



# HAPTICORE .NET Library v3.14

# Contents

|                                          |           |
|------------------------------------------|-----------|
| <b>1 Overview</b>                        | <b>2</b>  |
| 1.1 Requirements . . . . .               | 2         |
| 1.2 Compatibility . . . . .              | 2         |
| <b>2 Features</b>                        | <b>3</b>  |
| 2.1 Get Available Devices . . . . .      | 3         |
| 2.2 Establish a Connection . . . . .     | 3         |
| 2.3 Request Data . . . . .               | 3         |
| 2.3.1 Request per Function . . . . .     | 4         |
| 2.3.2 Request per Register . . . . .     | 5         |
| 2.3.3 Request in Bulk . . . . .          | 5         |
| 2.3.4 Request Periodic Data . . . . .    | 5         |
| 2.4 Receive Data . . . . .               | 6         |
| 2.4.1 Barrier . . . . .                  | 6         |
| 2.4.2 Commands . . . . .                 | 6         |
| 2.4.3 Current . . . . .                  | 6         |
| 2.4.4 CoilDriver . . . . .               | 6         |
| 2.4.5 Custom . . . . .                   | 7         |
| 2.4.6 Degauss . . . . .                  | 7         |
| 2.4.7 Encoder . . . . .                  | 7         |
| 2.4.8 Freewheeling . . . . .             | 7         |
| 2.4.9 HapticsGenerator . . . . .         | 7         |
| 2.4.10 Info . . . . .                    | 7         |
| 2.4.11 Leds . . . . .                    | 8         |
| 2.4.12 Lock . . . . .                    | 8         |
| 2.4.13 Report . . . . .                  | 8         |
| 2.4.14 SingleTick . . . . .              | 8         |
| 2.4.15 Tick . . . . .                    | 8         |
| 2.4.16 Torque . . . . .                  | 8         |
| 2.4.17 Response Status . . . . .         | 8         |
| 2.4.18 Sample . . . . .                  | 9         |
| 2.5 Send Data . . . . .                  | 9         |
| 2.6 Send Commands . . . . .              | 9         |
| 2.7 Connection Status . . . . .          | 10        |
| 2.8 Disconnect From The Device . . . . . | 10        |
| <b>3 Further Questions and Support</b>   | <b>10</b> |

# 1 Overview

The communication between a HAPTICORE and a Windows PC takes place via serial communication. Usually, the USB (serial to USB) port of the control unit is used. Several parameters, including haptic feedback, can be controlled with the provided HAPTICORE .NET Library. This version corresponds to the communication protocol v3.30 of our firmware.

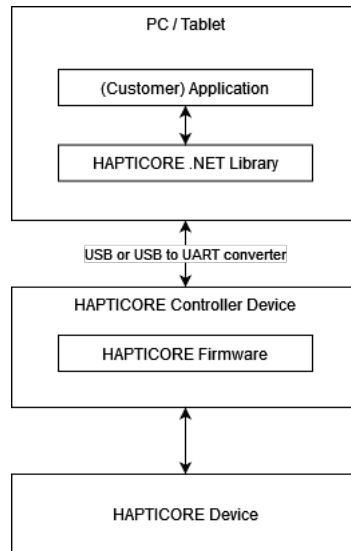


Figure 1: HAPTICORE System Overview

This document is an introduction to the aforementioned library and should guide the user through the initial steps as well as give an overview regarding the overall interface structure. A detailed description of individual haptic modes, functions and settings can be found in the *HAPTICORE Haptic Function Documentation*. Details regarding the communication protocol (and therefore the exposed interface options to the HAPTICORE .NET library) can be found in *HAPTICORE Serial Communication Protocol v3.30*.

## 1.1 Requirements

The following files are required:

- HaptiCoreLibrary.dll (developed and provided by XeelTech)
- Windows 7 or newer

The library is compiled with .NET Framework 4.6.

