

PalmSens SDK

Software Development Kit



Getting `started()`

Designed for:



INDEX

1. Contents of the PalmSens SDK	2
2. Using the SDK in your Visual Studio.NET project.....	3
3. PalmSens.DLL.....	4
4. Working with files.....	6
5. Connecting with PalmSens	11
6. PalmSens.Plot.dll	13
Appendix A: parameters for each technique	14

1. Contents of the PalmSens SDK

The PalmSens SDK contains the following libraries:

PalmSens.dll, containing the namespaces:

- PalmSens all necessary classes and functions for performing measurements and doing analysis with PalmSens or EmStat.
- PalmSens.Comm for Serial, USB or TCP communication with PalmSens or EmStat
- PalmSens.Comm.Streams contains classes for USB stream
- PalmSens.DataFiles for saving and loading method and data files
- PalmSens.Devices for handling communications, device capabilities and license information
- PalmSens.Script generic script parser (used by psscript.dll, not intended for other code development)
- PalmSens.Techniques contains all measurement techniques for PalmSens and EmStat
- PalmSens.Units contains a collection of units used with these libraries

PalmSens.Core.Data.dll, containing the namespace:

- PalmSens.Data can be used to represent the data in a Curve or in a Measurement as a System.Data.DataTable

PalmSens.Plot, containing the namespaces:

- PalmSens.Plot for visual representation of data measured with PalmSens or EmStat
- PalmSens.Printing representation of a Plot in a PrintDocument for printing purposes

PalmSens.Controls.dll a collection of controls used in PSLite and PSTrace

- Example program for Visual Basic .NET:

This program can be found in your PalmSens SDK installation directory, which is 'C:\Program Files\PalmSens SDK' by default.

This document contains a brief description of the basic functions of the DLL's in order to provide the programmer with a general idea about how to use them.

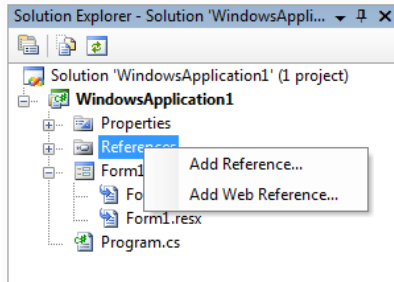
Technical information in MSDN library format can be found at:

<http://www.palmsens.com/software/custom-software/>

2. Using the SDK in your Visual Studio.NET project

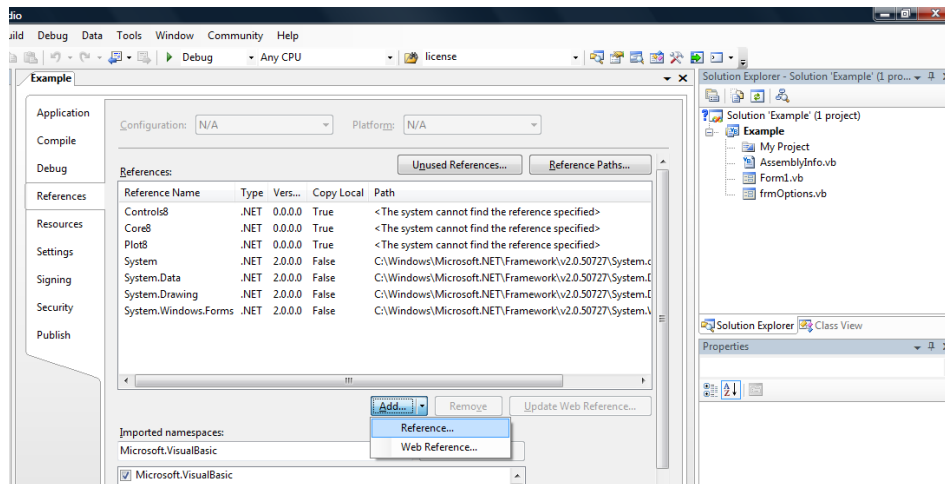
1) Add SDK References to the project In C#:

Right click on the References map in your project and click Add Reference:

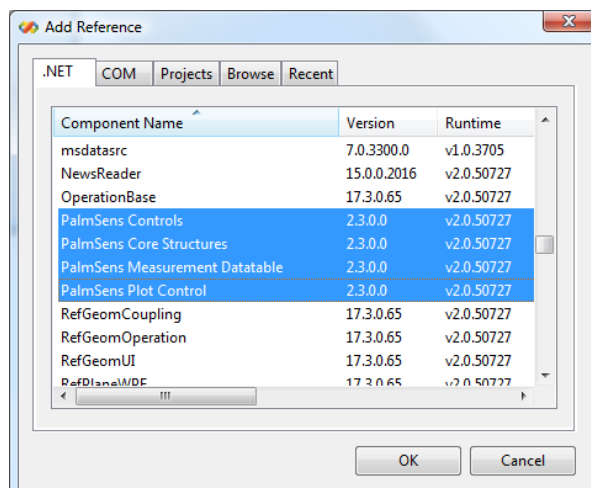


In VB.NET:

Open References tab in Project properties and click the arrow next to the Add... button. Choose Reference...

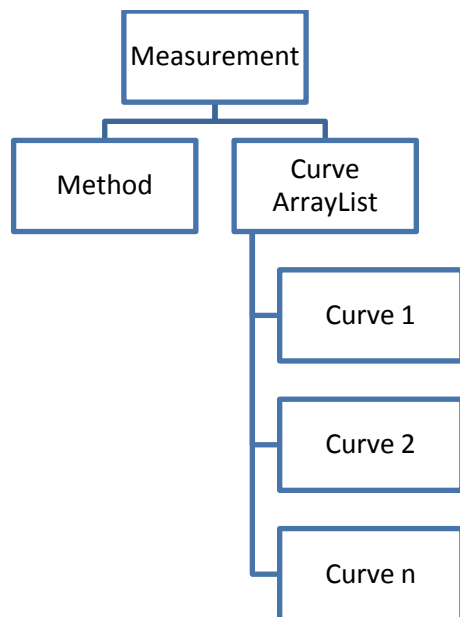


2) The libraries can be found in the tab with .NET components:

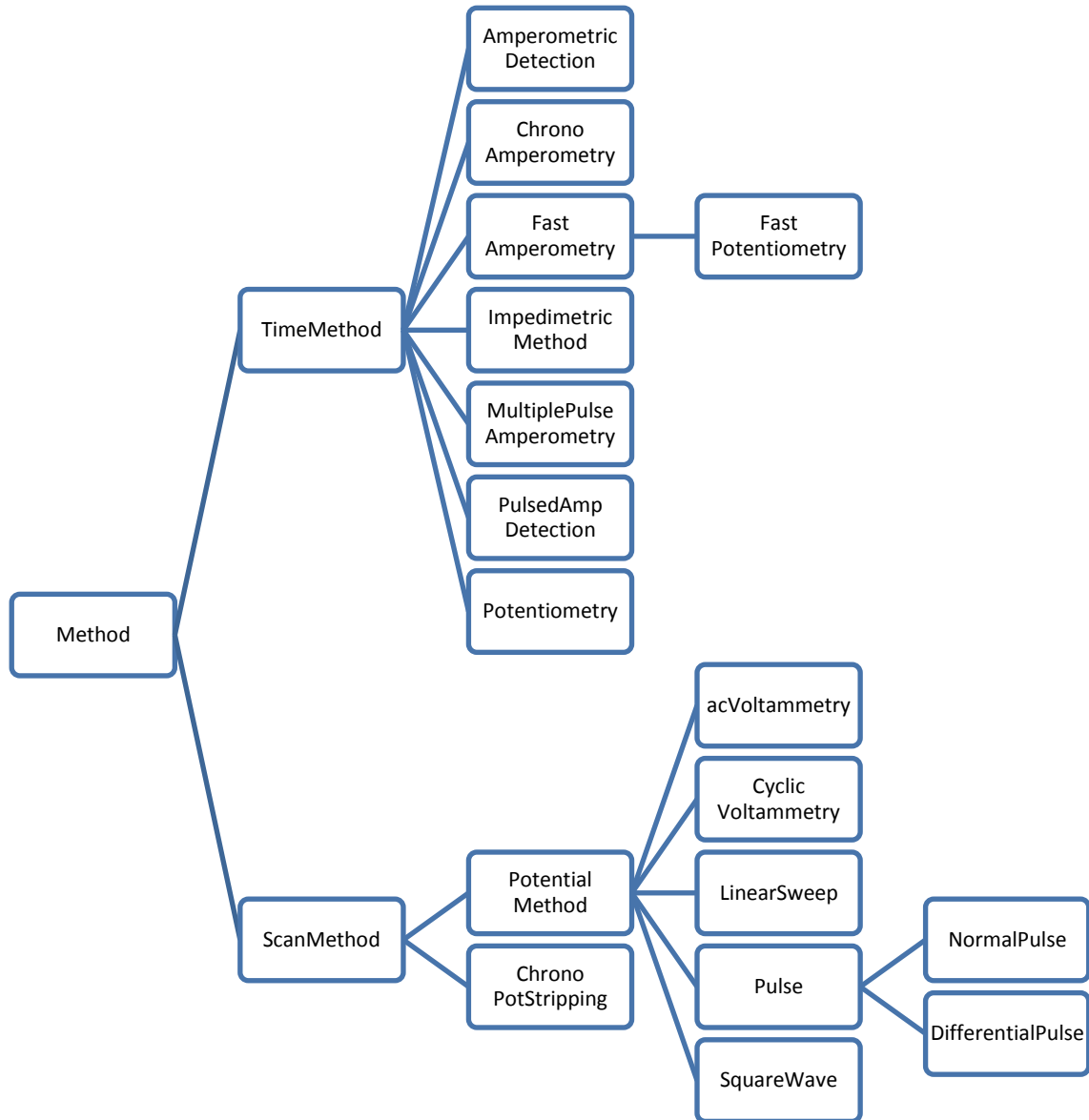


3. PalmSens.DLL

The basis for handling measurements is the `PalmSens.Measurement` class. The measurement class contains all classes, functions and parameters necessary for performing a measurement with a PalmSens or EmStat instrument. A measurement class can contain one method and multiple curves.



The following diagram shows the inheritance structure of the Method classes:



4. Working with files

A `Curve` class is always linked to a `Method` class, because the parameters determine what kind of measurement is done and also provides the parameters for peak searching, smoothing and Plot labels. This means that PalmSens datafiles (Curve files) always have a corresponding methodfile with the same name.

For example:

The data file [AmpDetection1.**pst**] is stored with methodfile [AmpDetection1.**pmt**]

The data file [DifferentialPulse1.**pss**] is stored with methodfile [DifferentialPulse1.**pms**]

There are two types of methodfiles; the scanmethod type (measure vs potential) and the timemethod type (measure vs time). The scanmethod type has the extension .pms, the timemethod type has the extension .pmt. Curves measured with a scanmethod have the extension .pss, curves measured with a timemethod have the extension .pst.

The following table shows the extensions currently used in PalmSens software:

	vs potential (scan method)	Measurement vs time
Method file	.pms	.pmt
Data (single curve) file	.pss	.pst
Analysis curves file	.psd	/
Multiplexer curves file	/	.mux

Loading a method

In order to use the following examples, make sure the PalmSens.Core.DLL is added as reference in your project and the following line is available at the top of your code:

```
Imports PalmSens.Core
```

The easiest way to work with method files is to use `Method` as a property that returns the method loaded by `MethodFile`:

```
Dim MethodFile As MethodFile  
Dim Method As Method
```

The parameters of a methodfile is loaded and put into a `Method` class by the following lines:

```
MethodFile = MethodFile.FromFile(filename)  
Method = MethodFile.Method
```

All loaded parameters are now available in the properties `EValue()` and `tValue()`. These parameters are identified by the properties `ELabel()` and `tLabel()`. The unit for each parameter is returned by the properties `EUnit()` and `tUnit()`.

