

DEWESoft

Data acquisition, processing, analyzing and storage software

DEWESoft X Tutorials

(RC)



地址：北京市海淀区小营西路27号金领时代大厦12层
电话：136 1171 664；010-5361 2036
传真：010-5635 3026
网站：www.chinaksi.com
电邮：ksi@chinaksi.com

General information

Software version

DEWESoft Tutorials corresponds with software version X.

Printing notice

Specifications subject to change without notice.

Copyright notice

© 2013 DEWESOFT d.o.o. All rights reserved. May not be duplicated or disseminated in any fashion without the express written permission of DEWESOFT d.o.o.

Trademark notice

All trademarks are acknowledged to be the property of their owners, with all rights and privileges thereto. No infringement is intended.

Microsoft, Excel and Word are registered trademarks of Microsoft Corporation.

Disclaimer

DEWESOFT makes no claim about the efficacy or accuracy of the information contained herein.

Use of this manual is entirely at the user's own risk. Under no circumstances will DEWESOFT assume any liability caused by the use, proper or improper, of this manual or the information, textual, graphical or otherwise, contained within it.

地址：北京市海淀区小营西路27号金领时代大厦12层

电话：136 1171 664；010-5361 2036

传真：010-5635 3026

网站：www.chinaksi.com

电邮：ksi@chinaksi.com



地址：北京市海淀区小营西路27号金领时代大厦12层

电话：136 1171 664；010-5361 2036

传真：010-5635 3026

网站：www.chinaksi.com

电邮：ksi@chinaksi.com

Table of Contents

| | |
|---|-----------|
| Tutorials | 1 |
| Basic tutorials | 2 |
| Simple measurement | 2 |
| Channel setup | 3 |
| Video setup | 4 |
| First measurement | 5 |
| Making a good screen | 7 |
| Analysis | 7 |
| Reporting and exporting | 9 |
| Display settings | 11 |
| Post processing | 16 |
| Storage options | 20 |
| Always fast | 21 |
| Always slow | 22 |
| Fast on trigger | 24 |
| Fast on trigger, slow otherwise | 27 |
| Measurement tutorials | 29 |
| Voltage and current | 30 |
| Channel setup | 31 |
| Measurement | 33 |
| Glitch triggering | 35 |
| Triggered storing | 38 |
| Voltage/current/digital output sensors | 41 |
| Channel setup | 42 |
| Measurement | 44 |
| Analysis | 44 |

| | |
|-------------------------------------|------------|
| Temperature | 46 |
| Channel setup | 47 |
| Measurement | 48 |
| Vibration | 50 |
| Channel setup | 53 |
| Math setup | 57 |
| Video setup | 59 |
| Measurement | 59 |
| Analysis | 61 |
| Strain measurement | 63 |
| Experiment setup | 66 |
| Quarter bridge setup | 67 |
| Quarter bridge measurement | 68 |
| Full bridge setup | 72 |
| Full bridge measurement | 74 |
| Load cell setup | 76 |
| Counter | 77 |
| Event counting | 77 |
| Simple event counting | 78 |
| Gated event counting | 80 |
| Up/down counting | 82 |
| Encoder | 83 |
| X1, X2, X4 modes | 84 |
| Encoder with zero pulse | 86 |
| Period and pulse-width | 88 |
| Period measurement | 88 |
| Pulse-width measurement | 90 |
| Duty cycle measurement | 91 |
| Frequency / super-counter | 92 |
| Sensor measurement | 95 |
| Counters in automotive applications | 96 |
| Video acquisition | 101 |
| Channel setup | 102 |

| | |
|--|------------|
| Measurement | 104 |
| Video post synchronization | 107 |
| CAN bus acquisition | 109 |
| Channel setup | 109 |
| Measurement | 112 |
| GPS acquisition | 114 |
| Channel setup | 115 |
| Measurement | 116 |
| Power module tutorials | 119 |
| Single phase power measurement | 119 |
| Channel setup | 121 |
| Measurement | 123 |
| Sensor correction | 126 |
| Dynamic signal analysis tutorials | 129 |
| Sound level | 129 |
| Channel setup | 130 |
| Microphone calibration | 131 |
| Measurement | 133 |
| Torsional vibration | 134 |
| Rotational vibration setup | 135 |
| Rotational vibration measurement | 136 |
| Rotational order extraction | 137 |
| Torsional vibration setup | 139 |
| Torsional vibration measurement | 140 |
| Torsional order extraction | 141 |
| Human vibration | 142 |
| Input channel setup | 143 |
| Output channel setup | 145 |
| Measurement | 146 |
| Order tracking | 147 |

| | |
|--|------------|
| Channel setup | 148 |
| Measurement | 150 |
| Combustion analysis | 152 |
| Channel setup | 152 |
| Combustion setup | 154 |
| Measurement | 161 |
| Analysis | 162 |
| Networked data acquisition tutorial | 163 |
| Setup Measurement Units and Client | 164 |
| Remotely Controlling a Slave MU | 168 |
| Channel Setup on the MU | 170 |
| Display Setup on the MU | 170 |
| Transfer Setup of a MU | 171 |
| Measurement | 172 |
| Analysis | 175 |
| Index | 177 |

Tutorials

DEWESoft X provides you with **basic tutorials**, which describe procedures for working with certain parts of the program:

Basic tutorials

- **How to perform simple measurements in DEWESoft?**
- **Display settings**
- **Post processing**
- **Storage options**

Measurement tutorials

- **Voltage and current**
- **Sensors with voltage output**
- **Temperature**
- **Vibration**
- **Strain gage**
- **Counter**
- **Video acquisition**
- **CAN bus measurement**
- **GPS acquisition**

Power tutorials

- **Single phase power**

Dynamic signal analysis tutorials



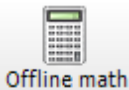

- **Sound level**
- **Torsional vibration**
- **Human vibration**
- **Order tracking**

Combustion analysis

Networked data acquisition tutorial

1 Basic tutorials

Basic tutorials are a "pre-requisite" of sorts for first time users. It gives an overview of how to perform **basic measurement tasks**.

| | | |
|--|--|--|
|  | How to perform simple measurement in DEWESoft? | This tutorial explains basic procedures for working with DEWESoft : setup, acquisition and storage of data, reviewing the results. |
|  | Display settings | This tutorial explains how to design custom screens . |
|  | Post processing | This tutorial explains how to add the mathematic on to already stored data files. |
|  | Storage options | This tutorial explains how to store the data ; when and how to use different storage options. |

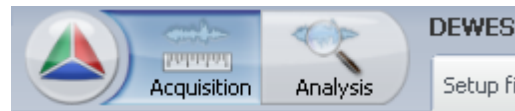
1.1 Simple measurement

This tutorial will first take a look at how to do a simple measurement with **DEWESoft**. The hardware used is have a *sound card* and a *web cam*. To follow this tutorial, the user should select **Audio card** in the **Analog** section of the **Settings** → **Hardware setup...**, *plug in* the web cam and choose **DirectX** camera in the **Video** section of the **Hardware setup**. Even though there is not much instrumentation used in this example, this gives the user a good basic picture of what can be done with this software.

The following table displays the required hardware and software for completing this tutorial.

| | |
|--------------------------|--|
| <i>Required hardware</i> | Sound card, web cam |
| <i>Required software</i> | Any license |
| <i>Setup sample rate</i> | At least 1 kHz (the setup sample rate is chosen in Settings -> Global setup -> General -> Setup sample rate; some math modules require higher rates) |

Two main icons found on the upper left side are **Acquisition** and **Analysis**. These are used to switch between *Acquisition mode* to **acquire the data**, and *Analysis mode* to **process the data**.



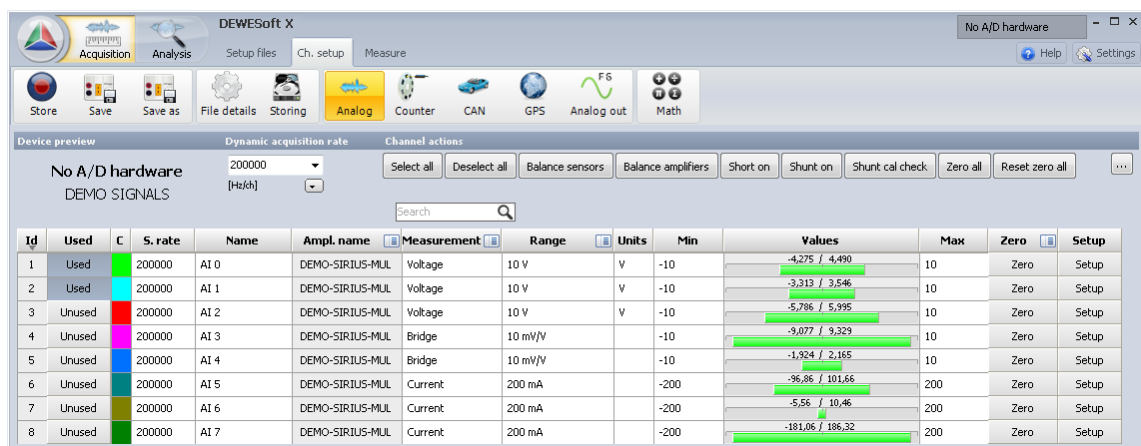
When **DEWESoft** is started, it is in the Acquisition mode already, so let's take a look how to **setup the channels**.

Also in this section:

- **Video setup**
- **First measurement**
- **Making a good screen**
- **Analysis** and
- **Reporting and exporting**

1.1.1 Channel setup

Click the **Ch. setup** button. This opens the **channel setup** where we see the **Analog** section with two analog channels (stereo), a **Video** section and **Math** section.