## 1.2 Compatibility

To ensure compatibility with the firmware implementing the communication protocol, the HAPTICORE library provides the compatible communication protocol version. It is strongly advised to use firmware with the same communication protocol version as the library to ensure correct behavior and functionality. If you use the firmware and .NET library from the same bundle provided by XeelTech, these versions will match by default. Just keep this requirement in mind when updating one of these components or using an older build / firmware version.

```
1 Version requiredVersion = hapticoreDeviceInstance.CompatibleCommunicationProtocolVersion;
2 ...
3
4 private void Handle(DataReceivedEventArgs args)
5 {
6     switch (args.Register)
7     {
8         case Register.REG_COMMUNICATION_PROTOCOL_VERSION:
9             var firmwareCommunicationProtocolVersion = (Version)args.Value;
10             // Ensure that the firmware communication protocol and .NET Library version match
11             bool compatible = firmwareCommunicationProtocolVersion == requiredVersion;
```

```
12         ...
13         break;
14
15     ...
16 }
17 }
```

---

## 2 Features

This chapter presents an overview of the HAPTICORE library architecture. The library is separated into four main namespaces as shown in graphic 2.

### 2.1 Get Available Devices

The class `HapticoreDevice` provides two static methods to list available serial ports:

---

```
1 // Version 1 to get the port names as strings only
2 ICollection<string> portNames = HapticoreDevice.GetAvailableSerialPorts();
3 ...
4
5 // Version 2 to receive more details
6 ICollection<UsbDeviceInfo> deviceInfos = HapticoreDevice.GetDeviceInfos();
7 ...
```

---

Version 1 returns a collection of strings in the form of "COM3", "COM4", "COM17", etc. while version 2 returns more detailed information, containing the device's VID and PID, manufacturer name, ...

If further filtering is required use Version 2, for a simple port selection / connection without preprocessing, the first version will suffice.

### 2.2 Establish a Connection

The class `HapticoreDevice` provides three connect methods. Each suits a different use case.

---

```
1 // Version 1
2 bool success = Connect(out string portName, out Exception exception);
3 ...
4
5 // Version 2
6 bool success = Connect(string portName, out Exception exception);
7 ...
8
9 // Version 3
10 bool success = ConnectUnchecked(string portName, out Exception exception);
11 ...
```

---

The first version connects to the first available, suitable device. This is preferred for establishing a connection to a demonstrator or evaluation kit quickly, without explicit COM port handling.

Each version returns the success status of the operation. If the method returns "false", the out parameter exception is set and contains the detailed exception.

The second variant connects to a provided COM port. This method is designed to be used by applications, which allow the user to select a COM port manually.

The third version omits certain checks (COM port used otherwise etc.) and is mainly provided to bypass certain compatibility issues at the cost of error handling. It is advised to use this method only in these special circumstances.

### 2.3 Request Data

Data can be requested in bulk, by functions or a by providing a register. The following sections describe each option to maximize flexibility.

The general communication is structured as request / reply model. An event (`DataReceived`) is provided by the `HapticoreDevice` instance, which provides all received data, while another event (`ErrorOccurred`), informs, as the name implies, about occurred errors.