The following tables show in which way the property-arrays are filled when an example **DifferentialPulse** method is loaded:

i	ELabel(i)	EValue(i)	EUnit(i)
0	E begin	-0.500	V
1	E end	0.500	V
2	E step	0.005	V
3	E pulse	0.025	V
4		0	V
5	E cond	-0.600	V
6	E dep	-0.500	V
7	E standby	-0.500	V

i	tLabel(i)	tValue(i)	tUnit(i)
0	Scan rate	0.0250	V/s
1	t pulse	0.070	s
2		0	s
3		0	s
4		0	s
5	t cond	0	s
6	t dep	0	s
7	t eq	8	s

As you notice ELabel(i) and tLabel(i) can be empty. This is because the corresponding parameter Evalue(4) is not relevant with this method type as is tLabel(2) to tLabel(4). All parameters for each technique have fixed positions in the tables. This means that if an acVoltammetry method is loaded for example, tLabel(1) will be empty because t pulse is not used with this method.

The following tables show in which way the property-arrays are filled when an example **acVoltammetry** method is loaded:

i	ELabel(i)	EValue(i)	EUnit(i)
0	E begin	-0.500	V
1	E end	0.500	V
2	E step	0.005	V
3	E ampl	0.025	V
4		0	V
5	E cond	-0.600	V
6	E dep	-0.500	V
7	E standby	-0.500	V

i	tLabel(i)	tValue(i)	tUnit(i)
0	Scan rate	0.0250	V/s
1		0	s
2		0	s
3	Freq	25	Hz
4		0	s
5	t cond	0	s
6	t dep	0	s
7	t eq	8	s

See appendix A for the tables of all measurement techniques

Editing a method

If you work with multiple types of measurement techniques in your program and use a table for editing parameters, you will find it easy to use Evalue() and tValue() for changing the parameters in the Method. But if your program uses only one or two techniques, you might want to use the Method parameters as properties. The next example defines a DifferentialPulse method with the same parameters shown in the table on the previous page.

Example:

```
Dim DiffPulse As New Techniques.DifferentialPulse

'set potential related parameters
DiffPulse.BeginPotential = -0.5
DiffPulse.EndPotential = 0.5
DiffPulse.StepPotential = 0.005
DiffPulse.PulsePotential = 0.025
```

```

'Set time related parameters
DiffPulse.Scanrate = 0.025
DiffPulse.PulseTime = 0.07
DiffPulse.EquilibrationTime = 0

'Set conditioning and deposition parameters
DiffPulse.ConditioningPotential = -0.6
DiffPulse.ConditioningTime = 5
DiffPulse.DepositionPotential = 0.6
DiffPulse.DepositionTime = 5

```

Note: The EValue() and tValue property-arrays are always synchronized with the corresponding properties as shown in this example.

The validation of your given parameters is returned in the Validate property. The following line shows the validation of the DiffPulse method in a messagebox.

```
MsgBox (DiffPulse.Validate)
```

Validate returns nothing if no error or warning is detected.

MethodFileEditor control

The MethodFileEditor control found in PalmSens.Controls enables to load, save and modify all properties of a method.

See the example project that comes with the SDK on how to use it.

PSNoiseTest.pms

Measurement Peaks Info Multiplexer

Technique: Linear Sweep

Sensor: Testsensor

Sample: This method is used to check the noise level

Name	Value	Unit
E begin	-0.5	V
E end	0.5	V
E step	0.005	V
E cond	-1.2	V
E dep	-1	V
scan rate	0.5	V/s
t cond	0	s
t dep	0	s
t eq	8	s

