Waveplus.Daq.Net software library

Table of Contents

1. Introduction

The Waveplus.Daq.NET software library implements all the functionalities required to fully control a Waveplus device (Wireless EMG, Accelerometers and Inertial Sensors) through a Personal Computer. The communication with the Waveplus device happens by USB cable, following the USB 2.0 specifications, by using a driver, which respects the WDM (Windows Driver Model) standard. The Waveplus.Daq.NET library can be used in C# or C++ Visual Studio projects for .NET platforms with Windows XP, Vista, 7, 8, 8.1, 10 (32/64 bit).

2. General Description

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

- The Waveplus device configuration:
  - data acquisition process configuration
  - sensors configuration
- Continuous data acquisition and acquired data integrity evaluation
- Monitoring of external Start/Stop triggers
- Impedance check on the EMG electrodes
- Offset compensation of signals produced by the accelerometers
- Inertial sensors calibration

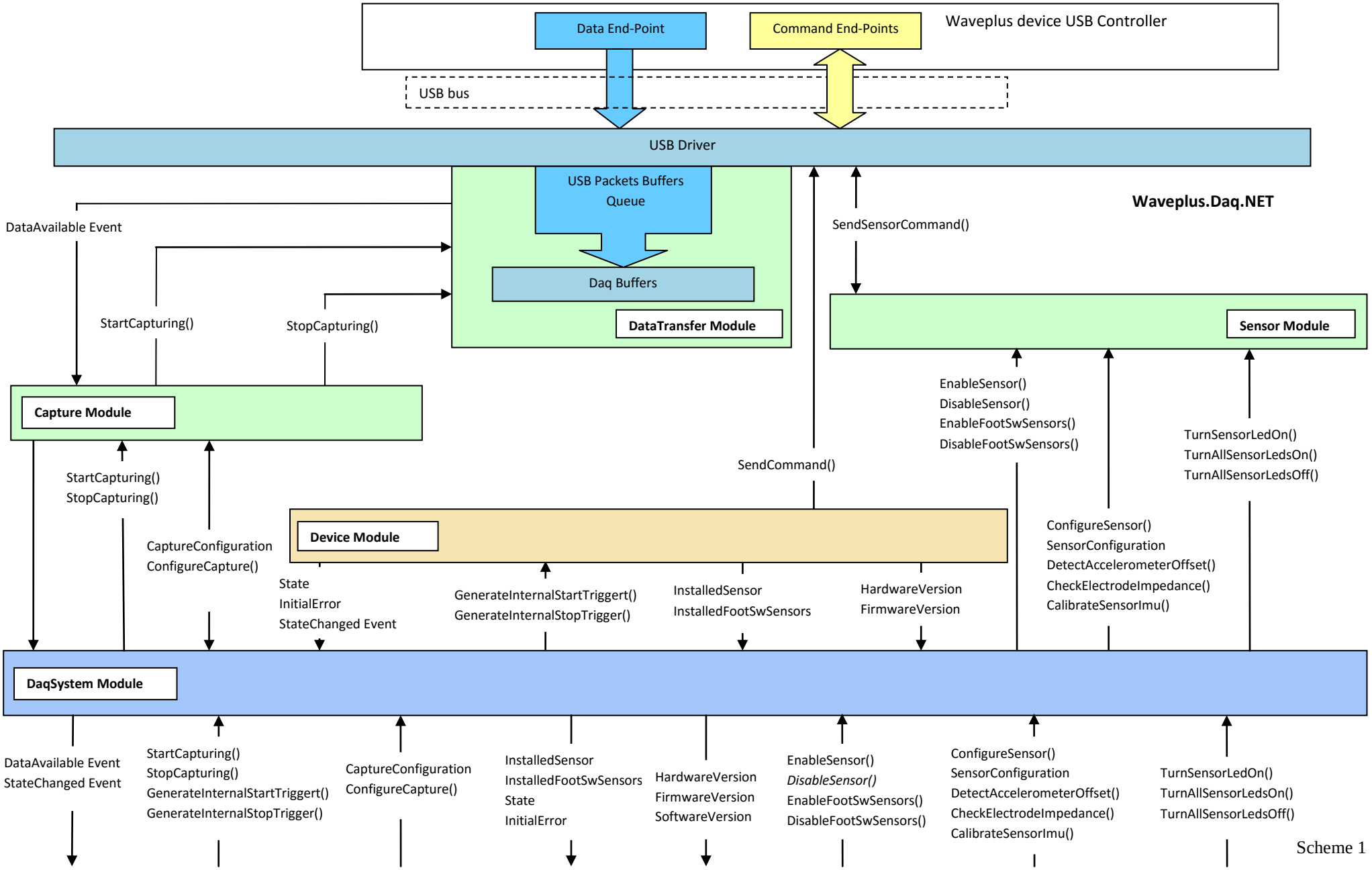
Being thread-safe, Waveplus.Daq.NET realizes also a state machine in order to guarantee higher software reliability, avoiding wrong sequences in the access to the library functionalities, since it generates exceptions in case of errors occurred during the implementation of its own functionalities.

