# WaveX.Net Software Library

## 1. Introduction

The WaveX.NET software library implements all the functionalities required to fully control WaveX and Waveplus devices through a Personal Computer. The communication with the WaveX and Waveplus devices happens by USB cable, following the USB 3.0 or USB 2.0 specifications, by using a kernel mode USB function driver, which is Windows Driver Foundation (WDF) compliant. The WaveX.NET library can be used in C# or C++ Visual Studio projects for .NET platforms with Windows 11/10 (32/64 bit).

## 2. General Description

WaveX.NET is a multi-threading library developed in C# language which allows:

- The WaveX and Waveplus devices configuration:
  - data acquisition process configuration
  - sensors configuration
  - analog outputs configuration (available for WaveX system)
- Continuous data acquisition and acquired data integrity evaluation
- Monitoring of external Start/Stop triggers
- Impedance check on the EMG electrodes (available for Waveplus system)
- Offset compensation of signals produced by the accelerometers
- Inertial sensors calibration

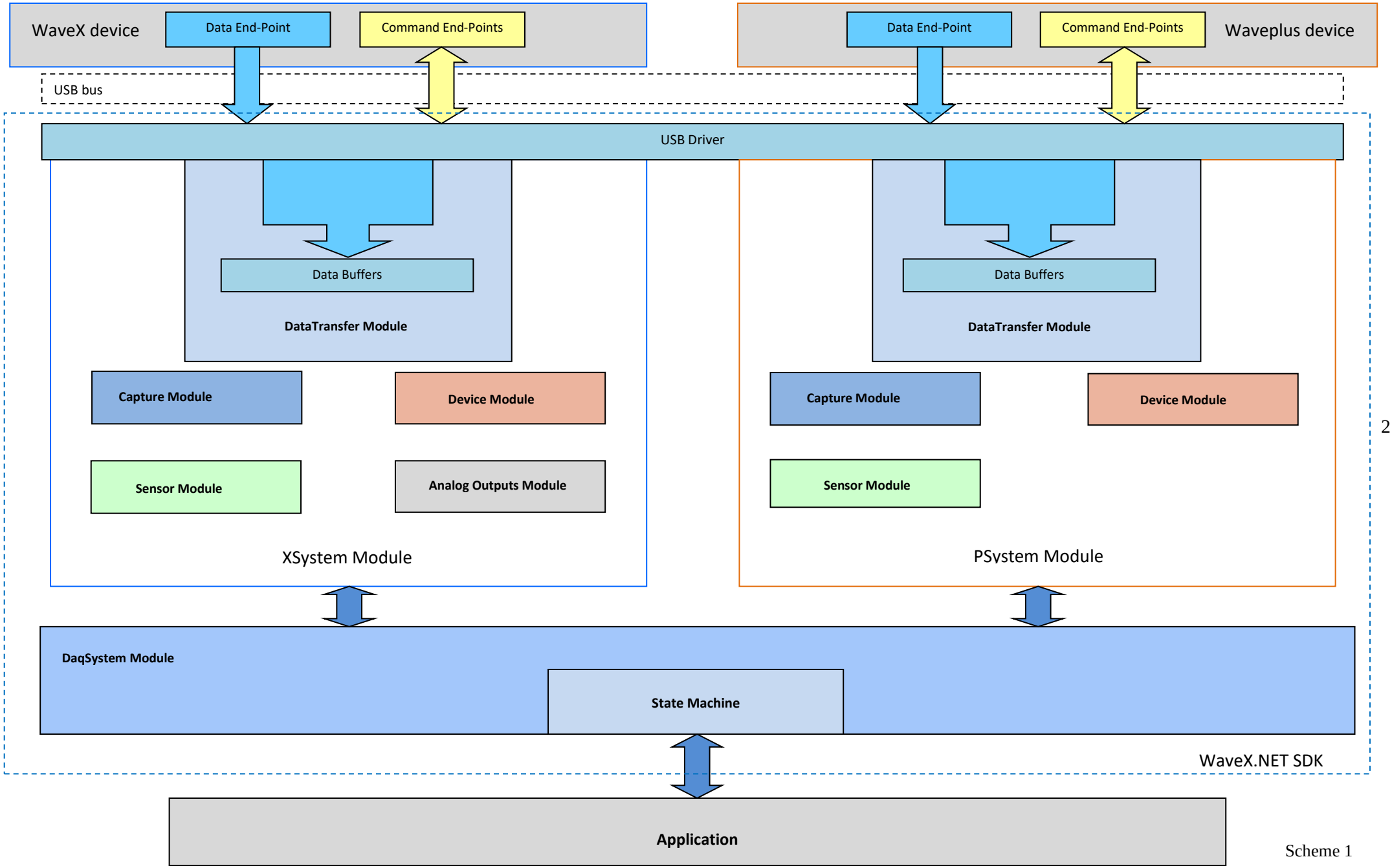
Being thread-safe, WaveX.NET realizes also a state machine in order to guarantee higher software reliability, avoiding wrong sequences in the access to the library functionalities, since it generates exceptions in case of errors occurred during the implementation of its own functionalities.

The WaveX.NET includes the following assemblies:

*WaveX.dll*
*WaveX.Common.dll*
*WaveX.BSys.dll*
*WaveX.PSys.dll*
*WaveX.XSys.dll*
*PicoBlue.DaqSys.dll*
*CyUSB.dll*

## 3. Architecture

Scheme 1 shows the main software modules.