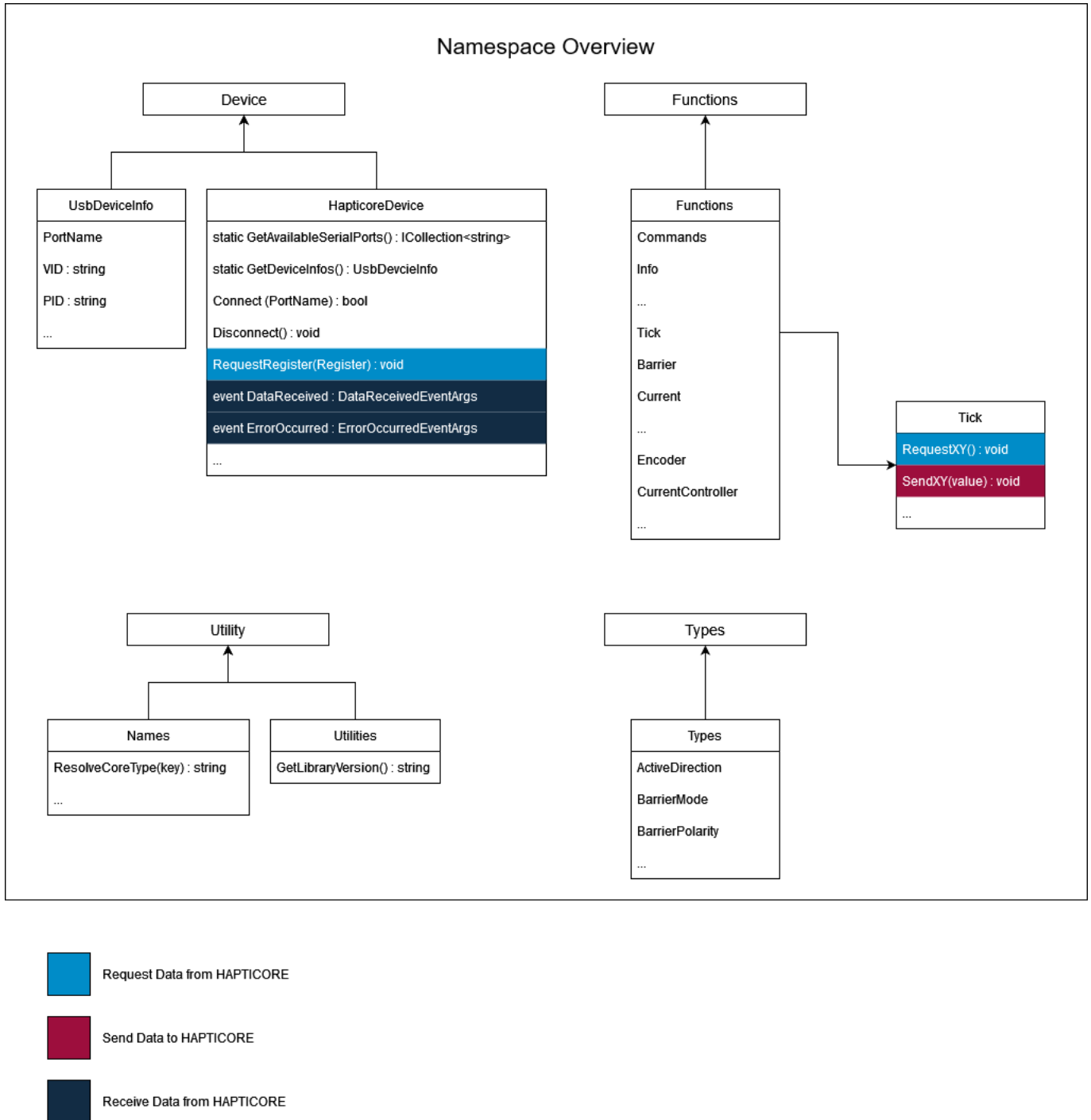


Figure 2: An overview regarding the structure of the HAPTICORE .NET library.

### 2.3.1 Request per Function

This is the most common use case. The HapticoreDevice instance provides instances of all available HAPTICORE functions. These functions include haptic (Tick, Barrier, Current, Torque, ...) capabilities as well as device related functions (CoilDriver, Encoder, ) and general information (Info, ...). All members of a specific function can be requested via the "RequestXY" method.

```

1 // Request the tick angle
2 hapticoreDeviceInstance.Tick.RequestAngleCw();
3 ...
4
5 // Request the encoder temperature
  
```

```
6     hapticoreDeviceInstance.Encoder.RequestTemperature();
7     ...
```

---

### 2.3.2 Request per Register

It's possible to query the byte value of the register directly using this function. This is only necessary in special cases and not recommended in general.

```
1     // Request the tick angle using the register directly
2     hapticoreDeviceInstance.RequestRegister(Register.REG_TICK_ANGLE_CW);
3     ...
4
5     // Request the encoder temperature using the register directly
6     hapticoreDeviceInstance.RequestRegister(Register.REG_ENCODER_TEMPERATURE);
7     ...
```

---

This variant yields the same results, retrieving the tick angle and encoder temperature as the method listed above does.

### 2.3.3 Request in Bulk

In rare cases it is necessary to retrieve all values of a HapticoreDevice function. For example, an application which lists all Tick parameter values currently set. To prevent, that all these values have to be requested one by one, there are the multiple options facilitating that process. The "ReadAllData" method is provided by each HAPTICORE function instance as well as by the HapticoreDevice instance as well. The second option requests all data for all the functions.

```
1     // Requests a single value - Same as above but listed here for comparison
2     hapticoreDeviceInstance.Tick.RequestAngleCw();
3     ...
4
5     // Requests all values related to "Tick" (Angle, Current, Mode, ...)
6     hapticoreDeviceInstance.Tick.RequestAllData();
7     ...
8
9     // Requests all values related to the "Encoder" function
10    hapticoreDeviceInstance.Encoder.RequestAllData();
11    ...
12
13    // Request all possible data from all provided HAPTICORE functions
14    hapticoreDeviceInstance.RequestAllData();
15    ...
```