The Waveplus.Daq.NET includes the following assemblies:

Waveplus.DaqSys.dll
Waveplus.DaqSysInterface.dll
CyUSBI.dll

3. Architecture

Scheme 1 shows the main software modules and functionalities.


Waveplus device USB Controller
Data End-Point
Command End-Points
USB bus
USB Driver
USB Packets Buffers Queue
Daq Buffers
DataTransfer Module
Waveplus.Daq.NET
DataAvailable Event
StartCapturing()
StopCapturing()
SendSensorCommand()
Sensor Module
Capture Module
EnableSensor()
DisableSensor()
EnableFootSwSensors()
DisableFootSwSensors()
TurnSensorLedOn()
TurnAllSensorLedsOn()
TurnAllSensorLedsOff()
SendCommand()
StartCapturing()
StopCapturing()
CaptureConfiguration
ConfigureCapture()
Device Module
State
InitialError
StateChanged Event
GenerateInternalStartTriggert()
GenerateInternalStopTrigger()
InstalledSensor
InstalledFootSwSensors
HardwareVersion
FirmwareVersion
ConfigureSensor()
SensorConfiguration
DetectAccelerometerOffset()
CheckElectrodeImpedance()
CalibrateSensorImu()
DaqSystem Module
DataAvailable Event
StateChanged Event
StartCapturing()
StopCapturing()
GenerateInternalStartTriggert()
GenerateInternalStopTrigger()
CaptureConfiguration
ConfigureCapture()
InstalledSensor
InstalledFootSwSensors
State
InitialError
HardwareVersion
FirmwareVersion
SoftwareVersion
EnableSensor()
DisableSensor()
EnableFootSwSensors()
DisableFootSwSensors()
ConfigureSensor()
SensorConfiguration
DetectAccelerometerOffset()
CheckElectrodeImpedance()
CalibrateSensorImu()
TurnSensorLedOn()
TurnAllSensorLedsOn()
TurnAllSensorLedsOff()

Scheme 1

3.1 The DaqSystem Module

The DaqSystem module includes the homonymous class whose methods and properties allow the full control of the Waveplus device. This class implements the IDaqSystem interface which includes the following properties, methods and event managers:

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 occurred during the execution of the DaqSystem class construction (see DeviceError)

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

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

ICaptureConfiguration CaptureConfiguration();
Represents the data acquisition process current configuration (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 activaton (see DataAvailableEventArgs)

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

List<IVersion> HardwareVersion { get; }
Represents the hardware version list of the 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 number of the installed sensors (EMG + INERTIAL)

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

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

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

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)

void DetectAccelerometerOffset(int sensor);
Reads and compensates the offset of x, y, z channels of the selected accelerometes, 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 CalibrateSensorImu (int sensor);
Executes the calibration of the selected inertial sensor, when sensor has value between 1 and InstalledSensors . If sensor has value 0, than it executes the 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