WaveX device
Data End-Point
Command End-Points
Data End-Point
Command End-Points
Waveplus device
USB bus
USB Driver
Data Buffers
DataTransfer Module
Data Buffers
DataTransfer Module
Capture Module
Device Module
Capture Module
Device Module
Sensor Module
Analog Outputs Module
Sensor Module
XSystem Module
PSystem Module
DaqSystem Module
State Machine
WaveX.NET SDK
Application

Scheme 1

## 3.1 The DaqSystem Module

The DaqSystem module (namespace WaveX） includes the homonymous class whose methods and properties allow full control of the WaveX and Waveplus devices. This class implements the IDaqSystem interface (namespace WaveX.Interfaces） which includes the following properties, methods, and event handlers:

Red Rectangles functions: Wave Plus only
Blue Rectangles functions: WaveX only

### WaveX and Waveplus

DeviceState State { get; }
Represents the current state of the state machine which regularizes the access to the library (see DeviceState)

DeviceError InitialError { get; }
Includes the codification of a possible error that occurred during the execution of the DaqSystem class construction (see DeviceError)

List<DeviceType> Type { get; }
Represents the device type list of the WaveX and Waveplus devices connected to USB ports (see DeviceType)

List<DeviceMode> Mode{ get; }
Represents the device mode list of the WaveX and Waveplus devices connected to USB ports (see DeviceMode)

List<int> DeviceInstalledSensors { get; }
Represents the list of the installed sensors number for each WaveX and Waveplus device connected to USB ports

List<DeviceDependentFunctionalities> { get; }
It represents the device-dependent functionalities list of the WaveX and Waveplus devices connected to USB ports (see DeviceDependentFunctionalities)



void ConfigureCapture(ICaptureConfiguration captureConfiguration);
Configures the data acquisition process (see CaptureConfiguration)

ICaptureConfiguration CaptureConfiguration();
Gets the current configuration of the data acquisition process (see CaptureConfiguration)

void StartCapturing(DataAvailableEventPeriod dataAvailableEventPeriod);
Determines the start of the data acquisition process
dataAvailableEventPeriod represents the time interval between two consecutive DataAvailableEvents events (see DataAvailableEvents and DataAvailableEventPeriod)

void StopCapturing();
Determines the stop of the data acquisition process

void GenerateInternalStartTrigger();
Generates via software the internal start trigger activation (see DataAvailableEventArgs)

void GenerateInternalStopTrigger();
Generates via software the internal stop trigger activation (see DataAvailableEventArgs)

List<IVersion> FirmwareVersion { get; }
Represents the firmware version list of the WaveX and Waveplus devices connected to the USB ports (see Version)

List<IVersion> HardwareVersion { get; }
Represents the hardware version list of the WaveX and Waveplus devices connected to the USB ports (see Version)

IExtVersion SoftwareVersion { get; }
Represents the Waveplus.Daq.NET software library version (see ExtVersion)

int InstalledSensors { get; }
Represents the total number of the installed sensors

int InstalledFootSwSensors { get; }
Represents the total number of the installed Foot Switch sensors

SensorModel[] SensorsModel()
Returns a vector of InstalledSensors elements each of them represents the corresponding sensor model (see SensorModel)

void EnableSensor(int sensor);
Enables the specified sensor when sensor assumes the value between 1 and InstalledSensors. If sensor assumes the value 0, all the installed sensors are enabled.

void DisableSensor(int sensor);
Disable (stand-by mode) the selected sensor when sensor assumes the value between 1 and Installedsensors. If sensor assumes the value 0, all sensors are disabled.

void ConfigureSensor(ISensorConfiguration sensorConfiguration, int sensor);
Configures the selected sensor when sensor assumes the value between 1 and Installedsensors.
If sensor assumes the value 0, all sensors are configured (see SensorConfiguration)

ISensorConfiguration SensorConfiguration(int sensor);
Returns the current configuration of the selected sensor from the "sensor" parameter. This parameter can have value between 1 and InstalledSensors (see SensorConfiguration)

event EventHandler<DeviceStateChangedEventArgs> StateChanged;
Represents the handler of the StateChanged event. This is generated every time the state machine, that manages the access to the library, changes state (see DeviceStateChangedEventArgs)

event EventHandler<DataAvailableEventArgs> DataAvailable;
Represents the handler of the DataAvailable event. This is generated during the acquisition process, when new samples are available (see Scheme 2). To optimize the data processing and transfer, the samples are buffered and published around every DataAvailableEventPeriod ms (see StartCapturing() and DataAvailableEventPeriod)



void UpdateDisplay();
Update the WaveX display according to the current device configuration

void EnableFootSwSensors();
Enables all Foot Switch sensors (2 in total)

void DisableFootSwSensors();
Disables (stand-by mode) all Foot Switch sensors (2 in total)

void DetectAccelerometerOffset(int sensor);
Reads and compensates the offset of x, y, z channels of the selected acceleromets (Waveplus EMG sensor accelerometers), when sensor has value between 1 and InstalledSensors. If sensor has value 0, than all accelerometers are compensated for offset

SensorCheckReport[] CheckElectrodeImpedance(int sensor);
Executes the impedance check on the sensor selected, when sensor has value between 1 and InstalledSensors. If sensor has value 0, than all sensors are checked for impedance. It returns a vector whose elements are of the type SensorCheckReport; it includes all check results for all installed sensors (see SensorCheckReport)

void CalibrateSensorImuOffset (int sensor);
Executes the accelerometer and gyroscope offset calibration of the selected inertial sensor, when sensor has value between 1 and InstalledSensors . If sensor has value 0, than it executes the accelerometer and gyroscope offset calibration of all the installed inertial sensors

void CalibrateGyroscopeOffset (int sensor);
Executes the gyroscope offset calibration of the selected inertial sensor, when sensor has value between 1 and InstalledSensors . If sensor has value 0, than it executes the gyroscope offset calibration of all the installed inertial sensors

void TurnSensorLedOn(int sensor);
Activates the blinking of the sensor LED, when sensor has value between 1 and InstalledSensors. If sensor has value 0, than the operation is repeated on all sensors

void TurnAllSensorLedsOn();
Activated the blinking of all sensors, including Footswitch sensors