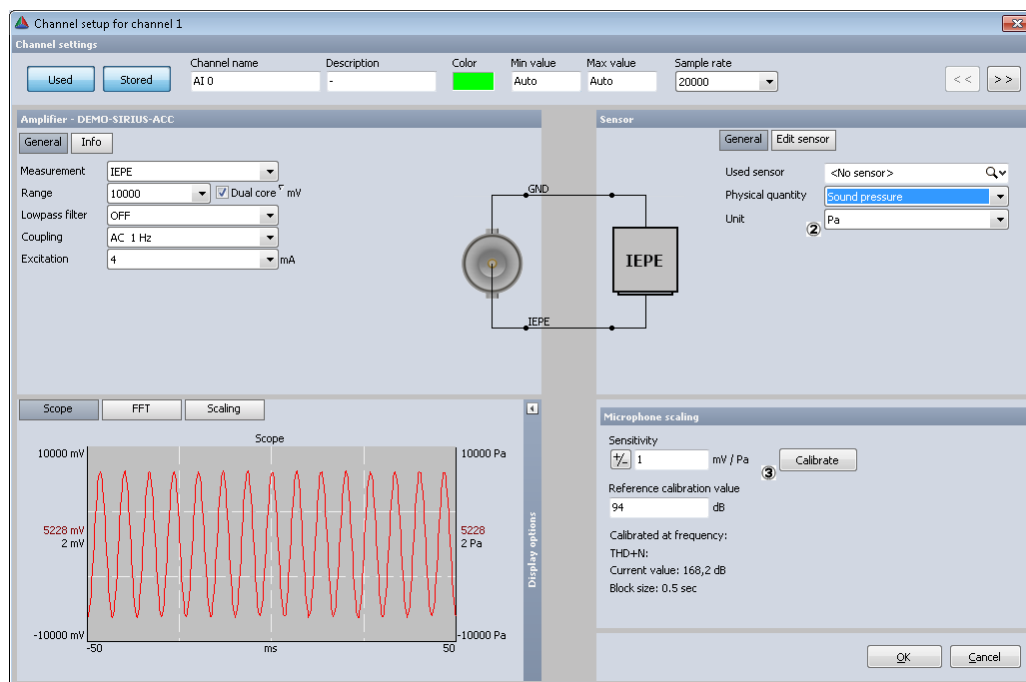


In the Storing section we can enter the **file name** and **path** where the files will be stored, in the Analog section is **sample rate** ①. A sample rate is the basic speed with which data will be acquired. This rate needs to be chosen wisely according to the specific situation. If *temperatures* are measured, we might need only **100 Hz** (samples per second) or even less. If the number chosen is too high, we will waste disk space, yet gain no additional information.

In this case that we will measure *sound*, so we have to choose at least a sample rate of **10 kHz** (10000 values per second) to really acquire sound correctly. For a good *audio performance* we should choose **40 kHz** or more.

Next, the appropriate channels need to be selected. Since only a simple microphone is used for this example, only one channel is sufficient. The user can select it by clicking the **on/off** button. Note there are **Setup** buttons on the right side of the list.

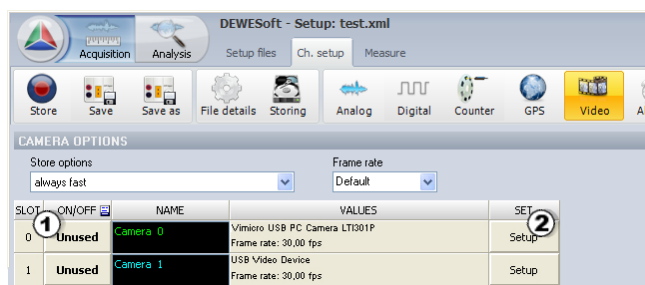
Even though this is a simple microphone, the importance of correct measurement should be emphasized. First we need to define the **Units** ② of measurements. The physical quantity measured by a microphone is a *sound pressure* measured in **Pascal**. **Pa** should be entered as the unit.



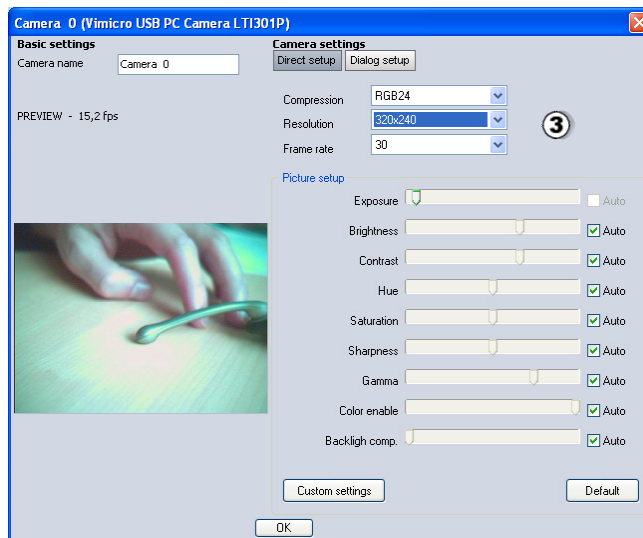
Next we will try to perform a simple **"calibration"**. Sound is calibrated with a sine wave. We can try to make a nice sine wave with a whistle. The RMS value of the signal grows to a certain level and we can use the **Calibrate from RMS** button ③ to equalize the level to what might be approximately **1 Pa**. A real microphone is calibrated exactly with the same method as used in this tutorial, but we can trust that the calibration value will be 1 Pa with the real microphone and calibrator.

1.1.2 Video setup

The next step is **setup** the *video channel*. Go to **Video** section. In the **CAMERA OPTIONS** section where the web cam should be displayed if it is installed correctly. Click the **Unused** ① button to switch on the camera and end setup with the **Setup** button ②.



Here, we can change the *frame rate*, *resolution*, *compression* and other *properties* ③ of the camera. Details about camera setup are explained in the **Video acquisition tutorial**, but let's stay with the simple setup at the moment.



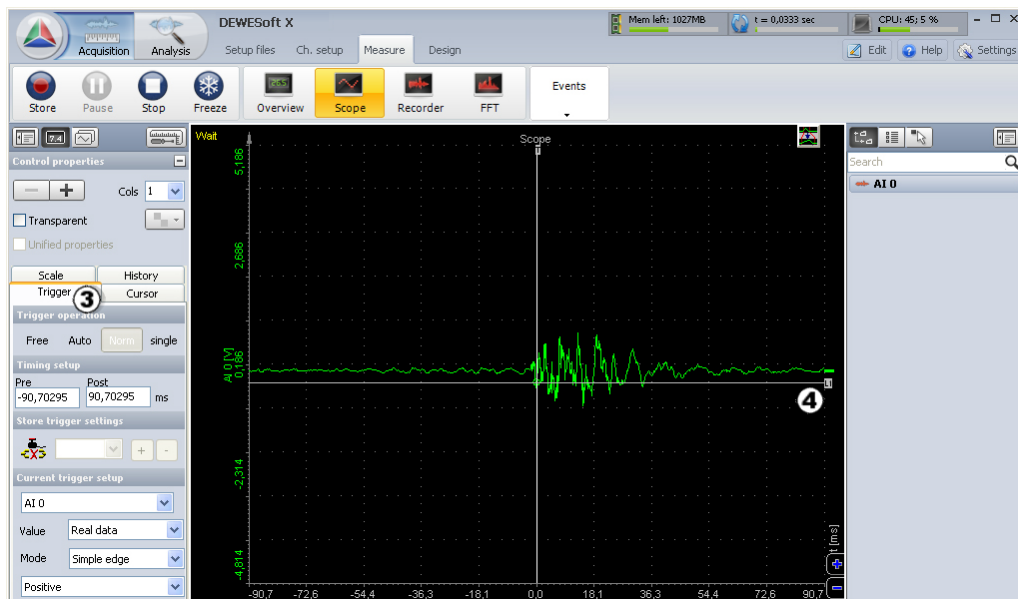
We are now ready to take a first measurement.

1.1.3 First measurement

Click the **Recorder** button ① in the upper section. The Settings will start to **acquire the data**. We can now test how the microphone works by making some sounds. We can use the **plus** or **minus** button ② on the lower right side of the display to **zoom in** or **out** of the time axis.



To see the data at the *faster rate*, switch to the **Scope** screen. On the **trigger** tab, select the **Norm** trigger ③ and drag level **L1** ④ with mouse to about **10 %** of the full range. One can now tap on the microphone to create trigger shots. It will be nicely distinguishable how different sounds create different patterns.



Next the **FFT** screen can be used to perform the *frequency analysis*. A nice line resolution and **logarithmic** ⑥ scale is helpful to better see the harmonics (in the screen shot below there are **4096 lines** ⑤ to have approximately **1 Hz** between each frequency line).

Here you can try to sing your best A tone (A3 has a frequency of **220 Hz**, A2 has a frequency of **110 Hz**) or, if you have a piano or any other musical instrument is available, one can see how well it is tuned. It is also nice to see the difference between a whistle, singing and a musical instrument. Actually, the *higher* harmonics give the *color* of the tones of music.



Now you can go back to the *recorder* and see that all the data is still there. You can zoom out and observe the full period of time you someone have played with the microphone.

The next step is to store some data. Click the **Store** button ⑦ and the **red** light on the "store" button should appear. The upper right corner should display the *size* of stored *analog data* and the *video* file size.

1.1.4 Making a good screen

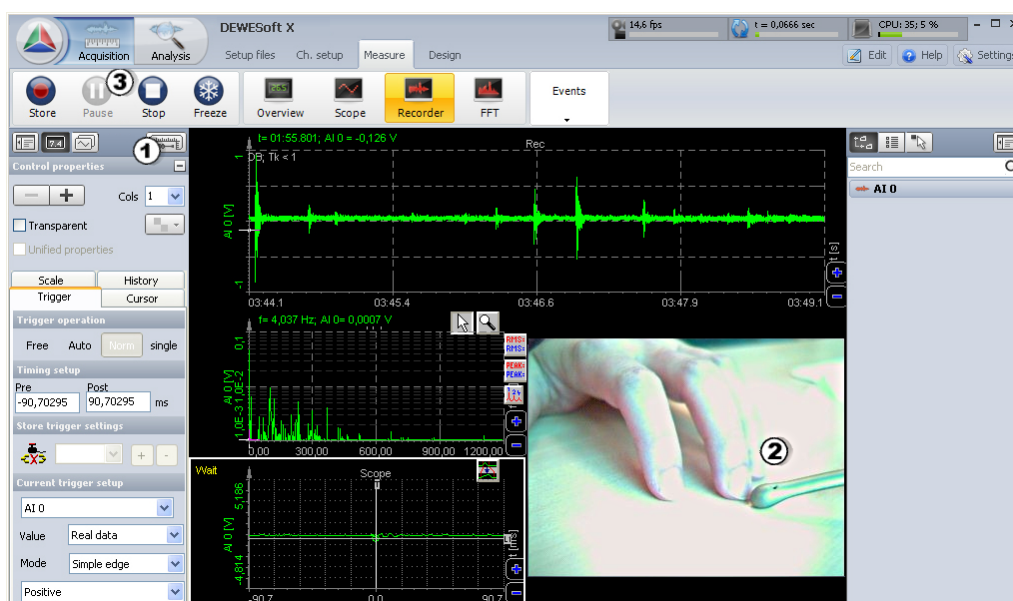
The previous example was about how to use separate **instruments**. In **DEWESoft**, it is possible to combine those instruments to create great looking displays. Actually, any display can be changed according to the user's wishes. Let's change the **Overview** display, which is intended for combining instruments.