---

### 2.3.4 Request Periodic Data

Periodic data, called reports, can be accessed via the Report function provided by the HapticoreDevice instance. Whilst several registers can be queried by the "RequestXY" methods, the "SendFlags" method provides the option to retrieve selected data in a defined interval. This interval, the report frequency, can be set by the user. The data is sent in a defined interval, if the report type is set to "cyclic". In contrast to that, "acyclic" reporting can be used to retrieve information only on change.

```
1     // Disable all reports (not recommended)
2     hapticoreDeviceInstance.Report.SendFlags(ReportFlags.NOTHING);
3     ...
4
5     // Enable only vital reports
6     hapticoreDeviceInstance.Report.SendFlags(
7     ReportFlags.DEVICE_ERROR_STATUS
8     | ReportFlags.DEVICE_CONNECTION_STATE);
9     ...
10
11    // Set the preferred report type
12    hapticoreDeviceInstance.Report.SendType(ReportType.REPORT_TYPE_ACYCLIC);
13
14    // Add arbitrary reports based on the use case
```

```

15     hapticoreDeviceInstance.Report.SendFlags(
16     ReportFlags.DEVICE_ERROR_STATUS
17     | ReportFlags.DEVICE_CONNECTION_STATE
18     | ReportFlags.ENCODER_ANGLE
19     | ReportFlags.ENCODER_VELOCITY
20     );
21     ...

```

---

## 2.4 Receive Data

Requested data as well as periodically provided data (reports) are funneled through the "DataReceived" event handler. Therefore, casting from the generic type "object" to the value's specific type is required. Since most of the values are numeric, "float" is the common data type but exceptions exist. The following tables lists all these exceptions with the according data type required for the cast of the received value object.

### 2.4.1 Barrier

| Register                | Data Type       |
|-------------------------|-----------------|
| REG_BARRIER_ENABLE      | bool            |
| REG_BARRIER_POLARITY    | BarrierPolarity |
| REG_BARRIER_LIMIT_ANGLE | bool            |

### 2.4.2 Commands

| Register                    | Data Type |
|-----------------------------|-----------|
| CMD_HAPTICORE_POWER_SUPPLY* | bool      |

\* This register is only supported when using the HAPTICORE Control Unit Pro

### 2.4.3 Current

| Register                     | Data Type       |
|------------------------------|-----------------|
| REG_CURRENT_ENABLE           | bool            |
| REG_CURRENT_ACTIVE_DIRECTION | ActiveDirection |

### 2.4.4 CoilDriver

| Register                                       | Data Type                 |
|------------------------------------------------|---------------------------|
| REG_CURRENT_CONTROLLER_MODE                    | CurrentControllerMode     |
| REG_CURRENT_CONTROLLER_SUPPLY_VOLTAGE_OVERRIDE | bool                      |
| REG_COIL_DRIVER_PWM_POLARITY                   | Direction                 |
| REG_COIL_DRIVER_STOP_MODE                      | CurrentControllerStopMode |
| REG_CURRENT_SENSE_POLARITY                     | Direction                 |
| REG_CURRENT_CONTROLLER_UPDATE_FREQUENCY        | ushort                    |
| REG_CURRENT_CONTROLLER_PWM_FREQUENCY           | ushort                    |

### 2.4.5 Custom

| Register                         | Data Type           |
|----------------------------------|---------------------|
| REG_CUSTOM_MODE_ENABLE           | bool                |
| REG_CUSTOM_MODE                  | CustomMode          |
| REG_CUSTOM_PARAMETER1_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER2_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER3_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER4_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER5_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER6_PROPERTIES | Tuple<bool, ushort> |
| REG_CUSTOM_PARAMETER7_PROPERTIES | Tuple<bool, ushort> |

### 2.4.6 Degauss

| Register              | Data Type   |
|-----------------------|-------------|
| REG_DEGAUSS_MODE      | DegaussMode |
| REG_DEGAUSS_FREQUENCY | ushort      |

### 2.4.7 Encoder

| Register              | Data Type   |
|-----------------------|-------------|
| REG_ENCODER_MODE      | EncoderMode |
| REG_ENCODER_DIRECTION | Direction   |

### 2.4.8 Freewheeling

| Register                | Type |
|-------------------------|------|
| REG_FREEWHEELING_ENABLE | bool |