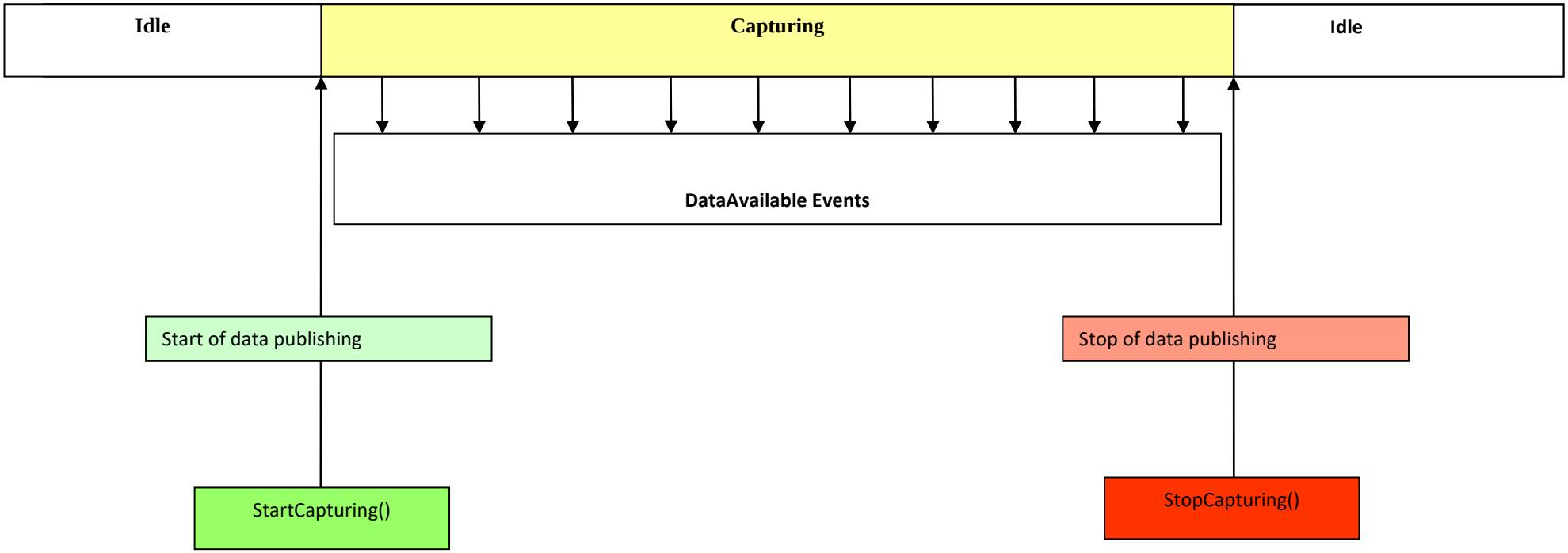
void TurnAllSensorLedsOff();
Disable the blinking of all sensors, including Footswitch sensors

## 3.2 DeviceState

Defines the ensemble of all states that the state machine can assume:

```
public enum DeviceState
{
    NotConnected,           No USB device is connected to the bus USB
    Initializing,           WaveX /Waveplus is initializing
    CommunicationError,     No communication with WaveX/Waveplus device available
    InitializingError,      WaveX/Waveplus device was not correctly initialized
    Idle,                   WaveX/Waveplus device is ready for use
    Capturing,              WaveX/Waveplus device is acquiring data
    ReadingSensorMemory,    WaveX/Waveplus device is reading data from sensor
                            memory
}
```

Typical state transition and events generation during data capturing process




Scheme 2

### 3.3 CaptureConfiguration

The class CaptureConfiguration implements the interface ICaptureConfiguration that includes the following properties:

bool ExternalTriggerEnabled { get; set; }
Enables/disables the external trigger

int ExternalTriggerActiveLevel { get; set; }
Defines the level at which trigger will start/stop the acquisition

bool ExternalTriggerStopDisable { get; set; }
Disables/Enables the external stop trigger

ImuAcqXType IMU_AcqXType { get; set; }
Defines WaveX IMU sensors acquisition type (see ImuAcqXType)

EmgAcqXType EMG_AcqXType { get; set; }
Defines WaveX EMG sensors acquisition type (see EmgAcqXType)

EmgImuAcqXType EMG_IMU_AcqXType { get; set; }
Defines WaveX EMG + IMU sensors acquisition type (see EmgImuAcqXType)

IFootSwTransducerEnabled FootSwATransducerEnabled { get; set; }
Enables/Disables the Footswitch A transducers (see FootSwTransducerEnabled)

IFootSwTransducerEnabled FootSwBTransducerEnabled { get; set; }
Enables/Disables the Footswitch B transducers (see FootSwTransducerEnabled)

IFootSwTransducerThreshold FootSwATransducerThreshold { get; set; }
Defines the activation levels for the transducers of Footswitch A (see FootSwTransducerThreshold)

IFootSwTransducerThreshold FootSwBTransducerThreshold { get; set; }
Defines the activation levels for the transducers of Footswitch B (see FootSwTransducerThreshold)

FootSwProtocol FootSwProtocol { get; set; }
Defines the protocol of processing of the data coming from the Footswitch sensors (see FootSwProtocol)

ImuAcqPType IMU_AcqPType { get; set; }
Defines Waveplus IMU sensors acquisition type (see ImuAcqPType)

EmgAcqPType EMG_AcqPType { get; set; }
Defines Waveplus EMG sensors acquisition type (see EmgAcqPType)

