
Nokia Energy Profiler

External APIs

Version 1.2
April 1, 2009

S60 platform

Legal notice

Copyright © 2009 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this document at any time, without notice.

License

A license is hereby granted to download and print a copy of this document for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1.	Introduction	5
2.	Installation	6
3.	Control API	6
3.1	Measurement control.....	7
3.1.1	Connect	7
3.1.2	Disconnect	7
3.1.3	Start Measuring.....	7
3.1.4	Stop Measuring	7
3.1.5	Start Recording	8
3.1.6	Stop Recording	9
3.1.7	Take Screenshot	9
3.1.8	Set Marker	9
3.1.9	Set HintState.....	10
3.2	Data acquisition.....	11
3.2.1	Callback-interface.....	11
3.2.2	Data structures.....	11
4.	Plug-In API	13
4.1	Data plug-in.....	13
4.1.1	Interface.....	13
4.1.2	Callback Interface.....	15
4.2	Statistics plug-in	15
4.2.1	Interface.....	15
5.	Summary	18
6.	Terms and abbreviations	19
7.	References	20

Change history

April 1, 2009	Version 1.2	Updated to version 1.2.
October 3, 2008	Version 1.0	Initial document release.

1. Introduction

Nokia Energy Profiler is a Series 60 application that is used to monitor and record mobile phone power consumption, battery current, and voltage over time. This document describes the external features of NEP. NEP provides interface functions starting with the name 'Juice'.

Currently, NEP offers the Control and Plug-in APIs that you can use to extend its functionality. These APIs are described in the following.

The Control API allows you to programmatically control NEP functionality from your application e.g. to start and stop measurements as well as to send the measurements to your application in real-time. These can be useful in case you want to perform some automated energy profiling or if you want your application to react to energy consumption in real-time. Note that you will have to manually start up the NEP application in order for the Control API to work.

The Plug-in API allows you to extend NEP's functionality by providing additional data to NEP's existing view collection. The Plug-in API allows you to provide both data measurement plug-ins as well as statistics plug-ins. Data measurement plug-ins add measurement views to NEP, which you can browse along with all of NEP's standard views. Statistics plug-ins allow you to add statistics views, which will appear along NEP's standard statistics under *Tools->Statistics*.



Note: For the Control API to work, the Nokia Energy Profiler application needs to be already running. The API does not currently start up the application by itself.



Note: The plug-in API section is written with assumption that the reader already knows how to develop Symbian-based ECOM plug-ins.

2. Installation

Follow the steps below in order to install the Nokia Energy Profiler APIs to your SDK.

1. Unzip the `NokiaEnergyProfilerAPI_v1.2.zip`
2. Copy the necessary header files (the *inc* folder under the *ControlAPI* and *PluginAPI* folders) to any of your SDK include folders. This can be your own user-defined folder under the include folder of your application. If you are planning on using the APIs on a regular basis, it might be a good idea to copy the header files under the `/epoc32/include/` folder of your SDK. Note that the header files have been split into the Control and Plug-in APIs for your convenience; you might want to use both.
3. Copy the necessary library files to your SDK. If you require the Control API, the contents of the *ControlAPI/release/* folder need to be copied under the `/epoc32/release/` folder of your SDK.

The Nokia Energy Profiler APIs are then installed to your SDK. You can take a look at the examples under the *Examples/* folder for a practical approach on how to use the APIs.

3. Control API

NEP provides *CJuiceExternalApi* class for both sending commands to NEP and receiving data measurements from NEP. NEP must be running in order to connect to it.

Through *CJuiceExternalApi* class, an external third party application can command NEP to start and stop measuring, start and stop recording, take a screenshot and set a marker or state hint on the current position.

Start and stop measuring do not affect NEP's user interface (UI). Instead, these functions signal NEP to start or stop sending measurement data samples to a third party application using the sampling rate configured through NEP's Setting dialog. Note that in the scope of this document, *measuring* refers to NEP sending data samples to an external application, whereas *recording* refers to NEP starting a measurement on its own UI, with NEP taking care of the data management. Measuring and recording are independent of each other, and you can request to start measuring while Juice is recording and vice versa.

Third party applications can receive measurement samples from NEP by passing a pointer to their own implementation of the *MJuiceMeasurementSampleReaderIf* interface to the *NewL()* function of *CJuiceExternalApi*. An external application can then request NEP to start sending measurement samples using the *StartMeasuring()* function. Measurement samples are made available to applications through the *MJuiceMeasurementSampleReaderIf* interface.