First, please note the **Design** button ①, which is located right next to **Measure**. When Design is clicked, the bar with available instruments appears. We can **add** all the instruments that were used independently before. Let's add a *recorder*, *scope* and *FFT*. We can *drag* and *resize* the instruments to fit our needs.

Don't forget - we have also a *video*. Click on the icon with the camera roll ("Video" icon) on this bar to add the **video** screen ②.



To *zoom in/out* or to change the *trigger levels*, the user needs to *quit* the design mode by clicking the **Design** button *again*. This removes the instrument bar and *enables* the user to utilize full *functionality* of the displays. Now, one can practice setting these basic instruments to have a nice overview of the measurements.



Remember that we are storing the data? Click **Stop** button ③ to stop storing.



1.1.5 Analysis

The next step is to look deeper how we can **analyze** the **data** being stored.

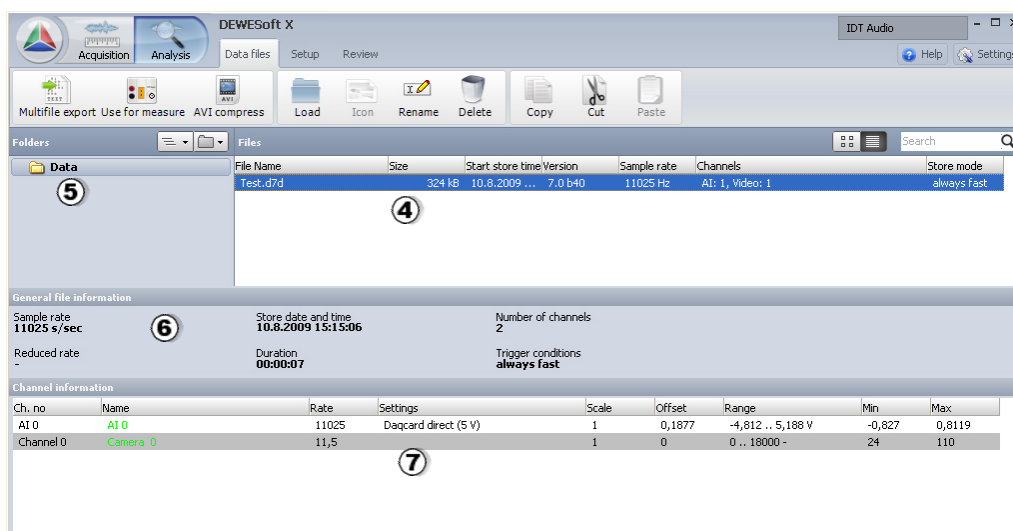
It's quite simple - just click **Analysis** button. Since we *just stored* a data file, **DEWESoft** assumes *this is the file* that we want to **review**. If the user click **Analysis** in other cases, (when launching **DEWESoft** or from the setup screen), the *file explorer* would open up ④ and we can see the *sub folders* of the main data folder, the *file list* and *detailed information* about the chosen file such as size, start store time, version, sample rate, number of channels, storage options and video files (if there are any).

Double click on the file which was previously stored.

If there are *sub folders*, we can select them by double clicking on the sub folder ⑤.

The first *level data folder* can be *changed* by clicking on the folder list drop down  and selecting any upper level folder. The default folder can be *saved* by clicking on folder drop down  and selecting *Set as default project folder*.

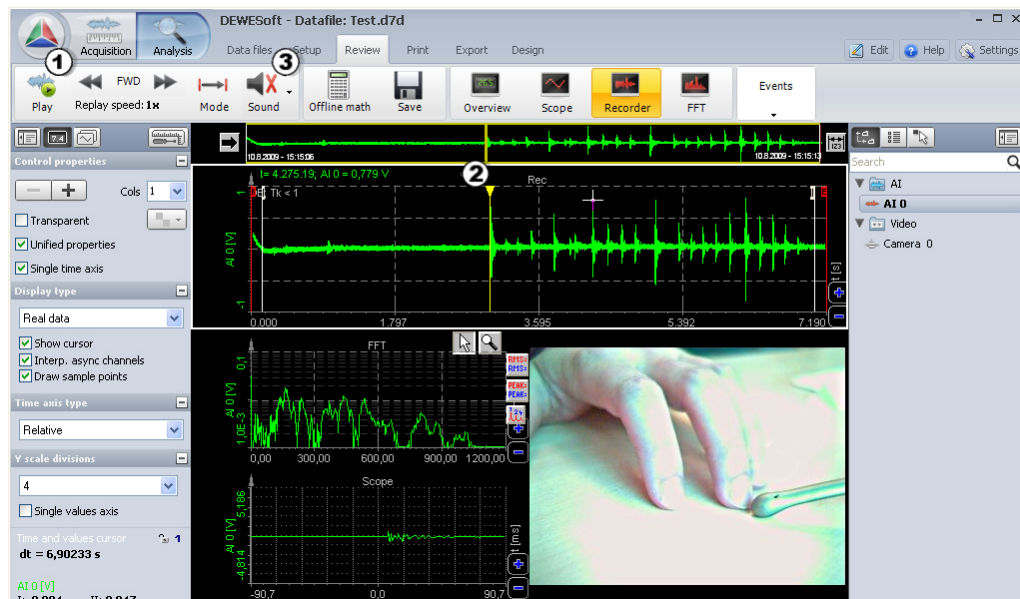
When the user selects certain file, it is pre-scanned to display *important information* about channels, events and data header. This information is shown in the General file information section ⑥ and on the Channel info list ⑦ below the file explorer. Note that there is also a *Min* and *Max* available, and that if the channel is *overloaded*, this will be shown in red



Now we can **open** a file. The user can choose *any display* that was *pre-built* in Acquisition mode.

There are many ways how to **review** the data file. On the upper right side there is a **Play** button ①. When selecting this, you will see that the *yellow* cursor ② moves through the data, FFT is calculated, *scope* shows the *current* data and the video file is replayed. When the replay starts, "play" button *changes* to a **Stop** button. Clicking to this button **stops** the replay. We can even hear the sounds we have just stored. By the *play* button, there is also a **Sound** button ③, but with the *red* cross. Clicking on it displays the *channel list*. By selecting the only stored channel, the loudspeaker will not have a red cross anymore. Now click the **Play** button again and whatever was have recorded will be heard from the loudspeakers. Click **Stop** again to stop the replay, and select **None** on the loudspeaker icon to switch off the audio replay.

While playing the data, you have noticed that the *yellow* cursor *moves* through the data. You can also drag this cursor with the mouse to **move** through the data file quickly.



If the data file is really big, it can be very useful to **zoom** in on a specific section. Please note that, additionally, the *recorder* has additional two **silver** cursors - named **I** and **II**. The user can drag these cursors to select a certain region.

Another way of selecting a specific area is to left-click and hold the mouse button on an *empty* area of the recorder (where no **red** events or cursors are present). This will position cursor **I** in the specified area, so that the user can then drag the mouse to the second location, which will position cursor **II**.

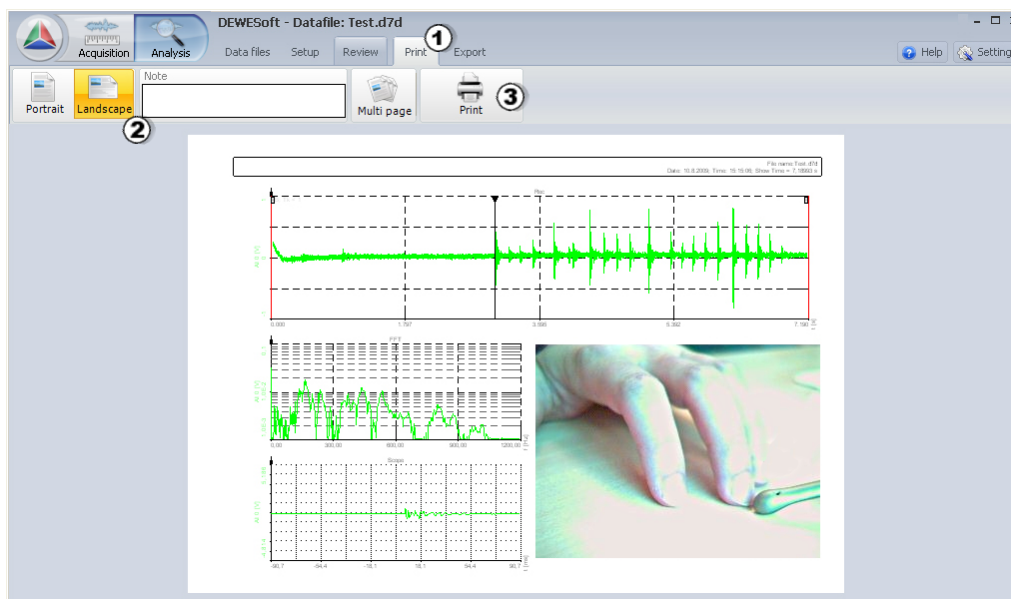
When this is done, the mouse icon will change to a **zoom** icon within the region of cursor **I** and **II**. Clicking the *left* mouse button will **zoom in** on the recorder. This can be done several times to come to the region of interest. The user can also drag the **x** scale left and right to position the data exactly. *Right* clicking on the recorder will **zoom out** to the full scale.