Defines the WaveX inertial sensors acquisition type

enum ImuAcqXType
{

| | |
|---|---|
| RawAccGyroMagData_400Hz | raw data acquisition (accelerometer, gyroscope, magnetometer) at 400 Hz |
| RawAccGyroData_500Hz | raw data acquisition (accelerometer, gyroscope) at 500 Hz |
| Fused6xData_400Hz | 6 axis fused data (quaternions) acquisition at 400 Hz |
| Fused9xData_400Hz | 9 axis fused data (quaternions) acquisition at 400 Hz |
| Mixed6xData_200Hz | 6 axis fused data (quaternions) acquisition at 200 Hz and raw data acquisition (accelerometer, gyroscope at 200 Hz, magnetometer at 100 Hz) |
| Mixed9xData_200Hz | 6 axis fused data (quaternions) acquisition at 200 Hz and raw data acquisition (accelerometer, gyroscope at 200 Hz, magnetometer at 100 Hz) |

}

**EmgAcqXType**

Defines the WaveX EMG and accelerometer sensors acquisition type

enum EmgAcqPType
{

| | |
|---|---|
| Emg_2kHz | EMG acquisition at 2 kHz |

}

**EmgImuAcqXType**

Defines the WaveX EMG+IMU sensors acquisition type

enum EmgImuAcqXType
{

| | |
|---|---|
| Emg_2kHz | EMG acquisition at 2 kHz |
| Emg_2kHz_RawAccGyroMagData_100Hz | EMG acquisition at 2 kHz and raw data acquisition (accelerometer, gyroscope and magnetometer at 100 Hz) |
| Emg_2kHz_RawAccGyroData_200Hz | EMG acquisition at 2 kHz and raw data acquisition (accelerometer and gyroscope at 200 Hz) |
| Emg_2kHz_Fused6xData_100Hz | EMG acquisition at 2 kHz and 6 axis fused data (quaternions) acquisition at 100 Hz |
| RawAccGyroMagData_400Hz | raw data acquisition (accelerometer, gyroscope, magnetometer) at 400 Hz |
| RawAccGyroData_500Hz | raw data acquisition (accelerometer, gyroscope) at 500 Hz |
| Fused6xData_400Hz | 6 axis fused data (quaternions) acquisition at 400 Hz |
| Fused9xData_400Hz | 9 axis fused data (quaternions) acquisition at 400 Hz |
| Mixed6xData_200Hz | 6 axis fused data (quaternions) acquisition at 200 Hz and raw data acquisition (accelerometer, gyroscope at 200 Hz, magnetometer at 100 Hz) |
| Mixed9xData_200Hz | 6 axis fused data (quaternions) acquisition at 200 Hz and raw data acquisition (accelerometer, gyroscope at 200 Hz, magnetometer at 100 Hz) |

}



**ImuAcqPType**

Defines the Waveplus inertial sensors acquisition type

enum ImuAcqXType
{

| | |
|---|---|
| RawAccGyroMagData_284Hz | raw data acquisition at 284 Hz |
| Fused9xData_142Hz | 9 axis fused data (quaternions) acquisition at 142 Hz |
| Fused6xData_284Hz | 6 axis fused data (quaternions) acquisition at 284 Hz |
| Fused9xData_71Hz | 9 axis fused data (quaternions) acquisition at 71 Hz |
| Fused6xData_142Hz | 6 axis fused data (quaternions) acquisition at 142 Hz |
| Mixed6xData_142Hz | 6 axis fused data (quaternions) acquisition at 142 Hz and raw data acquisition (accelerometer and gyroscope at 142 Hz, magnetometer at 47 Hz) |

}

**EmgAcqPType**

Defines the Waveplus EMG and accelerometer sensors acquisition type

enum EmgAcqPType
{

| | |
|---|---|
| Emg_2kHz_Acc_142Hz | EMG acquisition at 2 kHz, accelerometer acquisition at 142 Hz |

}

#### FootSwTransducerEnabled

The class FootSwTransducersEnabled implements the interface IFootSwTransducersEnabled that includes the following properties:

bool T_A { get; set; }
Enables/Disables the transducer A

bool T_1 { get; set; }
Enables/Disables the transducer 1

bool T_5 { get; set; }
Enables/Disables the transducer 5

bool T_T { get; set; }
Enables/Disables the transducer T

#### FootSwTransducerThreshold

The class FootSwTransducersThreshold implements the interface IFootSwTransducersThreshold that includes the following properties:

double T_A { get; set; }
Defines the threshold used for transducer A

double T_1 { get; set; }
Defines the threshold used for transducer 1

double T_5 { get; set; }
Defines the threshold used for transducer 5

double T_T { get; set; }
Defines the threshold used for transducer T

#### FootSwProtocol

Defines the protocol used to process the Footswitch samples

```
public enum FootSwProtocol
{
    FullFoot,       FullFoot protocol
    HalfFoot,       HalfFoot protocol
    QuarterFoot     QuarterFoot protocol
}
```



### 3.9 Version

The class Version implements the interface IVersion that includes the following properties:

int Major { get; }
Represents the Major part of the version number

int Middle { get; }
Represents the Middle part of the version number

int Minor { get; }
Represents the Minor part of the version number

### 3.10 ExtVersion

The class ExtVersion implements the interface IExtVersion that includes the following properties:

int Major { get; }
Represents the Major part of the version number

int Minor { get; }
Represents the Minor part of the version number

int Build { get; }
Represents the Build part of the version number

int Revision { get; }
Represents the Revision part of the version number

## 3.11 SensorConfiguration

The class SensorConfiguration implements the interface ISensorConfiguration that includes the following properties:

SensorModel SensorModel { get; set; }
Represents the sensor mode (see SensorModel)

SensorMode SensorMode { get; set; }
Represents the sensor mode (see SensorMode)

AccelerometerFullScale AccelerometerFullScale { get; set; }
Represents the full scale value for the axis x, y, z of the accelerometer (see AccelerometerFullScale)