Passing a NULL pointer to *NewL()* when creating an instance of *CJuiceExternalApi* will result in the application not being able to receive data samples from NEP.

3.1 Measurement control

3.1.1 Connect

Connect() must be called before using any command functions. The call informs NEP that the application is connected and ready to send commands.

Any command functions called without a previous call to *Connect()* will return with error code *KErrNotReady*.

3.1.2 Disconnect

Disconnect() will disconnect from NEP. If a *CJuiceExternalApi* instance is deleted without a prior call to *Disconnect()*, the class' destructor will automatically disconnect from NEP if connected.

3.1.3 Start Measuring

StartMeasuring() enables NEP to start sending measurements (current, capacity, voltage, power and signal levels) using the currently enabled sampling rate settings. Note that this function signals NEP to start sending samples to your application but NEP itself will not be recording the data. Your application is responsible for handling the data samples itself. *StartMeasuring()* will not affect the state of other applications using the NEP APIs. It is possible to signal NEP to record data through the *SatrtRecording()* function.

Signal TX/RX measurements will be published only when signal view is enabled on NEP settings.

This will not affect NEP's UI recording state.

If this function is called before *Connect()*, it will return with *KErrNotReady* error.



Note: After requesting it through the *StartMeasuring()* function, measurement data can be accessed through the *MJuiceMeasurementSampleReaderIf* interface, as explained in section 3.2.

3.1.4 Stop Measuring

StopMeasuring() will signal NEP to stop sending measurement sample data to your application. Note that this function has no effect on NEP's own data recording. This function has no effect on other applications receiving measurement data from NEP through the same API.

This will not affect NEP's UI recording state.

If this function is called before *Connect()*, it will return with *KErrNotReady* error.

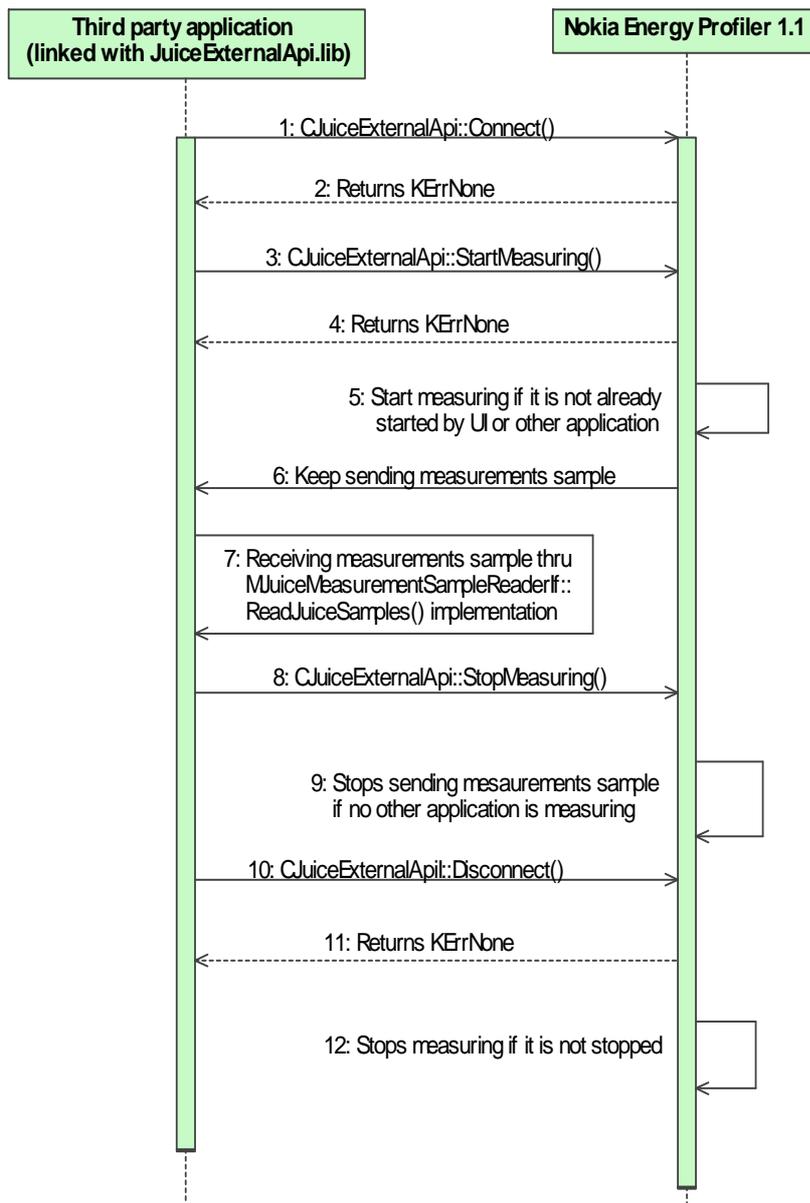


Figure 1. Sequence diagram for the basic measurement functions

3.1.5 Start Recording

StartRecording() signals NEP to start recording measurement data. This has the same effect as using the *Start* option from NEP’s UI. If NEP is already recording, this function has no effect. *StartRecording()* will start recording using the current settings.