### 2.4.9 HapticsGenerator

| Register                     | Data Type |
|------------------------------|-----------|
| REG_HAPTICS_GENERATOR_ENABLE | bool      |

### 2.4.10 Info

| Register                           | Data Type |
|------------------------------------|-----------|
| RPLY_STATUS                        | null      |
| REG_CONTROLLER_ID                  | ushort    |
| REG_CONTROLLER_HARDWARE_REVISION   | ushort    |
| REG_FW_VERSION                     | Version   |
| REG_COMMUNICATION_PROTOCOL_VERSION | Version   |
| REG_HAPTICORE_LIBRARY_VERSION      | Version   |
| REG_HAPTICORE_SERIAL_NUMBER        | string    |
| REG_HAPTICORE_TYPE                 | byte      |
| REG_HAPTICORE_PROTOTYPE_VERSION    | ushort    |
| REG_HAPTICORE_PROTOTYPE_NUMBER     | ushort    |
| REG_OPERATINGTIME1                 | ushort    |
| REG_OPERATINGTIME2                 | ushort    |
| REG_HAPTICORE_SYSTEM_CONFIGURATION | byte      |
| REG_HAPTICORE_PRODUCT_FAMILY       | byte      |
| REG_HAPTICORE_PRODUCT_MODEL        | ushort    |
| REG_HAPTICORE_PRODUCT_VARIATION    | byte      |
| REG_HAPTICORE_ITEM_NUMBER          | string    |



#### 2.4.11 Leds

| Register          | Data Type |
|-------------------|-----------|
| REG_LED_RGB_RED   | byte      |
| REG_LED_RGB_GREEN | byte      |
| REG_LED_RGB_BLUE  | byte      |

#### 2.4.12 Lock

| Register           | Data Type       |
|--------------------|-----------------|
| REG_LOCK_ENABLE    | bool            |
| REG_LOCK_DIRECTION | ActiveDirection |

#### 2.4.13 Report

| Register                          | Data Type                      |
|-----------------------------------|--------------------------------|
| REG_REPORT_TYPE                   | ReportType                     |
| REG_REPORT_FLAGS                  | ReportFlags                    |
| REG_REPORT_FREQUENCY              | ushort                         |
| REPORT_TICK_INDEX                 | short                          |
| REPORT_PUSH_PULL_STATE            | PushPullState                  |
| REPORT_ENCODER_MULTI_TURN_COUNT   | short                          |
| REPORT_TOTAL_TURN_COUNTER1        | ushort                         |
| REPORT_TOTAL_TURN_COUNTER2        | ushort                         |
| REPORT_DEVICE_ERROR_STATUS        | HapticoreDeviceErrorStatus     |
| REPORT_ENCODER_CALIBRATION_STATUS | EncoderCalibrationStatus       |
| REPORT_PUSH_CALIBRATION_STATUS    | PushPullCalibrationStatus      |
| REPORT_PULL_CALIBRATION_STATUS    | PushPullCalibrationStatus      |
| REPORT_DEVICE_CONNECTION_STATE    | HapticoreDeviceConnectionState |
| REPORT_DEGAUSS_STATUS             | DegaussStatus                  |

#### 2.4.14 SingleTick

| Register                         | Data Type       |
|----------------------------------|-----------------|
| REG_SINGLE_TICK_ENABLE           | bool            |
| REG_SINGLE_TICK_ACTIVE_DIRECTION | ActiveDirection |
| REG_SINGLE_TICK_MODE             | SingleTickMode  |

#### 2.4.15 Tick

| Register                  | Data Type          |
|---------------------------|--------------------|
| REG_TICK_MODE             | TickMode           |
| REG_TICK_ENABLE           | bool               |
| REG_TICK_ACTIVE_DIRECTION | ActiveDirection    |
| REG_TICK_INDEX_COUNT_MODE | TickIndexCountMode |

#### 2.4.16 Torque

| Register                    | Data Type       |
|-----------------------------|-----------------|
| REG_TORQUE_ENABLE           | bool            |
| REG_TORQUE_ACTIVE_DIRECTION | ActiveDirection |

#### 2.4.17 Response Status

The status of the response can be check by evaluating the "Status" byte. I can be cast to the "ReplyStatus" enum and separates the following possible states.

| Value         | Description                                                                |
|---------------|----------------------------------------------------------------------------|
| OK            | Operation successful                                                       |
| ERROR         | Operation failed due to wrong parameters or timing (already running, etc.) |
| NOT_SUPPORTED | The requested operation isn't supported                                    |

In general, when the status isn't "OK", the "Value" will most likely be null.