GyroscopeFullScale GyroscopeFullScale { get; set; }
Represents the full scale value for the axis x, y, z of the gyroscope (see GyroscopeFullScale)

## 3.12 SensorModel

Represents the sensor model:
enum SensorModel
{

| | |
|---|---|
| Undefined | undefined |
| Mini_EmgImu | Mini EMG + IMU |
| Mini_Emg | Mini EMG |
| Pico_EmgImu | Pico EMG + IMU |
| Pico_Emg | Pico EMG |
| Imu | Imu |

| | |
|---|---|
| MiniPicoImuWavePlus | Waveplus sensor model |

}

## 3.12 SensorMode

Represents the sensor mode:
enum SensorMode
{

| | |
|---|---|
| EMG_SENSOR | EMG |
| INERTIAL_SENSOR | inertial |
| ANALOG_GP_SENSOR | general purpose |
| EMG_INERTIAL_SENSOR | EMG + inertial |
| FSW_SENSOR | foot-switch |
| EMG_ACC_SENSOR | EMG + accelerometer |

}

### 3.13 AccelerometerFullScale

Represents full scale value for the axis x, y, z of the accelerometer

```
public enum AccelerometerFullScale
{
        g_2,            Full scale = 2g
        g_4,            Full scale = 4g
        g_8,            Full scale = 8g
        g_16            Full scale = 16g
 }
                        g = gravitational field
```

### 3.14 GyroscopeFullScale

Represents full scale value for the axis x, y, z of the gyroscope

```
public enum GyroscopeFullScale
{
        dps_250,        Full scale = 250 D/s
        dps_500,        Full scale = 500 D/s
        dps_1000,       Full scale = 1000 D/s
        dps_2000        Full scale = 2000 D/s
 }
                        D/s = degree per second
```

### 3.15 SensorCheckReport

Defines the result of the impedance check of one sensor

```
public enum SensorCheckReport
{
        NotPassed,
        Passed,
        NotExecuted         (the sensor was not enabled or the test was not requested for that sensor)
}
```



### 3.16 DeviceStateChangedEventArgs

The class DeviceStateChangedEventArgs implements the following property:

```
DeviceState State { get; }
```

Represents the current device state

### 3.16 DataAvailableEventArgs

The class DataAvailableEventArgs implements the following properties:

```
public int ScanNumber { get; set; }
```

Represents the number of samples available for each channel

```
public float[,] EmgSamples;
```

Represents a vector with two dimensions including ScanNumber samples for each EMG channel
The samples are expressed in [uV] unit
The first index identifies the sensor (it can be between 0 and InstalledSensors -1)
The second index identifies the sample (it can be between 0 and ScanNumber -1)

```
public float[,,] ImuSamples;
```

Represents a vector with three dimensions including ScanNumber samples for each IMU quaternion components
The first index identifies the sensor (it can be between 0 and InstalledSensors -1)
The second can assume values of 0, 1, 2, 3 that identifies respectively the components w, x, y and z of the quaternion
The third index identifies the sample (it can be between 0 and ScanNumber -1)
ImuSamples samples are available only during IMU fused data acquisition or mixed acquisition

```
public float[,,] AccelerometerSamples;
```

Represents a vector with three dimensions including ScanNumber samples for each Accelerometer channel
The samples are expressed in [g] unit
The first index identifies the sensor (it can be between 0 and InstalledSensors -1)
The second can assume values of 0, 1, 2 that identifies respectively the values x, y and z of the acceleration vector
The third index identifies the sample (it can be between 0 and ScanNumber -1)

public float[,,] GyroscopeSamples;
Represents a vector with three dimensions including ScanNumber samples for each Gyroscope channel
The samples are expressed in [D/s] unit
The first index identifies the sensor (it can be between 0 and InstalledSensors -1)
The second can assume values of 0, 1, 2 that identifies respectively the values x, y and z of the angular velocity vector
The third index identifies the sample (it can be between 0 and ScanNumber -1)

public float[,,] MagnetometerSamples;
Represents a vector with three dimensions including ScanNumber samples for each Magnetometer channel
The samples are expressed in [uT] unit
The first index identifies the sensor (it can be between 0 and InstalledSensors -1)
The second can assume values of 0, 1, 2 that identifies respectively the values x, y and z of the magnetic vector
The third index identifies the sample (it can be between 0 and ScanNumber -1)

public short[,] FootSwSamples;
Represents a vector with two dimensions including the ScanNumber for each Footswitch channel
The samples are expressed in [V] unit
The first number identifies the sensor (it can be between 0 and InstalledFSWSensors -1)
The second number identifies the sample (it can be between 0 and ScanNumber -1)

public float[] SyncSamples;
Reserved



public short[] SensorStates;
Represents a vector including the state of each sensor .
The index value identifies the sensor (it can be between 0 and InstalledSensors -1)
(See Sensors State)

public short[] FootSwSensorStates;
Represents a vector including the state of each Foot-switch
The index value identifies the sensor (it can be between 0 and InstalledFSWSensors -1)
(See Sensors State)

public bool StartTriggerDetected;
Indicates if a start trigger signal was detected

public bool StopTriggerDetected;
Indicates if a stop trigger signal was detected

public int StartTriggerScan;
Represents the position (sample index value), in the data vectors, of the sample in correspondence of which the start trigger was detected. If no start trigger was detected, the value is 0

public int StopTriggerScan;
Represents the position (sample index value), in the data vectors, of the sample in correspondence of which the stop trigger was detected. If no stop trigger was detected, the value is 0

## 3.17 DataAvailableEventPeriod

Represents the time interval between two consecutive DataAvailableEvents events

```
public enum DataAvailableEventPeriod
{
        ms_100 ,        Interval = 100 ms
        ms_50 ,         Interval = 50 ms
        ms_25 ,         Interval = 25 ms
        ms_10           Interval = 10 ms
}
```