If this function is called before *Connect()*, it will return with *KErrNotReady* error.

	<p>Note: <i>StartRecording()</i> will not signal NEP to send any measurements to your application. For that purpose, <i>StartMeasuring()</i> should be used instead.</p>
---	---

3.1.6 Stop Recording

StopRecording() will signal NEP to stop recording only when recording is started through *StartRecording()*. In case recording is started using NEP's UI, this function has no effect.

If this function is called before *Connect()*, it will return with *KErrNotReady* error.

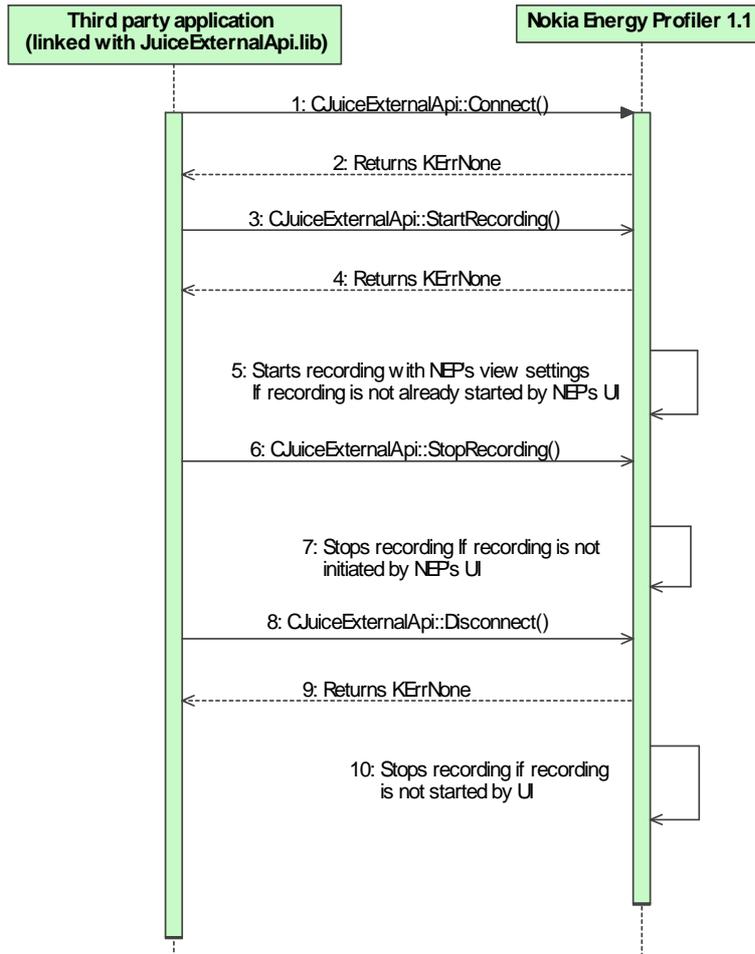


Figure 2. Sequence diagram for the recording functions

3.1.7 Take Screenshot

TakeScreenShot() will trigger NEP to take a screenshot.

This event will be discarded if NEP's UI is not in recording mode.

If this function is called before *Connect()*, it will return with a *KErrNotReady* error.

3.1.8 Set Marker

SetMarker () will trigger NEP to set a marker at the current position.

This event will be discarded if NEP's UI is not in recording mode.

If this function is called before *Connect()*, it will return with a *KErrNotReady* error.

3.1.9 Set HintState

SetState() sends a hint state to NEP so that it can be stored to the JCE file using the current timestamp. A valid hint state may consist of a maximum of 512 Unicode characters. Hint states are saved into JCE files and can be exported to .CSV files. The current version of NEP does not display hint states in the UI.

Since hint states may be exported to a .CSV file, the hint state message should not contain any special characters used in the CSV file format. Refer to the CSV file format documentation (<http://rfc.net/rfc4180.html>) for more information.