Cell on after measurement

Auxiliary Input

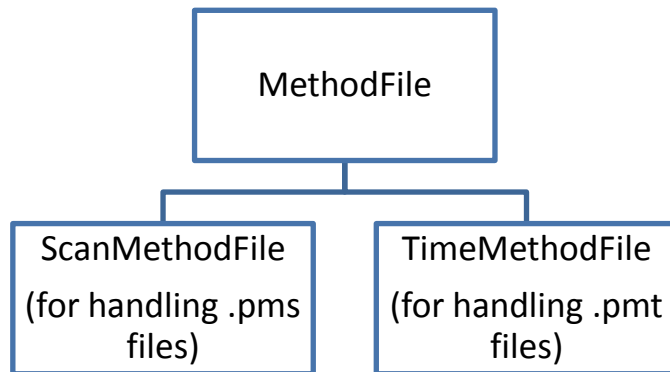
Record current on WE2 Record aux. input

1 nA 10 nA 100 nA 1 uA 10 uA 100 uA 1 mA 10 mA

Expected duration: 00:00:10 s
Number of data points: 201

Saving a method

To save the DiffPulse method with the parameters as defined in the previous example, the MethodFile class is used:



Example:

```
Dim DiffPulseFile As MethodFile = New ScanMethodFile  
  
DiffPulseFile.Method = DiffPulse  
DiffPulseFile.Save("DiffPulse.pms")
```

Curve Class

A `Curve` class contains the datapoints of a single curve. The property `Finished` determines whether the curve is ready for peak detection or smoothing. A curve received after a measurement is automatically flagged as `Finished` (`Finished=TRUE`).

Loading and saving curves

Every `Curve` is linked to its corresponding `Method` in order to perform peak detection and if necessary drawing the plot. This means that when a curve is loaded, the corresponding method file is loaded automatically in `Curve.Method`. Loading a curve from file is done by the following line:

```
Dim loadedCurve As Curve
loadedCurve = New CurveFile(stringfilename).Curve
```

Saving a curve using the `Save` function stores the curve in the standard `PalmSens` format (compatible with all available `PalmSens` software by Palm Instruments and Ivium). The `Save` function doesn't automatically save the methodfile.

IMPORTANT

```
loadedCurve = New CurveFile(stringfilename).Curve
```

Automatically loads the corresponding method.

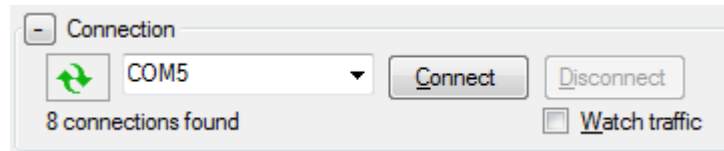
```
Dim cf As New CurveFile(Method, loadedCurve)
cf.Save(FileDialog.FileName)
```

Doesn't automatically save the corresponding method.

5. Connecting with PalmSens

Quickly connecting to PalmSens or EmStat

The easiest way to establish a connection with a PalmSens or EmStat is by using the `CommSelect` control in `PalmSens.Controls`.



This object automatically lists all connected devices for serial, USB and Bluetooth. The event `Connecting` can be used to receive the `Comm` class to control the connected device.

The following example has a private property `Comm` as `PalmSens.Controls.Comm`. After connection the connected device is set at a current range of 1 uA and a potential of 0 V.

```
Imports PalmSens
Imports PalmSens.Comm
Imports PalmSens.Controls

Dim WithEvents Comm As PalmSens.Controls.Comm

Private Sub Connecting(ByVal sender As System.Object, ByVal comm As _
    PalmSens.Controls.Comm) Handles CommSelect1.Connecting
    Me.Comm = comm
    comm.SendModeCR(Mode.Potentiostatic, False, _
    PalmSens.CurrentRange.FromMicroamps(1))
    comm.Potential = 0.0
End Sub
```

Manually connecting to a device

The following example shows how to get a list of all available devices and available serial com ports. The `ListBox1` control is a `System.Windows.Forms.ListBox`.

```
Dim devicelist As New PalmSens.Devices.DeviceList()

For Each device As PalmSens.Devices.Device In
    _devicelist.GetAvailableDevices(False)
    ListBox1.Items.Add(device.ToString())
Next
```

The boolean with `GetAvailableDevices(bool)` determines whether Bluetooth devices should be included in the search.