**Note**: interval values less than 100 ms need a very short execution-time of DataAvailableEvent event handler

### 3.18 Sensors state

Sensor state is represented by a 16 bit formatted according to the following table:

| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | BLAF | BL3 | BL2 | BL1 | BL0 |

where BL3 – BL0 represent the sensors battery charge level, BLAF represents the battery level available flag:

| BLA | BL3 | BL2 | BL1 | BL0 | Battery charge level |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0% |
| 1 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 0 | 1 | 1 | |
| 1 | 0 | 1 | 0 | 0 | |
| 1 | 0 | 1 | 0 | 1 | |
| 1 | 0 | 1 | 1 | 0 | |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | |
| 1 | 1 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 1 | 100% |
| 0 | X | X | X | X | Not Available |

**Note**: for Waveplus system the sensors battery charge level information is available only during EMG/Accelerometer, Foot-switch and IMU raw-data acquisitions

### 3.19 DeviceError

Defines the possible errors:

```
enum DeviceError
{
        Success = 0,
        UnspecifiedError= 1,
        DeviceNotConnected = 2,
        SendingCommand = 3,
        ReceivingCommandReply = 4,
        DeviceErrorExecutingCommand = 5,
        ConfiguringCapture = 6,
        ConfiguringAudio = 7,
        WrongCaptureConfigurationImuAcqType = 8,
        WrongCaptureConfigurationSamplingRate = 9,
        WrongCaptureConfigurationSamplingRateFromDevice = 10,
        WrongCaptureConfigurationDataAvailableEventPeriod = 11,
        WrongAudioConfigurationSpeakerLevel = 12,
        WrongAudioConfigurationSource = 13,
```

WrongAudioConfigurationSourceFromDevice = 14,
WrongSensorConfigurationAccelerometerFullScale = 15,
WrongSensorConfigurationGyroscopeFullScale = 16,
WrongSensorConfigurationSensorType = 17,
WrongFootSwProtocol = 18,
WrongDeviceTypeFromDevice = 19,
WrongDeviceMode = 20,
WrongSensorNumber = 21,
WrongFootSwSensorNumber = 22,
FootSwSensorNotInstalled = 23,
ReadingBackCaptureConfigurationSamplingRate = 24,
ReadingBackAudioConfigurationAudioSource = 25,
ReadingBackAudioConfigurationSpeakerVolumeLevel = 26,
ReadingBackAudioConfigurationSpeakerEnabled = 27,
ReadingBackCommunicationTestData = 28,
ReadingBackSensorCommandBuffer = 29,
ParsingFpgaConfigurationFile = 30,
SendingFpgaConfigurationImage = 31,
ReceivingUpdateFpgaConfigurationCommandReply = 32,
DeviceErrorExecutingUpdateFpgaConfigurationCommand = 33,
FpgaConfiguredFlagNotAsserted = 34,
ReadingFpgaConfiguredFlag = 35,
DataTransferThreadStartingTimeout = 36,
ActionNotAllowedInTheCurrentDeviceState = 37,
FeatureNotSupportedByTheDevice = 38,
ActionNotAllowedInTheCurrentDataTransferState = 39,
WrongTrialNumber = 40,
WrongDeviceState = 41,
WrongDeviceAction = 42,
TimeoutExecutingSensorCommand = 43,
CommandNotExecutedByAllTheSensors = 44,
TimeoutReadingSensorMemoryStatus = 45,
SensorMemoryHeaderCorrupted = 46,
SensorMemoryWrongTrialAddressFormat = 47,
SensorMemoryTrialAddressOutOfRange = 48,
WrongDaqTimeOutValue = 49,
BadSensorCommunication = 50,
SyncBuffer1Overrun = 51,
SyncBuffer2Overrun = 52,
ImuCalibrationNotAvailable = 53,
SaveMPStatusNotAvailable = 54,
FunctionNotAvailable = 55,
WrongDeviceRFChannelFromDevice = 56,
WrongImuAcqType = 57,
SensorDoesNotProvideSelectiveMemoryReading = 58,
ConflictingDeviceModeFromEEPROMandFromMicro = 59,
WrongEmgHwGain = 60,
WrongCaptureConfigurationImuAcqPType = 61,
WrongCaptureConfigurationImuAcqXType = 62,
WrongCaptureConfigurationEmgAcqPType = 63,
WrongCaptureConfigurationEmgAcqXType = 64,
WrongCaptureConfigurationEmgImuAcqXType = 65,
WrongCaptureConfigurationEmgAcqBType = 66,
WrongCaptureConfigurationImuAcqBType = 67,
UnavailableBSystemEmgAcqType = 68,
UnavailablePSystemEmgAcqType = 69,
UnavailablePSystemImuAcqType = 70,

SendingAnalogOutputsConfiguration = 71,
ReceivingAnalogOutputsConfigurationReply = 72,
DeviceErrorSendingAnalogOutputsConfiguration = 73,
ReadingBackAnalogOutputsConfiguration = 74,

SendingSensorMemoryStatusRequest = 75,
ReceivingSensorMemoryStatusReply = 76,

```
        ExecutingSensorMemoryStatusRequest = 77,
        DataPacketLost = 78,

        WiFiSensorMemoryReadingError1 = 79,
        WiFiSensorMemoryReadingError2 = 80,

        SendingSensorIdRequest = 81,
        ReceivingSensorIdReply = 82,
        ExecutingSensorIdRequest = 83,

        WrongSensorConfigurationSensorId = 84,

        WrongRFDeviceChannel = 85,
        WrongRFSensorChannel = 86,

        WiFiSensorMemoryReadingError3 = 87,
        WiFiSensorMemoryReadingError4 = 88,
        WiFiSensorMemoryReadingError5 = 89,
        WiFiSensorMemoryReadingError6 = 90,
        WiFiSensorMemoryReadingError7 = 91,
        WiFiSensorMemoryReadingError8 = 92,

        WrongDeviceCurrentTrialIdFromDevice = 93,
        WrongCommandCodeFromSensor = 94,
        WrongRFChannel = 95,
        WrongRFChannelName = 96,
        WrongSensorId = 97,
        WrongSensorIdCode = 98,
        WrongSensorIdName = 99,
}
```