1.1.6 Reporting and exporting

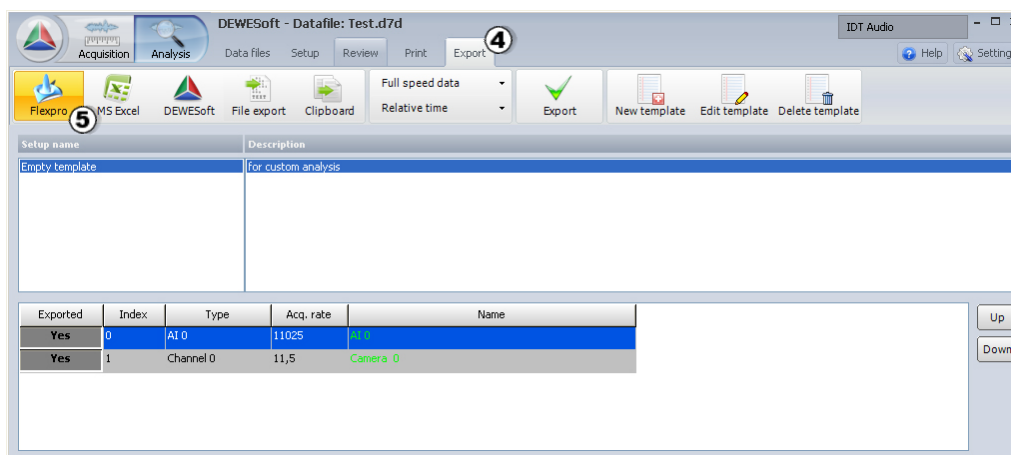
What can we do now with this data?

First of all we can **print** the current page. Just click the **Print** button ① and the same page (with white background, to be "printer friendly") will appear. You can select different *print options* ② and click the **Print** button in this menu ③ to print the data.



The next step is to **export** the data to other applications. **DEWESoft** is intended to be an **acquisition application**, thus for advanced analysis we can use other post processing applications. This can be done by selecting the **Export** button ④. We can then choose to export the data ⑤ to **MS Excel** or **Flexpro**. The user can even choose the *script* - which actually means the measurement reports. We can also choose **File export** and have a vast choice of *post processing* applications, but for this example, there is only the *data* file.

It is important to note that in this example we will export *only a zoomed region*. If we have **MS Excel** installed on the computer, it is not very wise to choose too much data for trying this feature, since Excel can't really handle much data. For *larger* data files, it is *recommended* to use **Flexpro** or **Matlab**.



Another interesting option is to make a *video* of the screen as it is replayed. We can do this by selecting the menu item **Edit** → **Export screen to AVI**. This will actually produce the video file like it would be played on the **DEWESoft** screen.

This is a short overview of what can be done with very basic hardware. There are many possibilities about what can be done with professional equipment. The measurement tutorials will demonstrate how to perform real measurements.

1.2 Display settings

This tutorial explains how to **design the screens** and work with displays in **DEWESoft**, there are four known types of *controls*:


- controls which typically show all the data (*recorder, vertical recorder, xy recorder, GPS map*)
- controls which show a part of data directly or calculated data (*scope, FFT, octave, vector scope, harmonic FFT, tabular display*)
- controls which show only one value (*digital meter, bar meter, analog meter, indicator lamp*)
- additional controls like *pictures, text* or *lines*

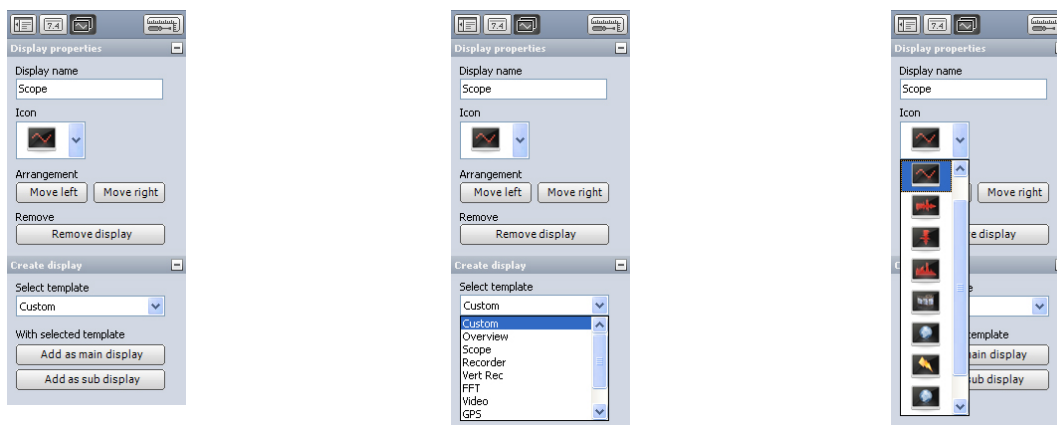
All controls can be combined into *one single screen* or we can build *several* screens for a specific part of the measurement. There are also *predefined screens* (like *scope, recorder, FFT*), but they *can be altered* to meet user's requirements.

Let's take a look how to use displays and how to set up an ideal display. An **Overview** display is intended to be defined by the user, but predefined screens can also be altered to meet specific requirements.



A conflict of interest arises in that the subsequent chapters describe in greater detail how to incorporate different measurements. This, however, is under the assumption that the user already knows how to create displays (this chapter). For the sake of simplicity in this tutorial, one measurement's *channels* will be used to make a display set-up. The next chapter will explain in greater detail how those channels can be acquired.

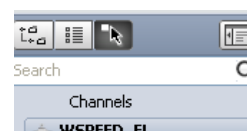
With **DEWESoft** software, the user can also find the **Example_Drive01** data file. Let's open this file because it has lots of interesting channels for creating the display setup. Go to **Overview**. Since this display is pretty crowded, let's make **another display**. This can be achieved by clicking **Display properties** button , selecting display template and clicking on **Add as main display**.



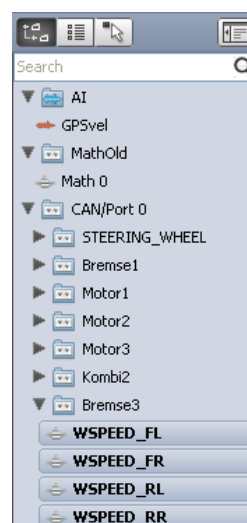
There should now be a blank display in the **Design** mode. The visual controls can *only* be **added** in **Design** mode, which is selected by clicking the **Design** button ① from main menu or on upper left side of display. Now let's see what we got from the channels. We can **add** one *recorder* for velocities. Note that there are 8 handles around the recorder. We can *adjust the size* of the control with dragging those handles. The control can also be *moved* by dragging the control to the appropriate position.



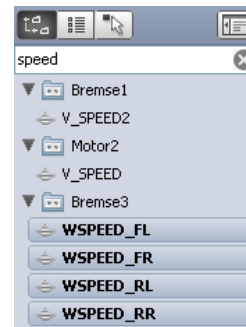
There are all four *wheel speed* measurements. When the recorder is added, the first channel is *automatically* assigned to it. To *deselect* this channel, go to the last icon in channel selector to show channels on current visual control and click on all channels to deselect them.



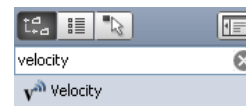
Now we can **select** the **channels** to display. Go back to the group view and note that the channels are organized into groups. In this case, there are *analog channels, counters, CAN, Math and GPS channels*. Let's click on the CAN group and select the **Bremse3** message. All four wheel speeds become visible. Clicking on them will *add the channel* to the selected visual control. Clicking on the *selected* channel will *remove* the channel from the visual control. Each visual control has a set maximum amount of channels that can be displayed on them.



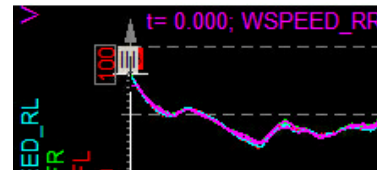
With many channels there is much easier way to find the channels. If we know that we want to have "speed" channel, just enter the keyword speed in the search caption. The channel selector will only show channels with the keyword at any point in the string, so it is much easier to find them.



The "**Recorder**" can hold up to **four** channels at maximum. This is basically *limited by* the number of **y** axis. Since there is only one measured value (speed), we can select to have all the channels related to one single axis by selecting the **Single value axis** option in the recorder. In this case the recorder can hold up to **16** channels. Now we can add the Velocity channel from GPS. We can browse through the folder structure, but it is much easier to again type "**Velocity**" keyword.

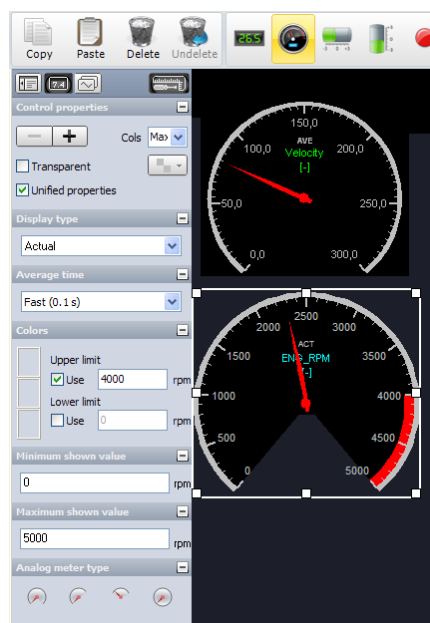


Visual control actions like zooming or scaling *cannot be done while in design mode*, so we need to click **Design** button again to be able to type in the maximum scale or to autoscale the y axis. Since the speed at this test didn't exceed 100 km/h, we can move the mouse over the maximum value of the y axis until it shows a **gray** rectangle and click on it to *enter the value*.

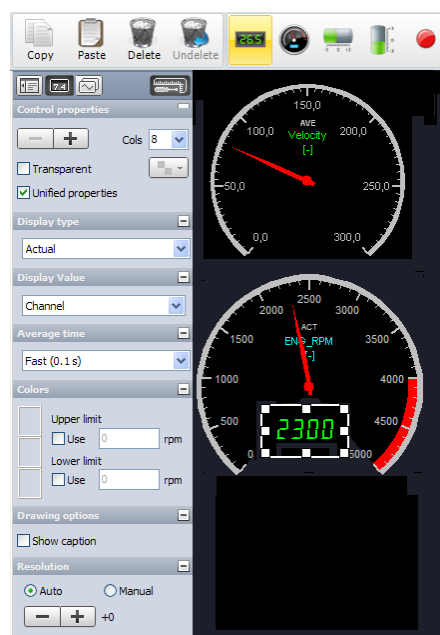


Now that this is set, we can click the **Design** button again to *continue building up* the display.