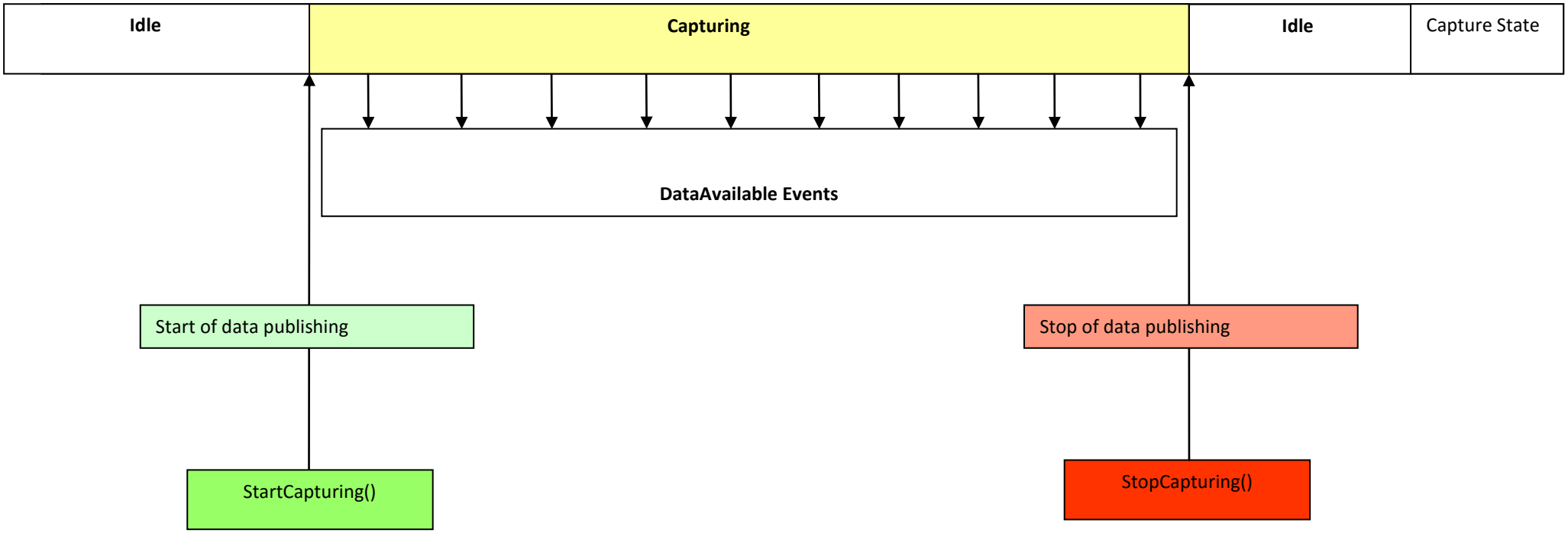
event EventHandler<DeviceStateChangedEventArgs> StateChanged;
Represents the handler of the StateChanged event. This is generated every time the state machine that manages the access to the library, changes state (see DeviceStateChangedEventArgs)

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)

3.2 DeviceState

Defines the ensembles of all states that the state machine can assume:
Waveplus.Daq.NET:

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


Idle
Capturing
Idle
Capture State
DataAvailable Events
Start of data publishing
Stop of data publishing
StartCapturing()
StopCapturing()

Scheme 2

3.3 CaptureConfiguration

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

SamplingRate SamplingRate { get; set; }
Defines the sampling frequency used (see SampligRate)

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

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

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)

ImuAcqType IMU_AcqType { get; set; }
Defines IMU acquisition type (see ImuAcquisitionType)

3.4 SamplingRate

Defines the sampling rate used during the acquisition of data

```
public enum SamplingRate
{
        Hz_2000                 2000 Hz
}
```

3.5 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

3.6 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

3.7 FootSwProtocol

Defines the protocol used to process the Footswitch samples

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

3.8 ImuAcqType

Defines the inertial sensors acquisition type

```
public enum ImuAcqType
{
    RawData,                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)
}
```


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 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 ISensorConfigurationthat includes the following properties:

SensorType SensorType { get; set; }
Represents the sensor type (see SensorType)

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 SensorType

Represents the sensor type:

```
public enum SensorType
{
        EMG_SENSOR,             EMG sensor
        INERTIAL_SENSOR,        INERTIAL sensor
        ANALOG_GP_SENSOR        ANALOG general purpose sensor
        FSW_SENSOR              FOOT-SWITCH sensor
 }
```

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 DataAvailableEventArgs

The class DataAvailableEventArgs implements the following properties:

public int SamplesNumber { get; set; }
Represents the number of samples available for each channel

public float[,] Samples;
Represents a vector with two dimensions including SamplesNumber 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 SamplesNumber -1)

public float[,,] ImuSamples;
Represents a vector with three dimensions including SamplesNumber 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 SamplesNumber -1)
ImuSamples samples are available only during IMU fused data acquisition

public float[,,] AccelerometerSamples;
Represents a vector with three dimensions including SamplesNumber 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 SamplesNumber -1)

public float[,,] GyroscopeSamples;
Represents a vector with three dimensions including SamplesNumber 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 SamplesNumber -1)
GyroscopeSamples samples are available only during IMU raw data acquisition

public float[,,] MagnetometerSamples;
Represents a vector with three dimensions including SamplesNumber 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 SamplesNumber -1)
MagnetometerSamples samples are available only during IMU raw data acquisition

public short[,] FootSwSamples;
Represents a vector with two dimensions including the SamplesNumber 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 SamplesNumber -1)

public float[] SyncSamples;
Reserved

public short[,] SensorStates;
Represents a vector with two dimensions including the state of each sensor in correspondence with every acquired sample
The first number identifies the sensor (it can be between 0 and InstalledSensors -1)
The second number (can assume values between 0 and SamplesNumber -1) identifies the sample that was acquired together with the state of the sensor.

public short[,] FootSwSensorStates;
Represents a vector with two dimensions including the state of each Footswitch sensor in correspondence with every acquired sample
The first number identifies the sensor (it can be between 0 and InstalledFSWSensors -1)
The second number (can assume values between 0 and SamplesNumber -1) identifies the sample that was acquired together with the state of the sensor.