### 3.20 DaqDeviceExceptionType



Defines the possible exception types generated by the library:

```
enum DaqDeviceExceptionType
{
        deviceNotConnected = 0,
        unableToStartCaptureDataTransfer = 1,
        unableToStartImpedanceDataTransfer = 2,
        unableToStartCapturing = 3,
        unableToStopCapturing = 4,
        unableToReadSensorMemoryStatus = 5,
        unableToGetCaptureConfiguration = 6,
        unableToSetCaptureConfiguration = 7,
        unableToGetInstalledSensors = 8,
        unableToGetDeviceType = 9,
        unableToConfigureSensor = 10,
        unableToGetSensorConfiguration = 11,
        unableToTurnInternalTrigger_OFF = 12,
        unableToTurnInternalTrigger_ON = 13,
        unableToEnableSensor = 14,
        unableToDisableSensor = 15,
        unableToEnableFootSwSensor = 16,
        unableToDisableFootSwSensor = 17,
        unableToDetectAccelerometerOffset = 18,
        unableToCheckElectrodeImpedance = 19,
        unableToGetElectrodeImpedanceReport = 20,
        unableToTurnSensorLedOn = 21,
        unableToTurnFootSwSensorLedOn = 22,
        unableToTurnAllSensorLedsOn = 23,
        unableToTurnAllSensorLedsOff = 24,
        unableToStartSensorMemoryRecording = 25,
        unableToStopSensorMemoryRecording = 26,
```

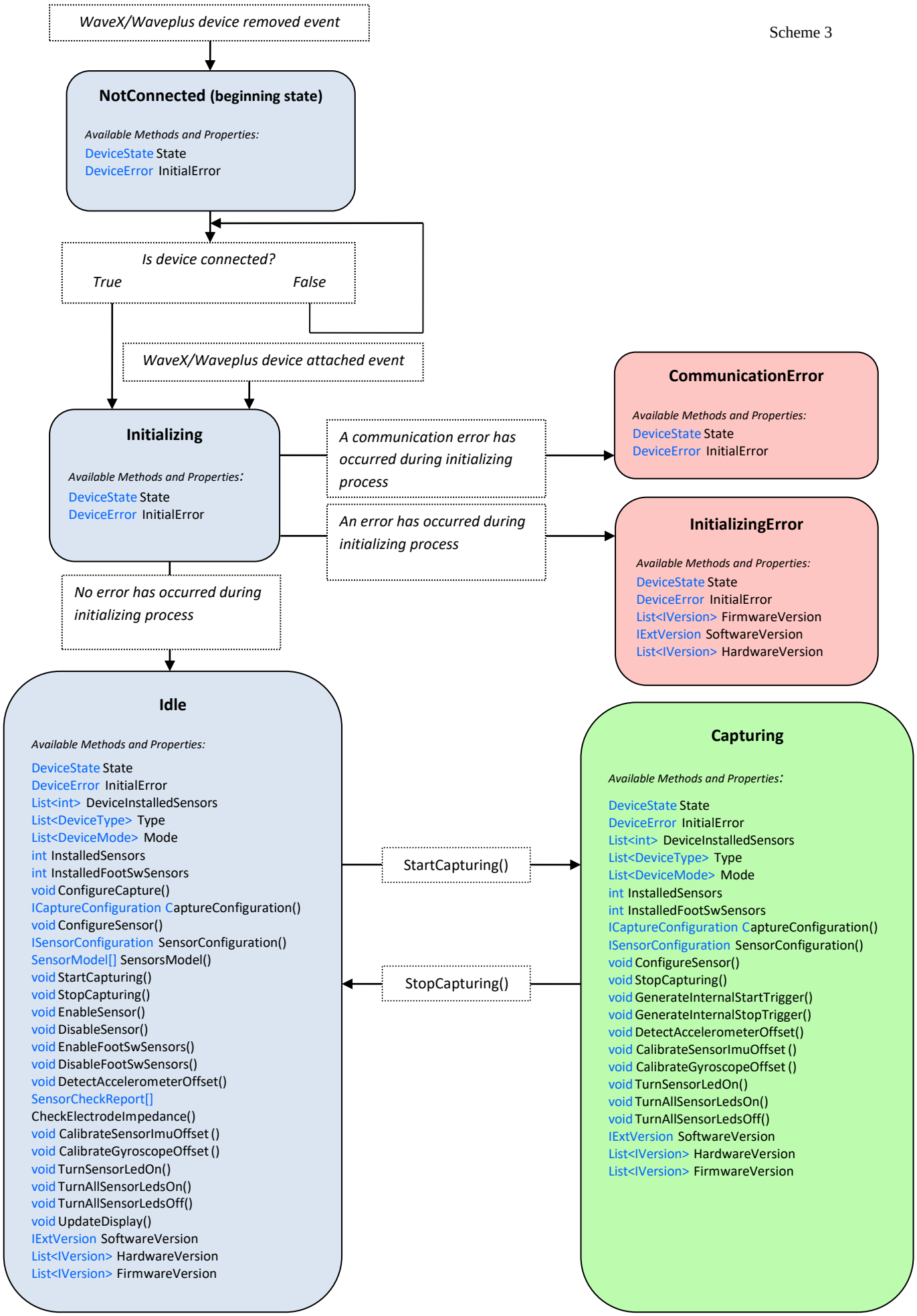
```
        unableToClearSensorMemory = 27,
        unableToFormatSensorMemory = 28,
        unableToStartSensorMemoryReading = 29,
        unableToStartSensorSelectiveMemoryReading = 30,
        unableToStopSensorMemoryReading = 31,
        unableToEnableSensorMemoryMode = 32,
        unableToDisableSensorMemoryMode = 33,
        unableToCalibrateSensorImu = 34,
        unableToCalibrateSensorGyroscope = 35,
        unableToSaveMPStatus = 36,
        unableToGetFirmwareVersion = 37,
        unableToGetHardwareVersion = 38,
        unableToSetAudioConfiguration = 39,
        unableToGetAudioConfiguration = 40,
        unableToConvertParameter = 41,
        unableToUpdateFirmware = 42,
        unableToGetFPGAConfigFlag = 43,
        unableToUpdateDeviceBoardEEPROMInfo = 44,
        unableToGetDeviceBoardEEPROMInfo = 45,
        unableToSynchronizeData = 46,
        unableToChangeDeviceRFChannel = 47,
        unableToChangeSensorRFChannel = 48,
        unableToGetDeviceRFChannel = 49,
        unableToSetFirstImuCalibrationStep = 50,
        unableToSetNextImuCalibrationStep = 51,
        unableToSetIMUAcqType = 52,
        unableToGetDeviceDependentFunctAvailability = 53,
        unableToReadSensorInfo = 54,
        unableToEnableRemoteControl = 55,
        unableToDisableRemoteControl = 56,
        unableToCalibrateSensorAccelerometer = 57,
        unableToSetEmgImuAcqType = 58,
        unableToGetDeviceSensorsId = 59,
        unableToSetEmgAcqType = 60,
        unableToSetImuAcqType = 61,
        unableToSetAnalogOutputsConfiguration = 62,
        unableToGetAnalogOutputsConfiguration = 63,
        unableToEnableDeviceWiFi = 64,
        unableToDisableDeviceWiFi = 65,
        unableToEnableSensorWiFi = 66,
        unableToDisableSensorWiFi = 67,
        unableToSetSensorMemoryTrialId = 68,
        unableToStartSensorMemoryDataTransfer = 69,
        unableToFindAndInitializeBDevice = 70,
        unableToDisconnectBDevice = 71,
        unableToSetDeviceDmaBufferSize = 72,
        unableToDisableWiFi = 73,
        unableToGetSensorMemoryStatus = 74,
        unableToReadSensorId = 75,
        unableToSetSensorSleepMode = 76,
        unableToSaveConfigurationOnXDevice = 77,
        unableToGetDeviceCurrentTrialId = 78,
        unableToGetSensorInfo = 79,
        unableToCalibrateSensorEmg = 80,
        unableToGetFixedMemoryId = 81,
}
```



