices);
    foreach (SpringCard.Bluetooth.Device device in FoundDevices)
    {
        Console.WriteLine("{0}: Name={1}, Type={2}", device.Addr.ToString(),
device.Name, device.PrimaryServiceGuid.ToString();
    }
}
```

Error Reason

- `7 : No access to the BLE interface`
- `8 : No BLE readers available. Please verify the BLE interface and try again.`
- `9 : No compliant BLE reader found. Please verify that your device is turned ON and not already connected to another host.`

Command line

```
SpringCoreBleBridge.exe scan [--time=3000]
```

Running as a bridge application

Condition: driver must be installed.

Synopsis

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Create a new bridge between an instance of the driver and a BLE
device */
        public BLE(uint InstanceNumber);
```

```
        /** \brief Tell the bridge which BLE device to connect to */
        public SetDeviceAddr(BluetoothAddress addr);
        public SetDeviceAddr(string addr);

        /** \brief Parameter for authentication */
        public enum AuthMode
        {
            User = 0
            Admin = 1
        }

        /** \brief Tell the bridge that the BLE communication must be secured */
        public SetDeviceAuth(AuthMode authMode, byte[] secretKey);

        /** \brief Start running as a bridge */
        public void Start();

        /** \brief Stop running as a bridge */
        public void Stop();
    }
}
```

Start the bridge

C# code

Plain communication

```
SpringCard.PCSC.Bridge bridge = new SpringCard.PCSC.Bridge.BLE(0);

bridge.SetDeviceAddr("XX:XX:XX:XX:XX:XX");

bridge.Start();

(...)

bridge.Stop();
```

Secure communication

```
SpringCard.PCSC.Bridge bridge = new SpringCard.PCSC.Bridge.BLE(0);

bridge.SetDeviceAddr("XX:XX:XX:XX:XX:XX");
bridge.SetDeviceAuth(SpringCard.PCSC.Bridge.AuthMode.User, new byte[] { /* ...User
secret key... */ });

bridge.Start();

(...)

bridge.Join();

bridge.Stop();
```

Error Reason

- `10 : BlueTooth address is null or empty`
- `14 : Driver Instance is negative.`
- `10 : Driver does not exist.`
- `11 : Instance is already running.`

Command line

```
SpringCoreBleBridge.exe run [--device=XX:XX:XX:XX:XX:XX] [--auth-mode=<user|admin>] [--auth-key=XX..XX] [--instance=0]
```

Stop the bridge

C# code

```
SpringCard.PCSC.Bridge bridge = new SpringCard.PCSC.Bridge.BLE(0);

(...)

bridge.Stop();
```

Error Reason

- `10 : BlueTooth address is null or empty`
- `13 : Unable to use named event.`

Command line

```
SpringCoreBleBridge.exe stop [--device=XX:XX:XX:XX:XX:XX] [--instance=0]
```

Monitor the state of the device

Reading the reader state

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Device state */
        public enum DeviceState
        {
            Unknown,
            Missing,           /* The reader is not found in the nearby */
            Present,           /* The reader is present, but not connected yet */
            Error,             /* The reader is present, but the library has failed to
connect it */
            Connecting,        /* The reader is found, the library is trying to
connect it */
            Connected,         /* The library is connected to the reader */
            ConnectedSuspended /* The library is connected to the reader and the
reader is suspended */
        }
```

```
        /** \brief Get the current state of the device */
        DeviceState GetDeviceState();
    }
}
```

Notifications when the state changes

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief OnDeviceStateChange prototype */
        public delegate void DeviceStateChange(DeviceState newState);
        /** \brief Callback to be notified when the device state change */
        public DeviceStateChange OnDeviceStateChange;
    }
}
```

Reading the battery state

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Battery state */
        public enum BatteryState
        {
            Unknown,
            Discharged,
            Discharching,
            Charging,
            Charged
        }

        /** \brief Get the current state of the device's battery */
        BatteryState GetDeviceBatteryState();
    }
}
```

Notifications when the battery state (or level) changes

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief OnDeviceBatteryStateChange prototype */
        public delegate void DeviceBatteryStateChange(BatteryState newBatteryState);
        /** \brief Callback to be notified when the device battery state change */
        public DeviceBatteryStateChange OnDeviceBatteryStateChange;
    }
}
```

Controlling the device

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Force the device to disconnect from the PC */
        public void Disconnect();
        /** \brief Resume the device when it has entered the ConnectedSuspended state
*/
        public void Resume();
        /** \brief Shutdown the device */
        public void Shutdown();
        /** \brief Change a key */
        public void ChangeKey(AuthMode TargetKey, byte[] NewKeyValue);
        /** \brief Send a direct control command to the device */
        public bool Control(byte[] Command, out byte[] Response);
    }
}
```

Control the application to card workflow

NOTE This part is preliminary. Everything is subject to change...

Intercept the card status

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief OnCardInsert/OnCardRemove prototypes */
        public delegate void CardMoveCallback(BLE device, byte slot);
        /** \brief Callback to see the card insert movements */
        public CardMoveCallback OnCardInsert;
        /** \brief Callback to see the card remove movements */
        public CardMoveCallback OnCardRemove;
    }
}
```