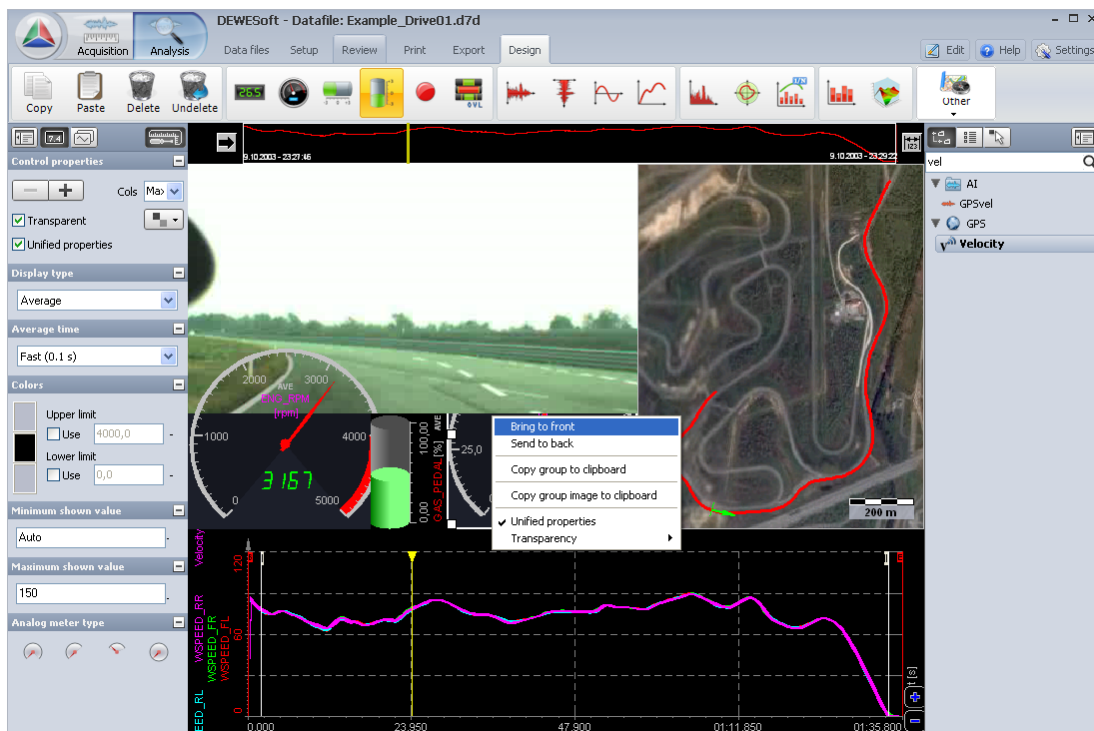
It would be nice to have some **analog meters** to display the vehicle RPM and the speed of the vehicle at the current moment. The *recorder* shows the *entire* or the *selected time period* while the *meters* show *only the current position*. In Acquisition mode this is always the *current* value, while in analyse mode this is *related to the position* of the **yellow cursor**. Let's add two analog meters. Select first meter and choose the **Eng_RPM** from **Motor1** message of the CAN bus. Select the other one and choose the **GPS velocity**. The scaling can be changed to more appropriate values. For the **Engine RPM** (since it was a diesel engine), the user could enter **5000 RPM** for the maximum value. We can also choose to use the *upper limit* (of the alarm) and enter **4000 RPM** as the tolerance limit.



We can also add two **digital meters** inside the *analog meters* to show the exact RPM and velocity as a number. The user can *add* and *move* the meters and *select* the *channels* from the channel selector as was done for the analog meters. Since it is already known that the shown value is velocity, the caption and frame can be switched off by *deselecting* the **Show caption** option. To be able to see the analog meter in behind, we can make that the digital meters are fully transparent. Since the RPM shows *one decimal*, which the user is not interested in, *decrease* the resolution by one can be selected by the **Dec** button.



Now comes to the fun part. Since video camera and GPS are already used, we have two additional visual controls available, which are **video camera** and **GPS map**. Let's add both. When trying this independently, the user will not have the GPS sky map behind the traveled path, but how to add this will be covered in the **GPS tutorial** how to add this. Let's place the controls to make good display screen. Please note that the user can *move the controls* to the foreground or the background. Since we added the video control later, it will be placed *in front of* the analog meter. With a right click on the analog meter we can choose **Bring to front** to bring the control to the foreground. The user can, of course *add a visual effect of transparency* to the analog meter, so as not to hide the video picture.

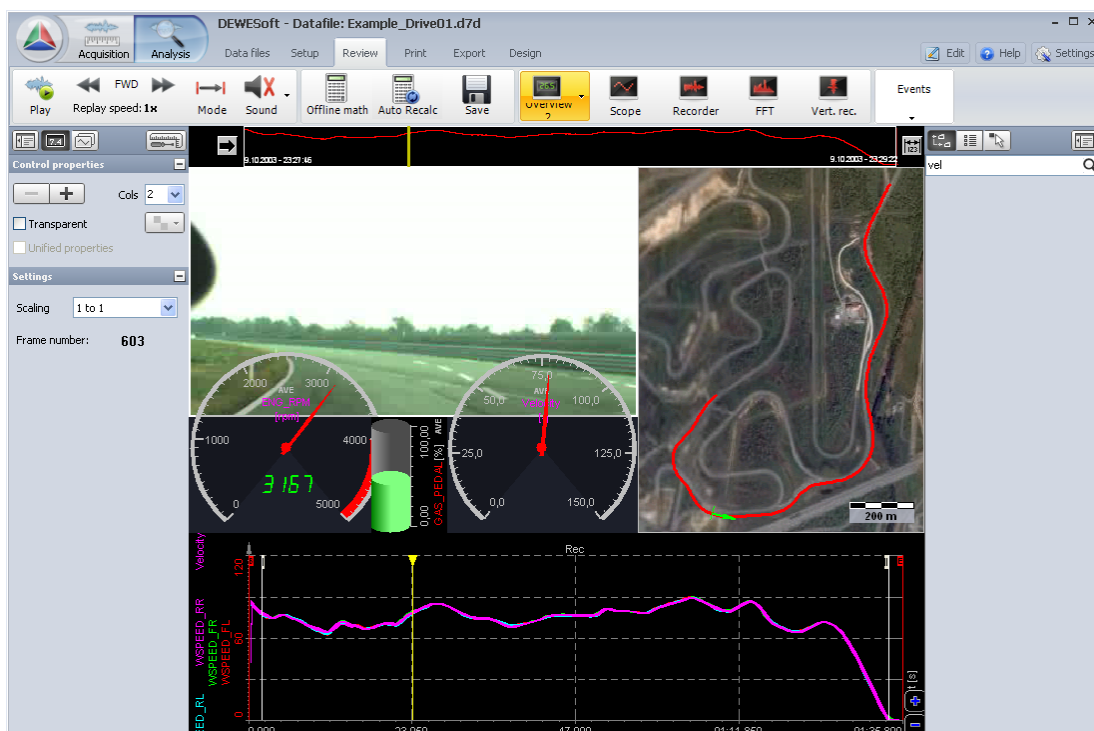


Let's *add* some more controls for the outside temperature, satellites used and gas pedal positions to finish it. The display settings can be done in **Acquisition** and **Analysis** mode. In *Acquisition* mode, these settings will be stored to the **setup** file. If we alter the display settings in the *Analysis* mode, those settings need to *manually* stored by those settings with choosing **Save** icon. We can also use *these settings for the next* measurement with DEWESoft round button



→ Use setup for measure.

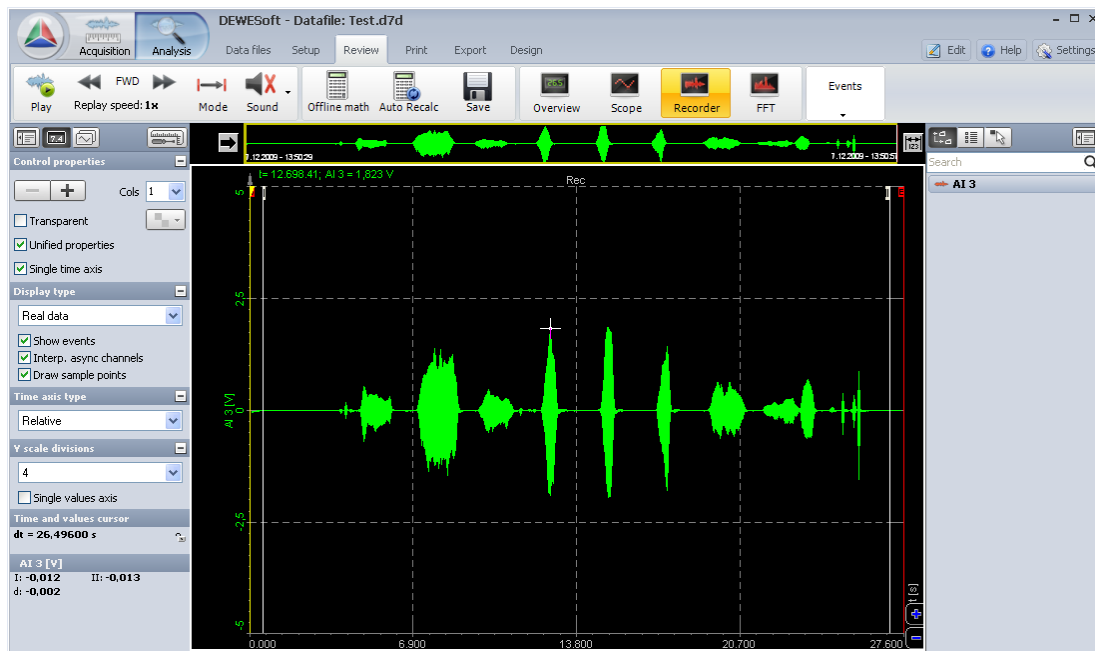
To be able to really use it, deactivate the **Design** mode. Now we can *zoom in* on the recorder and *move the yellow cursor* to update the digital meter or we can *replay the file*.



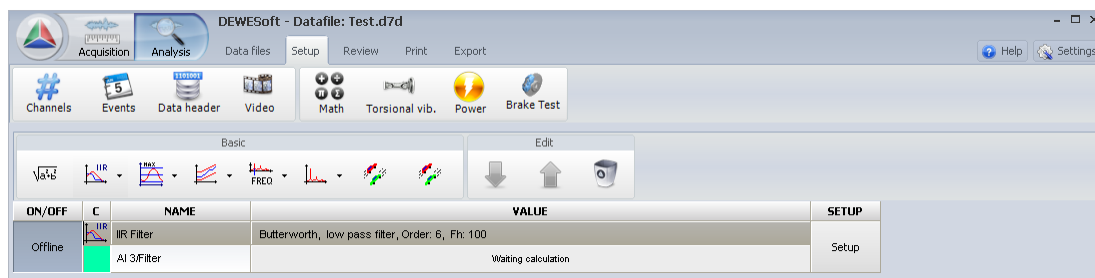
1.3 Post processing

A great new feature of DEWESoft version X is **post processing**. All the mathematic which are available *during* Acquisition (measurements) can now be added on also *to already stored* data files.