## 4. State Machine: transitions and availability of Properties and Methods

The access to the library functionalities is regulated by a state machine aimed at improving the software reliability. In Scheme 3 the allowed transitions and the available methods/properties are highlighted for every state.

Scheme 3


WaveX/Waveplus device removed event
NotConnected (beginning state)
Available Methods and Properties:
DeviceState State
DeviceError InitialError
Is device connected?
True
False
WaveX/Waveplus device attached event
Initializing
Available Methods and Properties:
DeviceState State
DeviceError InitialError
A communication error has occurred during initializing process
CommunicationError
Available Methods and Properties:
DeviceState State
DeviceError InitialError
An error has occurred during initializing process
InitializingError
Available Methods and Properties:
DeviceState State
DeviceError InitialError
List<IVersion> FirmwareVersion
IExtVersion SoftwareVersion
List<IVersion> HardwareVersion
No error has occurred during initializing process
Idle
Available Methods and Properties:
DeviceState State
DeviceError InitialError
List<int> DeviceInstalledSensors
List<DeviceType> Type
List<DeviceMode> Mode
int InstalledSensors
int InstalledFootSwSensors
void ConfigureCapture()
ICaptureConfiguration CaptureConfiguration()
void ConfigureSensor()
ISensorConfiguration SensorConfiguration()
SensorModel[] SensorsModel()
void StartCapturing()
void StopCapturing()
void EnableSensor()
void DisableSensor()
void EnableFootSwSensors()
void DisableFootSwSensors()
void DetectAccelerometerOffset()
SensorCheckReport[]
CheckElectrodeImpedance()
void CalibrateSensorImuOffset ()
void CalibrateGyroscopeOffset ()
void TurnSensorLedOn()
void TurnAllSensorLedsOn()
void TurnAllSensorLedsOff()
void UpdateDisplay()
IExtVersion SoftwareVersion
List<IVersion> HardwareVersion
List<IVersion> FirmwareVersion
StartCapturing()
StopCapturing()
Capturing
Available Methods and Properties:
DeviceState State
DeviceError InitialError
List<int> DeviceInstalledSensors
List<DeviceType> Type
List<DeviceMode> Mode
int InstalledSensors
int InstalledFootSwSensors
ICaptureConfiguration CaptureConfiguration()
ISensorConfiguration SensorConfiguration()
void ConfigureSensor()
void StopCapturing()
void GenerateInternalStartTrigger()
void GenerateInternalStopTrigger()
void DetectAccelerometerOffset()
void CalibrateSensorImuOffset ()
void CalibrateGyroscopeOffset ()
void TurnSensorLedOn()
void TurnAllSensorLedsOn()
void TurnAllSensorLedsOff()
IExtVersion SoftwareVersion
List<IVersion> HardwareVersion
List<IVersion> FirmwareVersion