Intercept the exchanges

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief OnCardCommand/OnCardResponse/OnControlCommand/OnControlResponse
prototypes
        If the callback returns false, the exchange is cancelled.
        */
        public delegate bool ExchangeCallback(BLE device, byte slot, byte[] buffer);
        /** \brief Callback to see all the C-APDUs going from applications to cards */
        public ExchangeCallback OnCardCommand;
        /** \brief Callback to see all the R-APDUs going from cards to applications */
        public ExchangeCallback OnCardResponse;
        /** \brief Callback to see all the Control commands going from applications to
readers */
        public ExchangeCallback OnControlCommand;
```

```
        /** \brief Callback to see all the Control responses going from readers to
applications */
        public ExchangeCallback OnControlResponse;
    }
}
```

Intercept connections/disconnections

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief OnPowerOnCommand/OnPowerOffCommand prototype
        If the callback returns false, the operation is cancelled.
        */
        public delegate bool CardPowerCommandCallback(BLE device, byte slot);
        /** \brief Callback to see the PowerOn commands */
        public CardPowerCommandCallback OnPowerOnCommand;
        /** \brief Callback to see the PowerOff commands */
        public CardPowerCommandCallback OnPowerOffCommand;

        /** \brief OnPowerOnResponse prototype
        If the callback returns false, the operation is cancelled.
        */
        public delegate bool CardPoweredResponseCallback(BLE device, byte slot, byte[]
atr);
        /** \brief Callback to see the PowerOn responses */
        public CardPoweredResponseCallback OnPowerOnResponse;
    }
}
```

Inject a fake response or simulate a card movement

```
namespace SpringCard.PCSC.Bridge
{
    public class BLE
    {
        /** \brief Inject a fake R-APDU */
        public void InjectCardResponse(byte slot, byte[] rapdu);

        /** \brief Inject a fake ATR */
        public void InjectPowerOnResponse(byte slot, byte[] atr);

        /** \brief Tell the driver that the card has been removed */
        public void InjectCardRemove(byte slot);

        /** \brief Tell the driver that the card has been inserted */
        public void InjectCardInsert(byte slot);
    }
}
```

Examples

Intercept a card C-APDU and provide a R-APDU later on

```
bool onCardCommand(BLE device, byte slot, byte[] capdu)
{
    /* Remember which device, which slot have a pending C-APDU */
    pendingDevice = device;
    pendingSlot = slot;
    /* Arm a timer that will provide my own R-APDU later on */
    sendResponseTimer.Enabled = true;
    /* Cancel the exchange - The C-APDU does not go down to the card */
    return false;
}

BLE pendingDevice;
byte pendingSlot;

void sendResponseTimer_Elapsed(Object source, System.Timers.ElapsedEventArgs e)
{
    /* Send my own R-APDU */
    pendingDevice.InjectCardResponse(pendingSlot, new byte[] { /* Fake R-APDU */ } )
}
```

Intercept a card C-APDU and simulate a card removal

```
bool onCardCommand(BLE device, byte slot, byte[] capdu)
{
    /* Remember which device, which slot have a pending C-APDU */
    pendingDevice = device;
    pendingSlot = slot;
    /* Arm a timer that will tell the card has been removed */
    removeCardTimer.Enabled = true;
    /* Cancel the exchange - The C-APDU does not go down to the card */
    return false;
}

BLE pendingDevice;
byte pendingSlot;

void removeCardTimer_Elapsed(Object source, System.Timers.ElapsedEventArgs e)
{
    /* Tell the driver the card has been removed */
    pendingDevice.InjectCardRemove(pendingSlot);
    /* Since the card is still in the reader, I must arm a new timer to 'insert' it
'again' */
    insertCardTimer.Enabled = true;
}

void insertCardTimer_Elapsed(Object source, System.Timers.ElapsedEventArgs e)
{
    /* Tell the driver the card has been inserted again */
    pendingDevice.InjectCardInsert(pendingSlot);
}
```

Provide a fake card R-APDU

```
bool OnCardResponse(BLE device, byte slot, byte[] buffer)
{
    /* Send my own R-APDU */
    device.InjectCardResponse(slot, new byte[] { /* Fake R-APDU */ });
    /* Cancel the exchange - The original R-APDU does not go up to the driver */
    return false;
}
```

Prevent the application from powering the card down

```
byte[] cardPoweredAtr;

bool OnPowerOnCommand(BLE device, byte slot)
{
    /* Don't repower a card that is already powered */
    if (cardPoweredAtr !=null)
    {
        /* Send to the driver the ATR we already know */
        device.InjectPowerOnResponse(slot, cardPoweredAtr);
        /* Cancel the operation - The PowerOn command does not go down to the reader
*/
        return false;
    }
    return true;
}

bool OnPowerOnResponse(BLE device, byte slot, byte[] atr)
{
    if (atr != null)
    {
        /* The card is now powered, remember its ATR */
        cardPoweredAtr = atr;
    }
    else
    {
        /* The card is mute */
        cardPoweredAtr = null;
    }
    /* Propagate the response up to the driver */
    return true;
}

bool OnPowerOffCommand(BLE device, byte slot)
{
    /* Cancel the operation - The card stays powered on forever */
    return false;
}

void OnCardRemove(BLE device, byte slot)
{
    /* Forget this card! */
    cardPoweredAtr = null;
}
```