If this function is called before *Connect()*, it will return with *KErrNotReady* error.

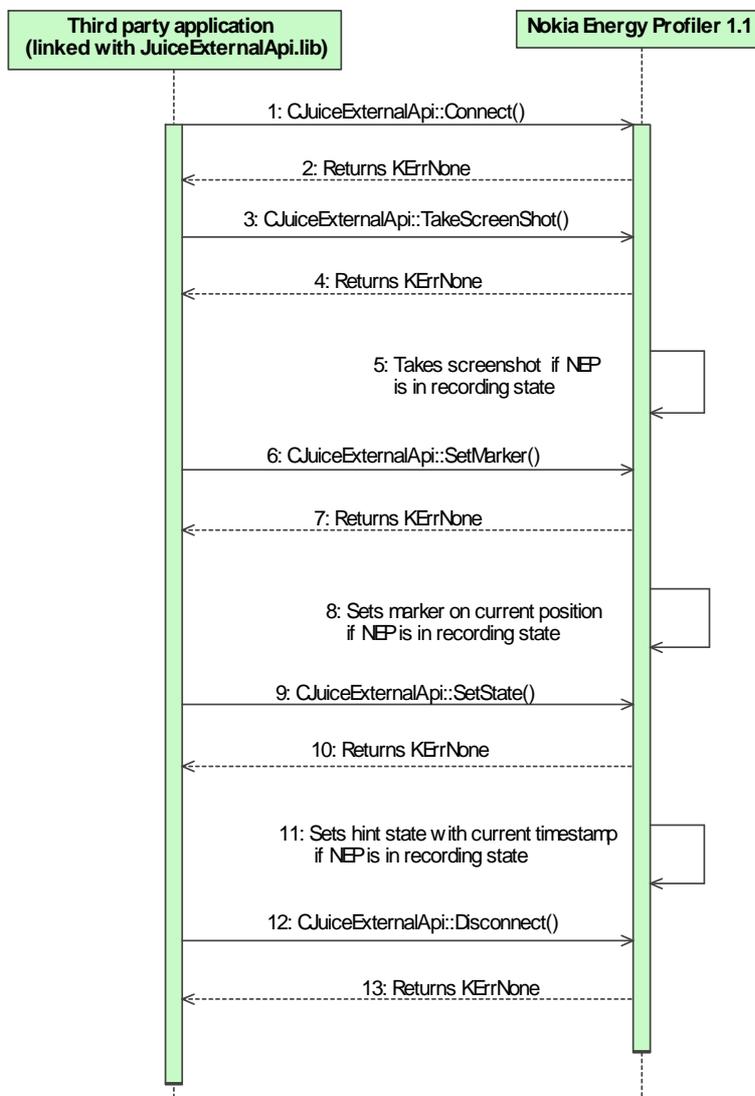


Figure 3. Sequence diagram for additional functions

3.2 Data acquisition

NEP measurements sample data can be received by third party applications through the *CJuiceExternalApi* class. Clients to this class must pass a pointer to their own implementation of *MJuiceMeasurementSampleReaderIf* when using the *CJuiceExternalApi::NewL()* function and upon calling the *StartMeasuring()* function.

3.2.1 Callback-interface

MJuiceMeasurementSampleReaderIf provides the pure virtual function *ReadJuiceSamples()* with the *TJuiceMeasurementSamples* data structure to receive NEP measurement samples.

ReadJuiceSamples

This function must be implemented by your application to receive NEP measurements. This implementation will be called by the framework whenever new measurement samples are received from NEP, provided measuring was started through the *StartMeasuring()* function.

3.2.2 Data structures

The following data structures are used to receive data from NEP.

3.2.2.1 *TJuiceMeasurementSample*

This structure contains a signed integer '*iSampleValue*' holding a measurement sample and an unsigned integer '*iSampleTime*' holding its timestamp in milliseconds.

3.2.2.2 *TRRCState*

This is an enumeration for different network modes. Attribute *iNetworkMode* in *TJuiceMeasurementSamples* contains an integer value that corresponds to this value.

3.2.2.3 *TJuiceMeasurementSamples*

This structure contains several instances of *TJuiceMeasurementSample* holding different NEP measurements. Each measurement sample may use a different unit scale value.

iCurrent

Current sample, in milliampere (mA).

iCapacity

Capacity sample, in milliampere-hours (mAh). This value is measured only once upon NEP startup.

iTemperature

Temperature sample value.