The following code is used to connect to the device selected in the list:

```
Dim device As PalmSens.Devices.Device
device = CType(ListBox1.SelectedItem, PalmSens.Devices.Device)

comm = New PalmSens.Controls.Comm(device)
```

In case the connection is made and `PalmSens`, the property `Active` is `TRUE`.

Receive readings

The readings of PalmSens can be read continuously using the ReceiveStatus event. The following information is received using this event:

- AuxInput (auxiliary input in V)
- Current (in uA)
- Current2 (in uA, in case a BiPot is used)
- Noise
- CurrentRange (the current range in use at the moment)
- CurrentStatus (as `PalmSens.Comm.Status` is ok, underload or overload)
- InputVoltage (voltage on auxiliary line)
- Potential (measured potential)
- ReverseCurrent (the reverse current for SquareWave)
- Status (string)
- VoltageOverload (boolean is TRUE in case of voltage overload)

Casting the StatusEventArgs to ExtendedStatus as in the following example provides Temperature and Noise readings:

```
CType(e.GetStatus(), PalmSens.Comm.ExtendedStatus).Noise()  
CType(e.GetStatus(), PalmSens.Comm.ExtendedStatus).Temperature()
```

Measuring

Starting a measurement is done by sending the method parameters to PalmSens:

```
Comm.Measure(Method)
```

As soon as the measurement is finished, the complete Curve is received in the Comm event ReceiveCurve. In case the measurement techniques allows online plotting, the Curve is provided in the event BeginReceiveCurve. The following example draws the data being received during a measurement in a Plot window (see next chapter):

```
Private Sub Comm_BeginReceiveCurve(ByVal sender As Object, ByVal e _  
As PalmSens.Comm.Comm.CurveEventArgs) Handles Comm.BeginReceiveCurve  
  
    Plot.Draw(e.GetCurve)  
End Sub
```

During a measurement the property `Comm.Busy` is TRUE.

Closing the Com port

The com port is **automatically closed** when the Com object is disposed:

```
Comm.Dispose()
```

6. PalmSens.Plot.dll

This DLL is used to visualize data measured by PalmSens.

Drawing curves

Curves can be plotted using the Draw method of Plot:

```
Plot.Draw(Curve)
```

A curve can be removed from the plot window using the Undraw property:

```
Plot.Undraw(Curve)
```

In case the curves are part of a single Measurement class all curves of the Measurement class can be plotted using this method:

```
Plot.Draw(m as Measurement)
```

To remove all curves from the plot window use the Clear method.

Scaling

In case the property SmartScaling is set to TRUE the maximum and minimum of the axis are rounded. If set to FALSE, the highest and lowest values in the Curve are used.

Appendix A: parameters for each technique

The method properties with corresponding indices, labels and units for all supported measurement techniques can be found here. Each technique has a specific integer value. This integer value can be used to create a class derived from this integer, as following:

```
PalmSens.DataFiles.MethodFile.FromTechnique(integervalue)
```

Scan measurement techniques

LinearSweep [0]

Class: PalmSens.Techniques.LinearSweep

i	ELabel(i)	EUnit(i)	Corresponding property
0	E begin	V	BeginPotential
1	E end	V	EndPotential
2	E step	V	StepPotential
3			
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	scan rate	V/s	Scanrate
1			
2			
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

DifferentialPulse [1]

Class: PalmSens.Techniques.DifferentialPulse

i	ELabel(i)	EUnit(i)	Method property
0	E begin	V	BeginPotential
1	E end	V	EndPotential
2	E step	V	StepPotential
3	E pulse	V	PulsePotential
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	scan rate	V/s	Scanrate
1	t pulse	s	PulseTime
2			
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

SquareWave [2]

Class: PalmSens.Techniques.SquareWave

i	ELabel(i)	EUnit(i)	Method property
0	E begin	V	BeginPotential
1	E end	V	EndPotential
2	E step	V	StepPotential
3	E ampl	V	PulseAmplitude
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E standby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0			
1			
2			
3	freq	Hz	Frequency
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

NormalPulse [3]