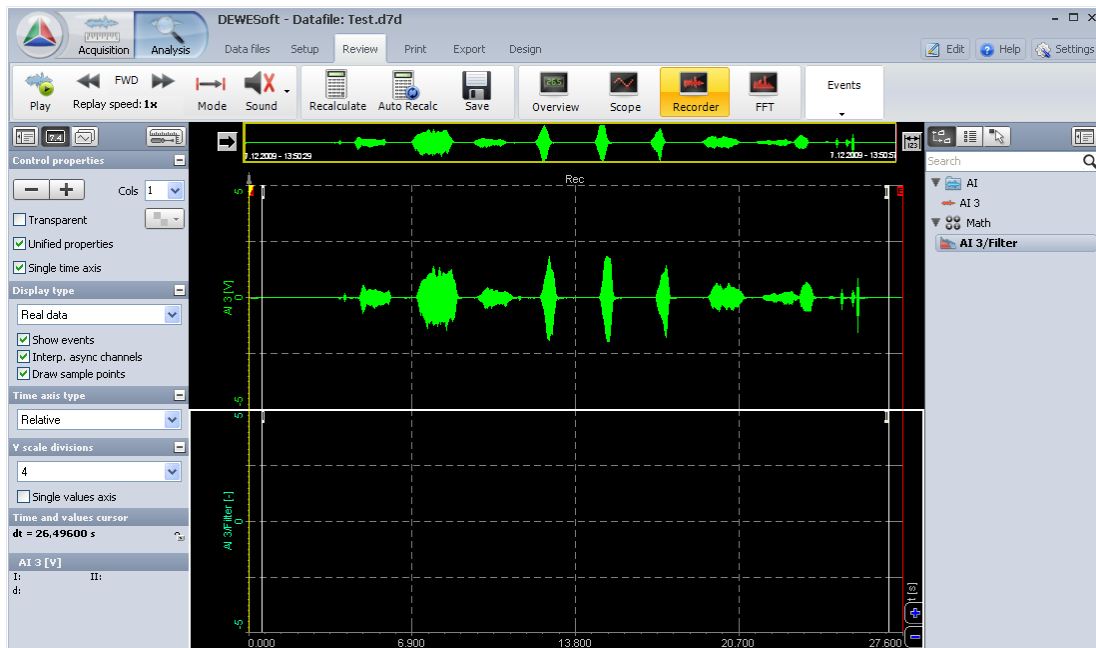
Let's acquire some analog data with the microphone and *open* the file.



Now press the **"Offline math"** button ①. This will open the **"Setup"** tab sheet in **Math** section. Here you are able to add any math just like being in **Acquisition** (measure) mode.

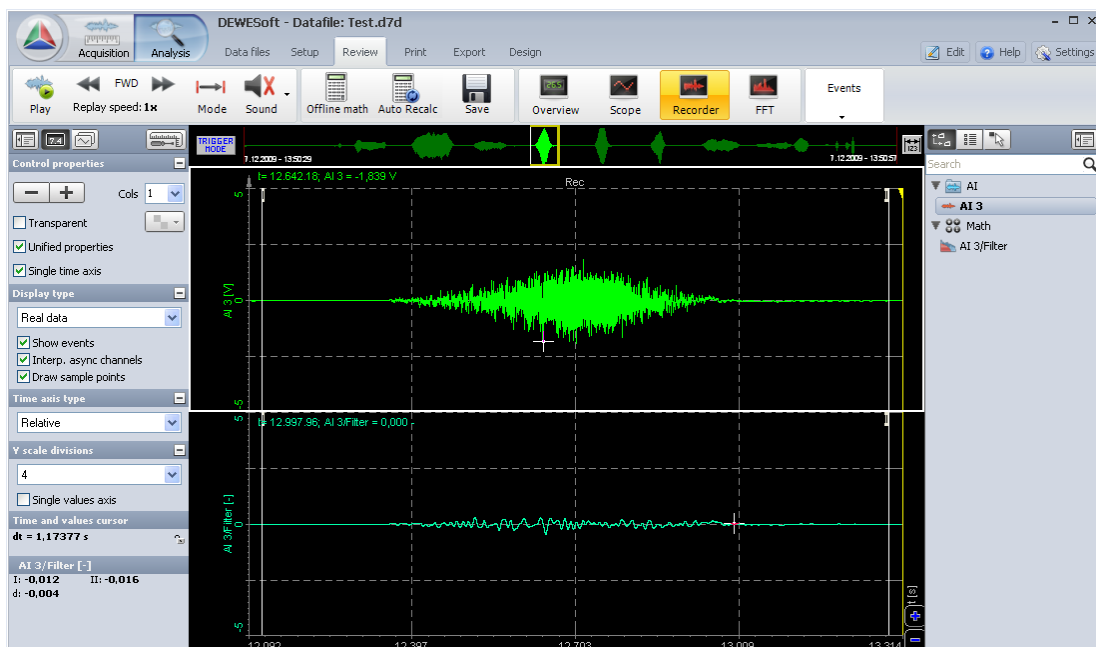


After going back to **Review**, the **Offline math** button changes to **Recalculate**. Let's add new recorder by pressing a **plus** button. When adding **AI3/Filter** channel which we have just created, we can see ... nothing. Recorder is empty. The reason for that is because we need to issue the **Recalculation manually**. The big benefit of that is that we can define a region where the data is recalculated.

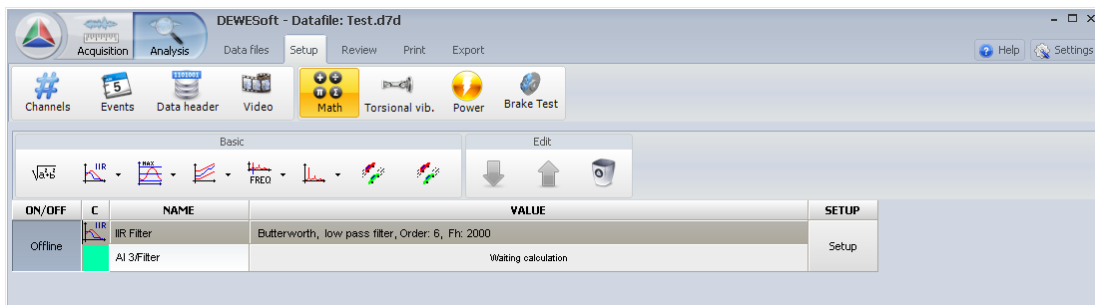


Imagine you have a huge file. Add a filter to many channels, issue the recalculation, go for a coffee, come back and see that it would be much better to choose different filter type.

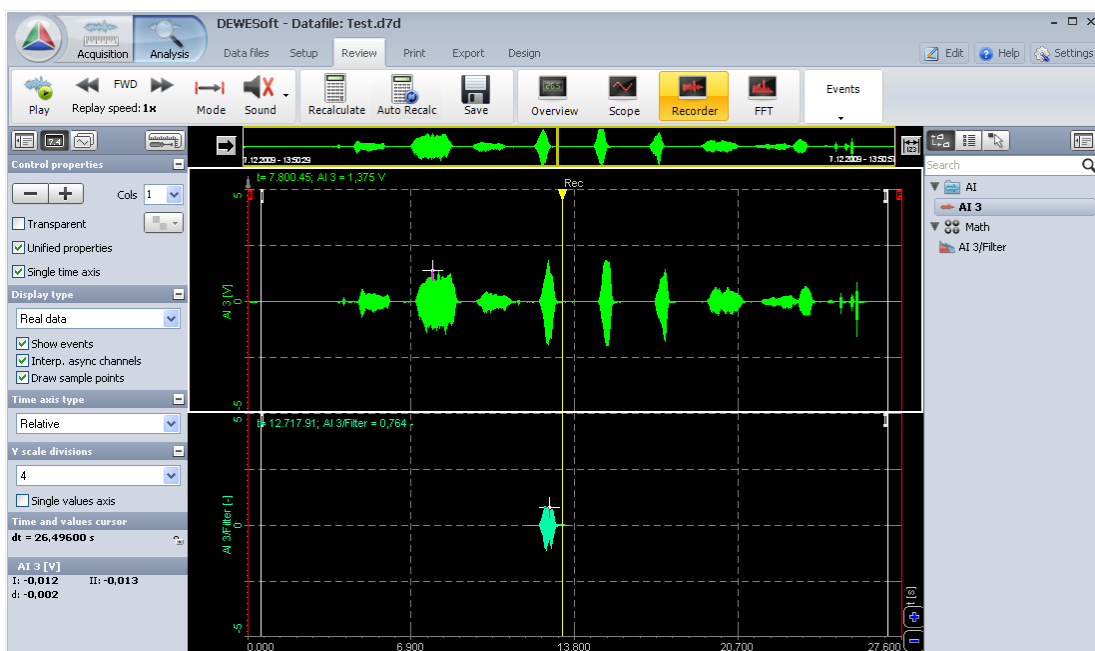
DEWESoft offers you to zoom in the region and calculate only for that region. After pressing Recalculate button, filter is calculated and the button turns back to **Offline math**. If the result is not perfect, we can go back to Setup (either by clicking **Setup** tab or clicking on **Offline math**) and *change* the math.



Let's change the cutoff frequency from 100 Hz to 2 kHz (in the **Setup** screen of IIR filter). After doing this, we can see that the status of the math channel changed from **Calculated** to **Offline** and the online values says "Waiting calculation".