(This is reserved for future use as NEP 1.1 does not support temperature measurements)

iVoltage

Voltage sample value, in millivolts (mV).

iPower

Power sample value, in microwatts (μ W).

iTxStrength

Signal transmission (TX) strength, in dBm.

The default value will be *KMinTInt* if signal strength is not measured or unavailable.

iRxStrength

Signal reception (RX) strength, in dBm.

The default value will be *KMinTInt* if signal strength is not measured or unavailable.

iWlanRssi

Wlan received signal strength, in dBm.

The default value will be *KMinTInt* if signal strength is not measured or is unavailable.

iWlanMode

Integer value corresponding to enumeration *TWlanConnectionMode* in wlan management API.

iCpuKernel

Kernel CPU usage, in per cent.

iCpuUser

User CPU usage, in per cent.

iMemHeap

Heap usage, in bytes.

iMemStack

Stack usage, in bytes

ilpUp

Data transfer out, in kB/s.

ilpDown

Data transfer in, in kB/s.

iNetworkMode

Integer value corresponding to enumeration *TRRCState*.

iEnergy

Energy consumed, in mAh.



Note: Signal TX/RX, WLAN, processor, RAM memory, network speed and network mode measurement values are updated only when the corresponding view is enabled in NEP's settings.

4. Plug-In API

4.1 Data plug-in

NEP provides the *CJuiceMeasurementDataProvider* ECOM interface for developing data measurement plug-ins. *CJuiceMeasurementDataProvider* is derived from *CActive* to support asynchronous sending of sample data to NEP at regular intervals.

Plug-ins should implement this interface with their unique implementation ID to create a new data plug-in. Plug-ins are expected to obtain and send measure data asynchronously. Measurement data must be sent through the plug-in's *RunL()* implementation. NEP also provides the *MJuiceBeating* interface to synchronize plug-in measurements with NEP's battery current measurements. Plug-ins may implement *MJuiceBeating* in order to synchronize their measurements with NEP's battery current measurements. Synchronizing with NEP has the advantage of reducing the number of measurement wake-ups, thereby reducing the overall measurement overhead.

NEP uses the ECOM framework to find available plug-in implementations. For each plug-in implementation found, NEP creates a new measurement graph at runtime to plot plug-in measurement data.

For more hands-on information on each function, you can check the example *MeasurementPlugin* plug-in implementation [2].

4.1.1 Interface

4.1.1.1 Virtual functions

CJuiceStatisticsViewProvider interface provides the following pure virtual functions that plug-in developers must implement.

StartTracingL

This function is called by the NEP framework whenever NEP starts recording if the given plug-in is enabled in NEP settings.

StopTracingL

This function is called by the NEP framework whenever NEP stops recording.

MinValue

This function must provide the minimum possible value of a data sample without scaling. The NEP framework makes use of this function to correctly display the vertical axis' minimum value for the plug-in graph.

MaxValue

This function must provide maximum value of sample without scaling, i.e. at the finest zoom level. The NEP framework makes use of this function to correctly display the vertical axis' maximum value for the plug-in graph. *MaxValue* is used to determine the various Y-axis zoom levels for the graph.

UnitScale

This function must provide the scaling factor to be used for showing graphs and plotting samples into graphs.

GetUnitName

This function must provide the name of the measurement units. The NEP framework uses this function to display the right unit name in the graph and statistics for the corresponding measurement.

GraphColor

This function must provide the color to be used for drawing the measurement graph.

ValueFormat

This function must provide the formatting for the measurement values (as a *TRealFormat*) to be used for showing the scaled sample values.

AxisFormat

This function must provide the formatting for the vertical axis values (as a *TRealFormat*) to be used for displaying the axis' values in graph.

4.1.1.2 Public functions

JuiceHeartBeatObserver

The NEP framework uses this function to get a pointer to the plug-in's implementation of the *MJuiceBeating* interface. NEP data measurement plug-ins may implement this function if they want to synchronize their measurements with NEP's own data measurements and thereby reduce measurement overhead. The default implementation of this function returns a NULL pointer.



Tip: *MJuiceBeating* provides the pure virtual function *JuiceHeartBeatEvent()* to synchronize with NEP Current measurement sample. *JuiceHeartBeatEvent()* will be called by NEP whenever it receives battery current measurement so that plug-in measurements can be synchronized with NEP measurements.

JuiceHeartBeatEvent

This function defines the logic for sending the latest plug-in measurement sample to NEP asynchronously. This interface is called by the NEP framework whenever it receives a battery current measurement sample.