Class: PalmSens.Techniques.NormalPulse

i	ELabel(i)	EUnit(i)	Method property
0	E begin	V	BeginPotential
1	E end	V	EndPotential
2	E step	V	StepPotential
3			
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	scan rate	V/s	Scanrate
1	t pulse	s	PulseTime
2			
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

acVoltammetry [4]

Class: PalmSens.Techniques.acVoltammetry

i	ELabel(i)	EUnit(i)	Method property
0	E begin	V	BeginPotential
1	E end	V	EndPotential
2	E step	V	StepPotential
3	E ampl	V	SineWaveAmplitude
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	scan rate	V/s	Scanrate
1			
2			
3	freq	Hz	Frequency
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

CyclicVoltammetry [5]

Class: PalmSens.Techniques.CyclicVoltammetry

i	ELabel(i)	EUnit(i)	Method property
0	E vtx1	V	BeginPotential
1	E vtx2	V	EndPotential
2	E step	V	StepPotential
3			
4	E start	V	StartPotential
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	scan rate	V/s	Scanrate
1			
2			
3			
4	number of scans		nScans
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

ChronoPotStripping [6]

Class: PalmSens.Techniques.ChronoPotStripping

i	ELabel(i)	EUnit(i)	Method property
0			
1	E end	V	EndPotential
2			
3			
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0			
1			
2	t meas	s	MeasurementTime
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

Time measurement techniques

AmperometricDetection [7]

Class: PalmSens.Techniques.AmperometricDetection

i	ELabel(i)	EUnit(i)	Method property
0	E	V	BeginPotential
1			
2			
3			
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	interval	s	Intervaltime
1			
2	t run	s	RunTime
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

PulsedAmpDetection [8]

Class: PalmSens.Techniques.PulsedAmpDetection

i	ELabel(i)	EUnit(i)	Method property
0	E	V	BeginPotential
1			
2			
3	E pulse	V	PulsePotential
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	interval	s	Intervaltime
1	t pulse	s	PulseTime
2	t run	s	RunTime
3			
4	t mode	1=DC, 2=Pulse, 3=Dif	tMode
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

FastAmperometry [9]

Class: PalmSens.Techniques.FastAmperometry

i	ELabel(i)	EUnit(i)	Method property
0	E	V	BeginPotential
1			
2			
3	E pulse	V	PulsePotential
4			
5	E cond	V	ConditioningPotential
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	interval	s	Intervaltime
1			
2			
3			
4			
5	t cond	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

Potentiometry [10]

Class: PalmSens.Techniques.FastAmperometry

i	ELabel(i)	EUnit(i)	Method property
0	I appl	* current range	Current
1			
2			
3			
4			
5			
6			
7			
i	tLabel(i)	tUnit(i)	Method property
0	interval	s	Intervaltime
1			
2	t run	s	Runtime
3			
4			
5			
6			
7			

MultiplePulseAmperometry [11]

Class: PalmSens.Techniques.MultiplePulseAmperometry

i	ELabel(i)	EUnit(i)	Method property
0			
1			
2			
3	E1(measure)	V	E1
4	E2	V	E2
5	E3	V	E3
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0			
1			
2	t run	s	Runtime
3	t1		
4	t2		
5	t3	s	ConditioningTime
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime

MultiplePulseAmperometry [12]

Class: PalmSens.Techniques.MultiplePulseAmperometry

i	ELabel(i)	EUnit(i)	Method property
0			
1			
2			
3			
4			
5			
6	E dep	V	DepositionPotential
7	E stby	V	StandbyPotential
8	E stby	V	StandbyPotential
i	tLabel(i)	tUnit(i)	Method property
0	interval	s	Intervaltime
1			
2			
3			
4	levels		nLevels
5			
6	t dep	s	DepositionTime
7	t eq	s	EquilibrationTime
8	cycles		nCycles

NOTE ON SAVING THE METHOD FILE

In order to specify the different levels, the property E() and t() are used. These values are not saved in the default .PMT file. Instead an extra .IMT file is created with the together with the .PMT file, when using the `.Save(string_filename)` method in class `PalmSens.DataFiles.TimeMethodFile()`