#### 2.4.18 Sample

Here is a sample for the event handler registered to DataReceived. Note how the different data types are handled by casting the value of "args.Value" to the respective data type.

```

1  private void HandleDataReceived(DataReceivedEventArgs args)
2  {
3      if ((ReplyStatus)args.Status != ReplyStatus.OK)
4      {
5          Trace.WriteLine($"Request for register {args.Register} failed " +
6              $"with status {(ReplyStatus)args.Status}");
7          return;
8      }
9
10     switch (args.Register)
11     {
12         case Register.RPLY_STATUS:
13             var replyStatus = (Tuple<byte, byte>)args.Value;
14             Register register = (Register)replyStatus.Item1;
15             byte errorCode = replyStatus.Item2;
16             bool notSupported = errorCode == ReplyStatus.NOT_SUPPORTED;
17             ...
18             break;
19
20         case Register.REG_TICK_ANGLE_CW:
21             float tickAngle = (float)args.Value;
22             ...
23             break;
24
25         ...
26     }
27 }
```

Further, its important to know that this event is invoked by a background thread. So, if the UI should update in regard to a received value, this has to be done from the respective UI thread.

## 2.5 Send Data

Sending data to the device has a smiliar structure than requesting data per function. Instead of the "RequestXY" the "SendXY" methods are used.

```

1  hapticoreDeviceInstance.Tick.SendMode(TickMode.TICK_MODE_1);
2  hapticoreDeviceInstance.Tick.SendAngleCw(15f);
3  hapticoreDeviceInstance.Tick.SendCurrent(0.7f);
4  ...
```

The HAPTICORE device confirms sending values by returning a response packet containing the current register value. These packages are received through the "DataReceived" event handler. The sent value can differ, if some limitations (min / max values) are applied by the firmware for example.

If a register isn't supported due to the firmware configuration, the reply status response (see in the sample above) is sent back, containing the register and status code of the failed attempt.

## 2.6 Send Commands

Commands trigger certain actions within the HAPTICORE firmware. Table 1 gives an overview of the available commands. They can be accessed with the "Commands" function on the HapticoreDevice instance.

| Command                       | Description                                                        |
|-------------------------------|--------------------------------------------------------------------|
| SendRebootSytem               | Restart the firmware                                               |
| SendLoadDefaultValues         | Load default parameter for the attached HAPTICORE                  |
| SendDegauss                   | Start the degauss process to remove any residual magnetism         |
| SendCalibrateEncoder          | Start the encoder calibration routine                              |
| SendRgbLeds                   | Configure the RGB lighting color (if supported)                    |
| SendResetStatistics           | Reset encoder statistics                                           |
| SendDisableAllHapticFunctions | Disables all active haptic functions (Tick, Barrier, Current, ...) |
| SendStartStopPowerMeasurement | Starts / stops the power measurement routine                       |
| SendEepromEraseOperation*     | Erase parts of the HAPTICORE's EEPROM memory                       |
| SendCalibratePushPull         | Starts the push or pull calibration routine (if supported)         |

Table 1: List of HAPTICORE commands

\* Don't use this command if not explicitly asked by the XeelTech support team!

## 2.7 Connection Status

The HAPTICORE connection status is provided by the HapticoreDevice instance. Further an event called "Connection-Changed" is available, which will be invoked if the connection status changes. The boolean parameter reflects the current connection status. False represents a disconnection event whilst true represents a successful (re)connection.

## 2.8 Disconnect From The Device

To disconnect from a device, simply invoke the "Disconnect" method provided by the respective HapticoreDevice instance.

## 3 Further Questions and Support

If the documentation doesn't help you regarding a specific problem or a part is unclear, please reach out to us for further assistance. We are glad to support you integrating the HAPTICORE .NET library.

## You still didn't find the answer to your question?

---

If you need additional assistance or have recommendations for changes, please don't hesitate to get in touch with us:

support@xeeltech.com  
+43 5552 93081-0  
www.xeeltech.com

Or visit [www.xeeltech.com/hapticore-support](http://www.xeeltech.com/hapticore-support) where you can find helpful FAQs and video tutorials.

Here you will also find our other HAPTICORE software solutions, allowing you to create even more personalized haptic feedback modes.

 **HAPTICORE**  
by  **XEELTECH**