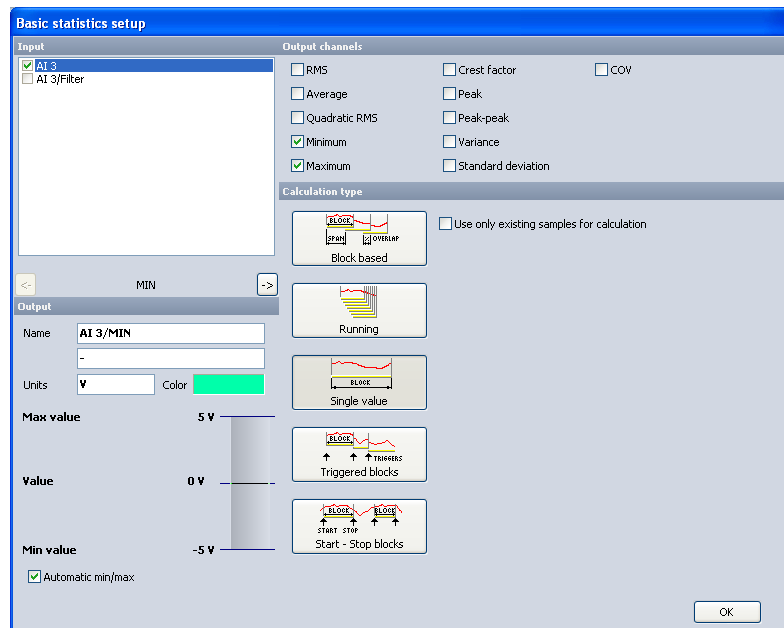
Go back to review and press **Recalculate**. New calculation will be performed and will replace previous one. When we are satisfied with the result on one section of the file, we can *zoom out* to the *whole* recording. We will see only a part of the calculated channel and the rest will be empty. The Offline math changes again to **Recalculate**. After pressing **Recalculate** we can go for a coffee with a good feeling that everything will be correct when we it will be recalculated.



We can add all kind of math channels, not only *formulas*, *filters* and *statistics*, but also *power*, *order tracking*, *torsional vibrations*, *combustion*, basically everything what is possible online.

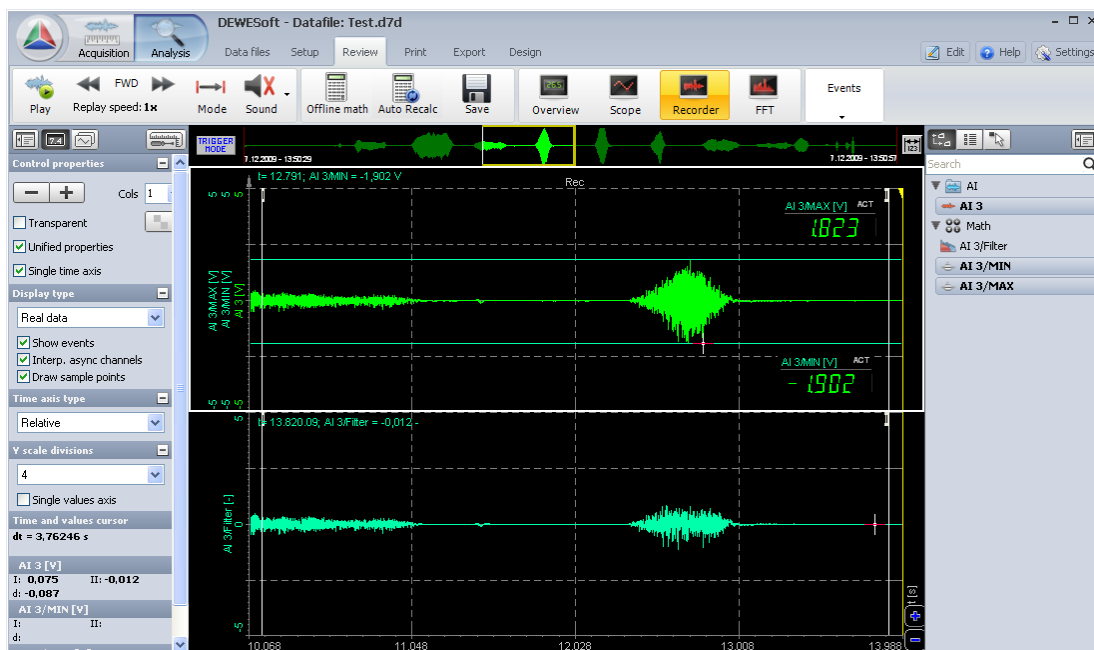
Let's look at another nice usage of offline math.

Go back to math setup and add another math block - this time **Basic statistic**. Go to the setup and select **Single value minimum** and **maximum**.



We have now two more channels: **AI3/MIN** and **AI3/MAX**.

Go back to **Review** and **add** those channels on the recorder. For practicing display settings we can also add some meters. When zooming in and pressing **Recalculate** we get the min and max values for that zoomed region.



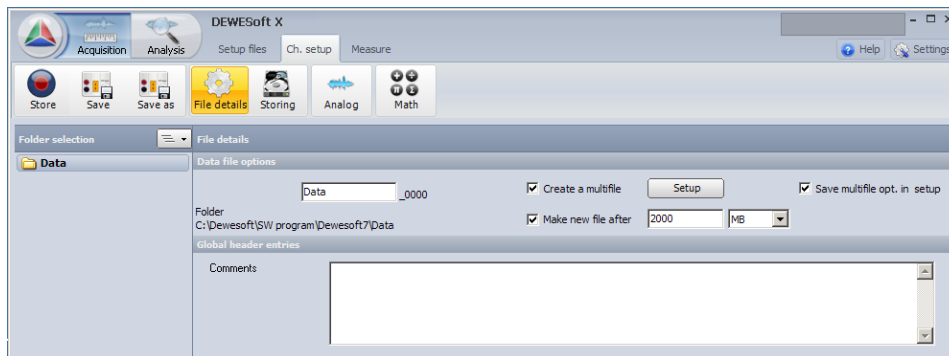
Another very nice option is the **Auto recalc** ②. Instead of pressing **Recalculate** every time a new section is chosen, **Auto Recalc** will do that automatically.

The last step is then saving these channels to the data file. When we have zoomed out in full and **Recalculated** the channels, we can press **Save** button to store *math settings*, *visual controls* as well as *calculated channels*.

1.4 Storage options

DEWESoft offers many ways of how to **store data**. All settings for this are done in the **Acquisition** setup screen.

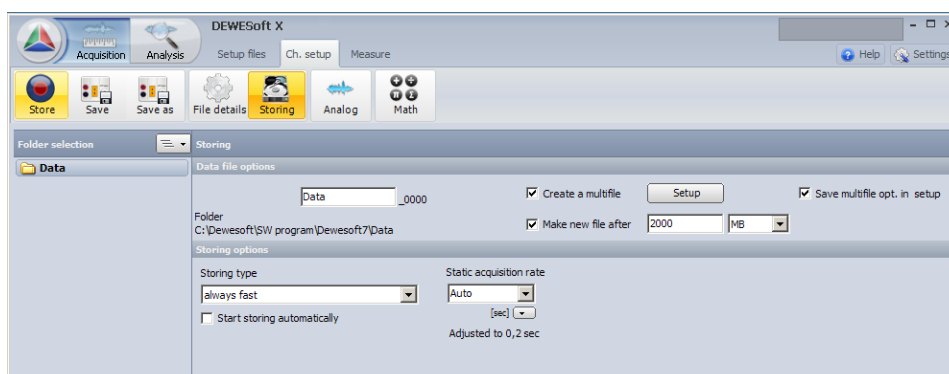
First, let's take a look at how we can *name the files* to be stored. We can define the file name for each measurement separately by entering it into the file name edit-field. The default folder where data is stored can be changed by clicking on the **File details** button.



For repetitive measurements, it can be helpful to use the multifile function. Multifile *automatically assigns* a new file name for each cycle (start) of storage. File names can be either consecutive (such as 0001, 0002, 0003) or by the date and time.

Additionally, we can create a new file *after a certain file size* is reached or *after a predefined time*. This is done by selecting **Make new file after** check box. The criteria for switching to a new files are either the *file size* or *time interval*, which can be defined in seconds, minutes or hours.

In this case we may wish to switch the file after *reaching absolute time*. This can be very useful when acquiring data for longer time periods. If we choose to switch the file each hour with absolute time, then the switching will be done *exactly on the hour* (01:00, 02:00, 03:00...). The time will be taken from absolute PC time (or other more exact timing source, if available, as defined in the hardware setup). The file switching is done in such a way that *no data point is lost* in the process.



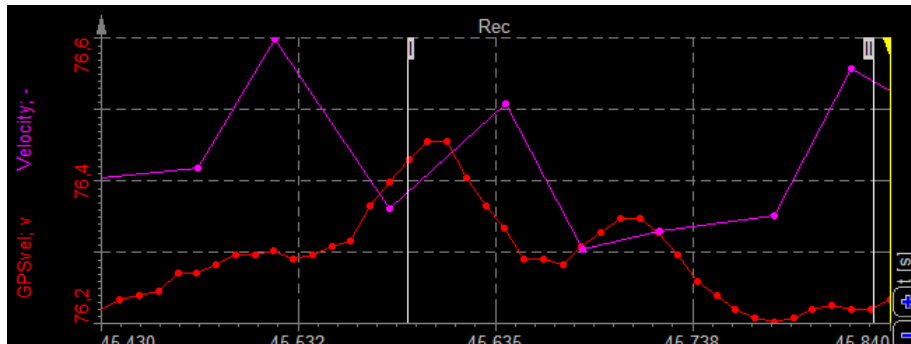
We have four different storage options which *relate* to the basic **sample rates**:

- **always fast**: data will be always stored at *high speed*, as defined by the dynamic acquisition rate
- **always slow**: data will be always stored at *reduced speed*, as defined by the static/reduced rate
- **fast on trigger**: data will be stored with the dynamic rate, once the *trigger point occurs*

- **fast on trigger, slow otherwise:** data will be stored with the dynamic rate at *trigger points*, and with the reduced rate when there is *no trigger*.

Now let's look at each individual storage options. Before looking into that, it is worth explaining there are two types of channels in DEWESoft: *synchronous* and *asynchronous channels*.

Synchronous channels (like *analog*, *counter* or *digital*) are channels which are acquired at an *exact predefined* speed, defined by the dynamic acquisition rate. *Asynchronous* channels are channels where the *rate is not known* beforehand (PAD, CAN, GPS...). These channels are always acquired at the speed with which *they come* from the device. The picture below is a typical example of a *red* analog channel with a *fixed* rate and, a *purple* GPS channel. The user can see that the rate of the channel is *not fixed*.







Only the synchronous channels are influenced by the dynamic acquisition rate, so raising the sample rate will increase the amount of data stored. The asynchronous channels are not influenced by the sample rate, we need only ensure that the dynamic rate *is faster* than the rate of data *coming from the asynchronous device*.