4.1.1.3 Protected functions

MeasurementId

This function returns the NEP assigned measurement ID for this plug-in implementation. The plug-in must use this ID when sending measurement data to NEP through the *WriteData()* function.

4.1.2 Callback Interface

NEP provides a pointer to its *MMeasurementDataCollectorIf* callback interface implementation. Plug-ins shall use this callback interface to send their measurement data samples to NEP, along with the plug-in's measurement ID, as provided by the *MeasurementId()* function.

4.2 Statistics plug-in

NEP provides the *CJuiceStatisticsViewProvider* ECOM interface for statistics plug-in development. Statistics plug-ins shall implement this interface to create a new statistics view page with the plug-in name. NEP will create a new statistics page for each statistics plug-in implementation found through the ECOM framework. NEP provides statistics plug-ins with a bitmap context, statistics data and active region data for active measurement views through the *WriteStatistics()* function. Plug-ins can draw their own data representations on the bitmap context by using the active region measurement data.

The *MJuiceStatisticsDataIf* interface includes functions to retrieve data from the current active region for statistics calculation.

4.2.1 Interface

4.2.1.1 *CJuiceStatisticsViewProvider*

The *CJuiceStatisticsViewProvider* interface provides the following pure virtual functions for plug-in developers.

WriteStatistics

WriteStatistics() is the function controlling the output of the statistics plug-in. It provides two parameters:

- Pointer to the bitmap context which has been created by NEP for the plug-in to write its own representation. The bitmap context has background and pen color attributes from current theme. So the plug-in need not set any pen color or background attributes unless there is a special need.
- Reference to NEP's implementation of the *MJuiceStatisticsDataIf* interface.

4.2.1.2 *MJuiceStatisticsDataIf*

The *MJuiceStatisticsDataIf* interface provides the following functions to access NEP statistics and active region data.

JuiceStatisticsValues

This function provides statistics that are already calculated by NEP based on current active region data. All values are calculated with scaling factor. This function will return *TJuiceStatisticsValues*.

MoveFirst

NEP provides an array of sample values that are available for the active region. This function moves the iterator to first sample of current active region. *SampleValue()* and *SampleTime()* must be used to access the sample data from the current position.

The function returns the error code for this operation.

MoveNext

Moves the iterator to the next sample for the current active region. *SampleValue()* and *SampleTime()* must be used for getting the sample data from the current position.

This function returns the error code for this operation.

SampleCount

This function returns the total number of samples available on the current active region. Plug-ins shall use this to iterate through all the samples in the current active region.

SampleValue

This function returns the sample value in the current position without scaling.

SampleTime

This function returns the timestamp for the current sample.

ScalingFactor

This function provides the scaling factor used for the measurement in NEP. This can be multiplied with *GetSample()* to get a scaled sample value.

UnitName

This function returns the measurement unit name for the available measurement data.

4.2.1.3 *TJuiceStatisticsValues*

The *TJuiceStatisticsValues* structure holds following values. All sample values are scaled and with respect to the current active region.

iSampleMean

Mean sample value

iSampleMax

Scaled maximum value received.

iSampleMin

Scaled minimum value received.

iSample10Percentile

10% of values are smaller than this value.

iSampleFirstQuartile

25% of values are smaller than this value.

iMedian

50% of values are smaller than this value.

iSampleThirdQuartile

75% of values are smaller than this value.

iSample90Percentile

90% of values are smaller than this value.

iActiveTime

Active time (backlight on) in seconds.

iStandbyTime

Standby time (backlight off) in seconds.

iMinValue

Value that will be used for minimum sample range (e.g. in the histogram and box plot pages)

iRange

The difference between the minimum and maximum values for range calculation.

iRangeFormat

Formatting used for representing the range value.

For more hands-on information on each function, refer to the example *StatisticsPlugin* plug-in implementation [2].

5. Summary

This document has presented the available external interfaces to the Nokia Energy Profiler tool. These interfaces enable 3rd party developers to both control the operation of NEP programmatically and provide additional data for display on NEP.

Please send your comments, feedback and suggestions for improvement to energyprofiler@nokia.com. We'd also like to hear about your success stories and cool applications of these APIs!

6. Terms and abbreviations

Term or abbreviation	Meaning
API	Application Programming Interface
NEP	Nokia Energy Profiler

7. References

- [1] NEP Measurement Plug-In Example. Available at [Examples/MeasurementPlugin/](#)
- [2] NEP Statistics Plug-In Example. Available at [Examples/StatisticsPlugin/](#)