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

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

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

public int StopTriggerScan;
Represents the position, in the data vectors, of the sample in correspondence of which the trigger stop was detected. If no trigger stop 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	0	0	0	BL1	BL0

where BL1 – BL0 represent the sensors battery charge level:

BL1	BL0	Battery charge level
0	0	0%
0	1	33%
1	0	66%
1	1	100%

Note: the sensors battery charge level information is available only during EMG/Accelerometer, Footswitch and IMU raw-data acquisitions

3.19 DeviceError

Defines the possible errors:

```
public enum DeviceError
{
    Success,
    DeviceNotConnected,
    SendingCommand,
```

```
        ReceivingCommandReply,
        DeviceErrorExecutingCommand,
        ConfiguringCapture,
        WrongCaptureConfigurationImuAcqType,
        WrongCaptureConfigurationSamplingRate,
        WrongCaptureConfigurationSamplingRateFromDevice,
        WrongCaptureConfigurationDataAvailableEventPeriod,
         WrongSensorConfigurationAccelerometerFullScale,
        WrongSensorConfigurationGyroscopeFullScale,
        WrongSensorConfigurationSensorType,
        WrongFootSwProtocol,
        WrongDeviceTypeFromDevice,
        WrongSensorNumber,
        WrongFootSwSensorNumber,
         FootSwSensorNotInstalled,
        ReadingBackCaptureConfigurationSamplingRate,
        ReadingBackCommunicationTestData,
         ReadingBackSensorCommandBuffer,
        DataTransferThreadStartingTimeout,
        ActionNotAllowedInTheCurrentDeviceState,
        ActionNotAllowedInTheCurrentDataTransferState,
        WrongDeviceState,
        WrongDeviceAction,
        TimeoutExecutingSensorCommand,
        CommandNotExecutedByAllTheSensors,
        WrongDaqTimeOutValue,
        BadSensorCommunication,
        SyncBuffer1Overrun,
        SyncBuffer2Overrun,
        ImuCalibrationNotAvailable
}
```


3.20 DaqDeviceExceptionType

Defines the possible exception types generated by the library:

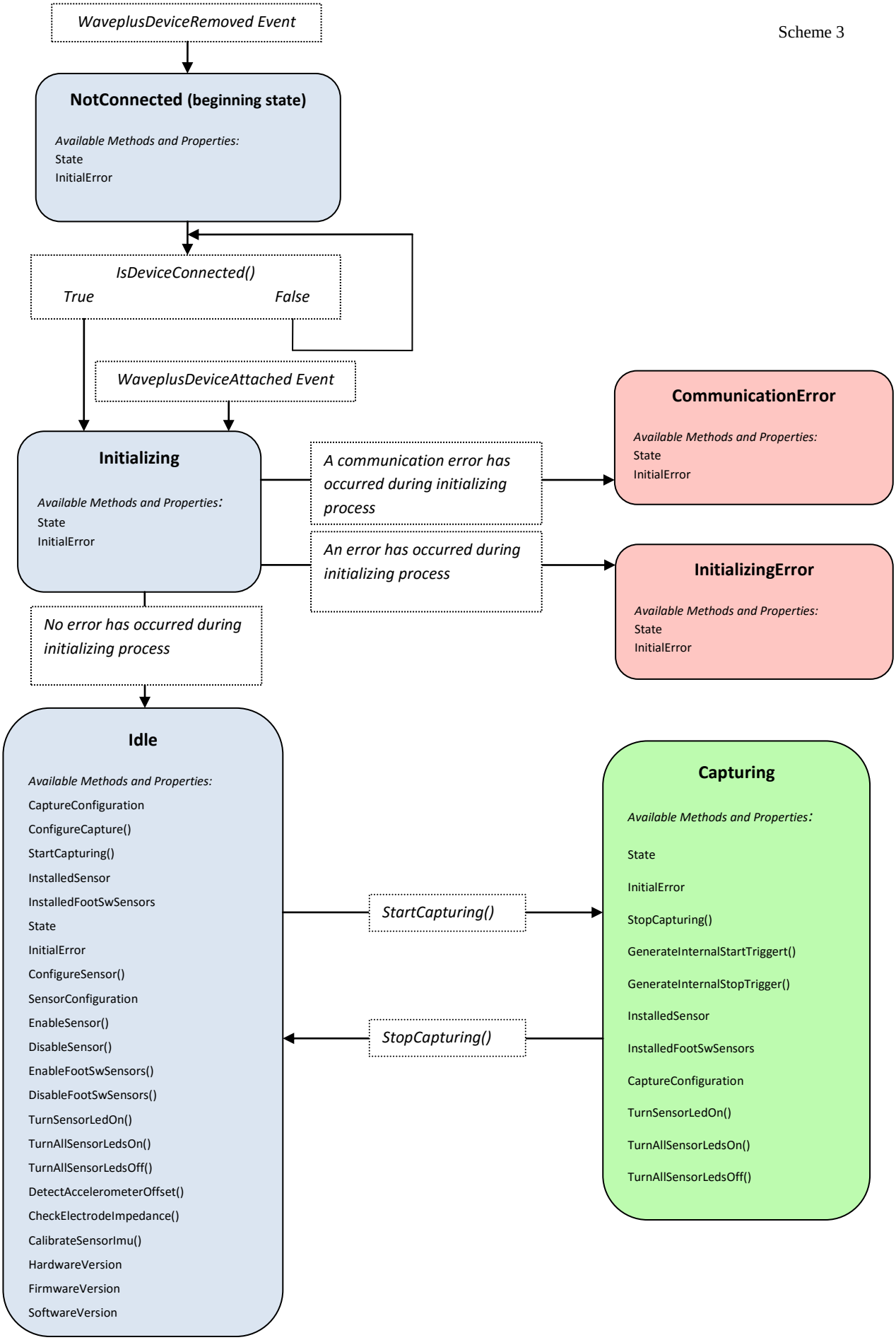
```
public enum DaqDeviceExceptionType
{
        deviceNotConnected,
        unableToStartCaptureDataTransfer,
        unableToStartImpedanceDataTransfer,
        unableToStartCapturing,
        unableToStopCapturing,
        unableToGetCaptureConfiguration,
        unableToSetCaptureConfiguration,
        unableToGetInstalledSensors,
        unableToGetDeviceType,
        unableToConfigureSensor,
        unableToGetSensorConfiguration,
        unableToTurnInternalTrigger_OFF,
        unableToTurnInternalTrigger_ON,
        unableToEnableSensor,
        unableToDisableSensor,
        unableToEnableFootSwSensor,
        unableToDisableFootSwSensor,
        unableToDetectAccelerometerOffset,
        unableToCheckElectrodeImpedance,
        unableToGetElectrodeImpedanceReport,
        unableToTurnSensorLedOn,
        unableToTurnFootSwSensorLedOn,
        unableToTurnAllSensorLedsOn,
        unableToTurnAllSensorLedsOff,
        unableToCalibrateSensorImu,
        unableToGetFirmwareVersion,
```

```
        unableToGetHardwareVersion,
        unableToConvertParameter,
        unableToGetFPGAConfigFlag,
        unableToSynchronizeData
}
```

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


WaveplusDeviceRemoved Event
NotConnected (beginning state)
Available Methods and Properties:
State
InitialError
IsDeviceConnected()
True
False
WaveplusDeviceAttached Event
Initializing
Available Methods and Properties:
State
InitialError
A communication error has occurred during initializing process
CommunicationError
Available Methods and Properties:
State
InitialError
An error has occurred during initializing process
InitializingError
Available Methods and Properties:
State
InitialError
No error has occurred during initializing process
Idle
Available Methods and Properties:
CaptureConfiguration
ConfigureCapture()
StartCapturing()
InstalledSensor
InstalledFootSwSensors
State
InitialError
ConfigureSensor()
SensorConfiguration
EnableSensor()
DisableSensor()
EnableFootSwSensors()
DisableFootSwSensors()
TurnSensorLedOn()
TurnAllSensorLedsOn()
TurnAllSensorLedsOff()
DetectAccelerometerOffset()
CheckElectrodeImpedance()
CalibrateSensorImu()
HardwareVersion
FirmwareVersion
SoftwareVersion
StartCapturing()
StopCapturing()
Capturing
Available Methods and Properties:
State
InitialError
StopCapturing()
GenerateInternalStartTriggert()
GenerateInternalStopTrigger()
InstalledSensor
InstalledFootSwSensors
CaptureConfiguration
TurnSensorLedOn()
TurnAllSensorLedsOn()
TurnAllSensorLedsOff()