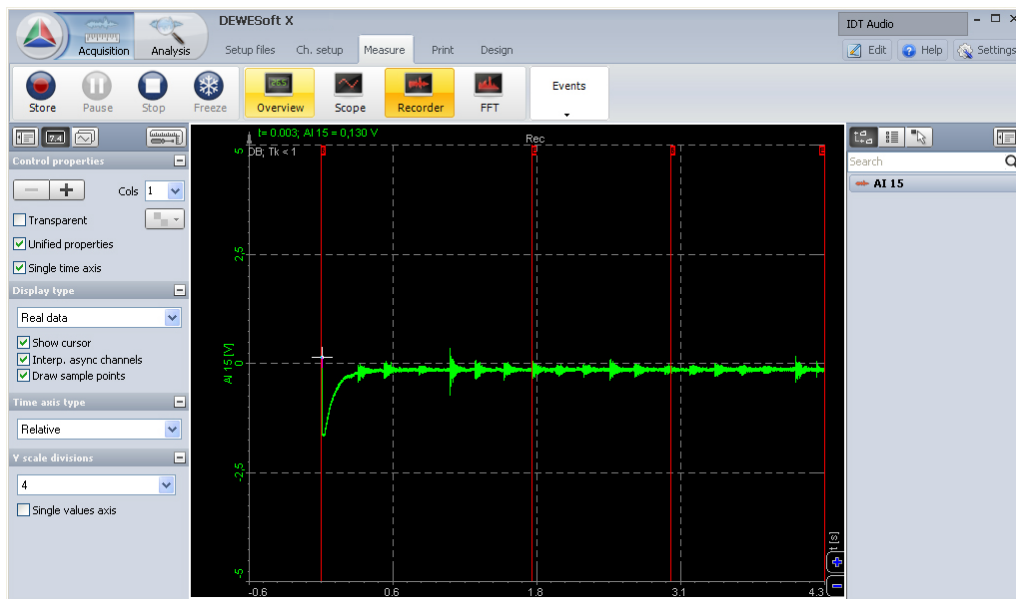
For example, for CAN bus it is usually enough to acquire the data with 100 Hz; other sources like PAD or GPS are even slower. If there are only asynchronous channels, the sample rate only defines the time stamp precision (resolution).

1.4.1 Always fast

If we choose to store **always fast**, the data will be stored *all the time*. Let's do this. The hardware used is the same as for the basic tutorial - only a sound card and a microphone. To start storing, click **Store** on the main menu. Now the data will be stored to the file with full speed.



The **Store** button changes to  and if we click the **Pause** button , data will still be acquired, but storing will be *paused*. At that point, the **Pause** button caption changes to **Resume** button  (**Store** button also change his appearance to ) , and if we click it again, storing will be resumed. Due to this, there will be two sections of data.



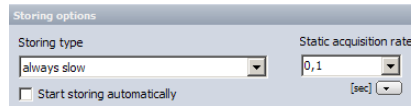
If the user clicks **Analysis**, they will see two sections with data and the *space in between* will be *blank* - no data is stored there.



1.4.2 Always slow

If we want to acquire the data with *slow speed*, we can choose the **always slow** storage option. The data will be stored at *intervals*, which are set with static/reduced rate. In our case, this is set to 0,1 seconds. Thus much *less disk space* will be used for storage.

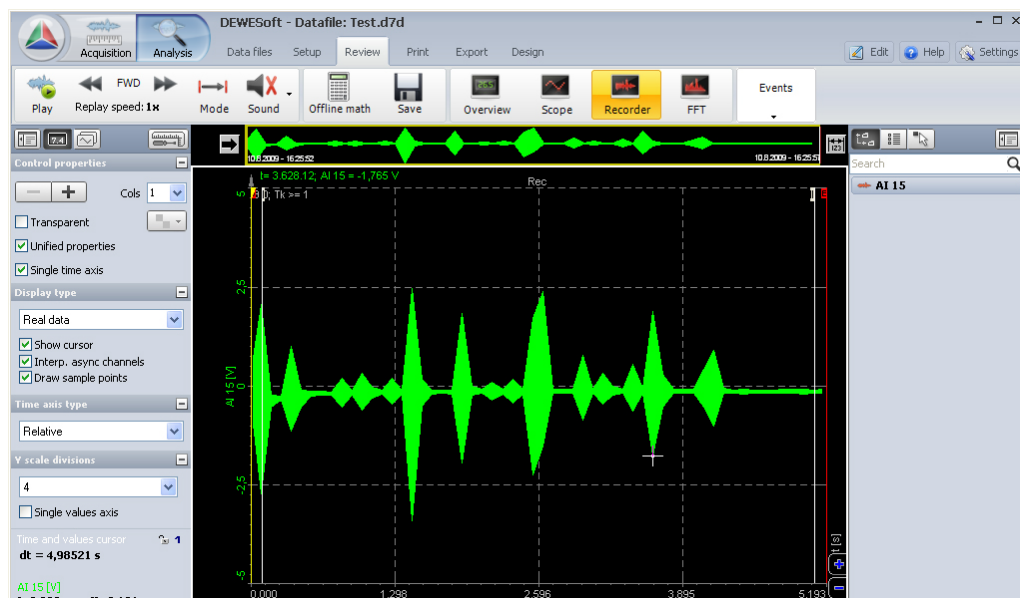
Even though storage is set to slow, **DEWESoft** will still **acquire** the data at full speed, **calculate** the minimum, maximum, average and RMS for this *time interval* and store *only* those values.



Let's store some data and take a look how the data appears in the *recorder*. This is what the data looks like at *full rate* in Acquisition mode.



And now let's switch to **analyse** mode. The data will be shown as the *envelope of the original signal* since the full rate rate is no longer available, merely the 0,1 second values. We can also switch to average or RMS modes in the recorder setup to see those two parameters.



The recorder below shows the RMS of the signal. We can judge from the *average*, *RMS* and *min/max* values what the original signal could be. For example if there is a big maximum and the average value did not grow, we can deduce that there was a short spike in that channel.



1.4.3 Fast on trigger

If our data consists of events which can be captured, we can choose to store it as **fast on trigger**. The trigger event can be defined in the software and then DEWESoft will wait for this event and *store only the portion* of interest.

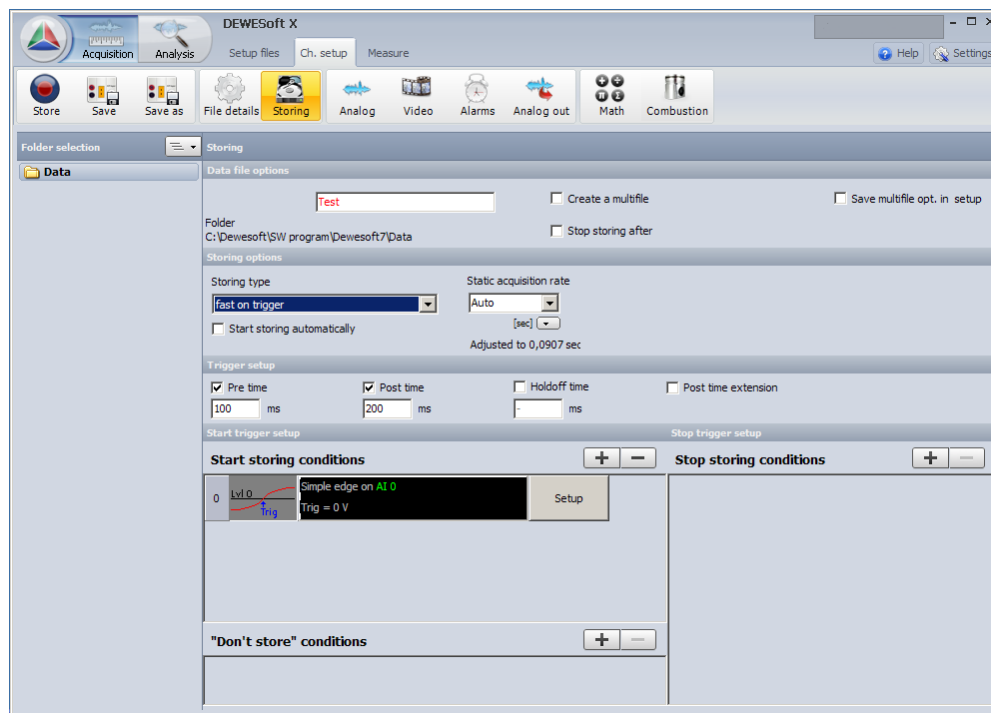
This can be set by choosing the **fast on trigger** storing option. After doing this, the *new tab* **Trigger** will appear where we are able to **set up** the trigger condition and strategy.

First we can choose to define the pre-time, post-time and holdoff-time.

Pre time is the time which will be stored *before* the trigger event occurs. We define the 100 ms before an event as the pre-trigger. This means that DEWESoft will keep the data in the buffer until the trigger event occurs and then store this data to the file in addition to the regular data.

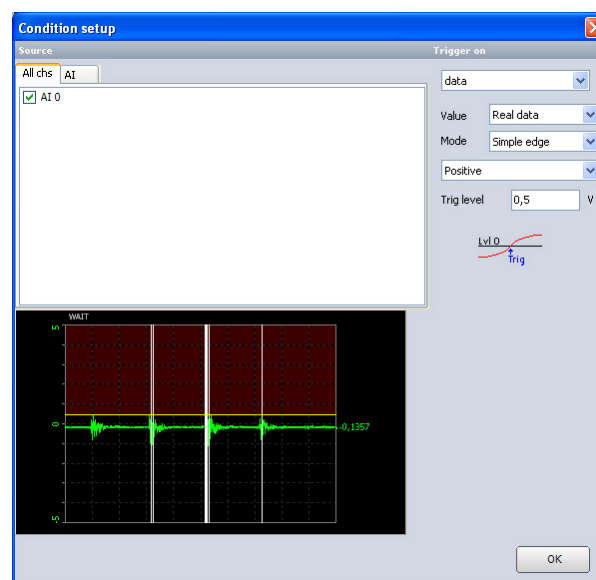
Post time is the time *after* trigger event which will be stored. If this is not defined, DEWESoft will *continue to store until* we stop it manually or a stop condition occurs. For this example, we want to capture trigger shots, so the post trigger should be set to 200 ms, so a total of 300 ms of data will be captured per trigger event.

Now we need to *define the trigger conditions for the beginning* of storage. Let's **add** one trigger condition by clicking setup to define it.





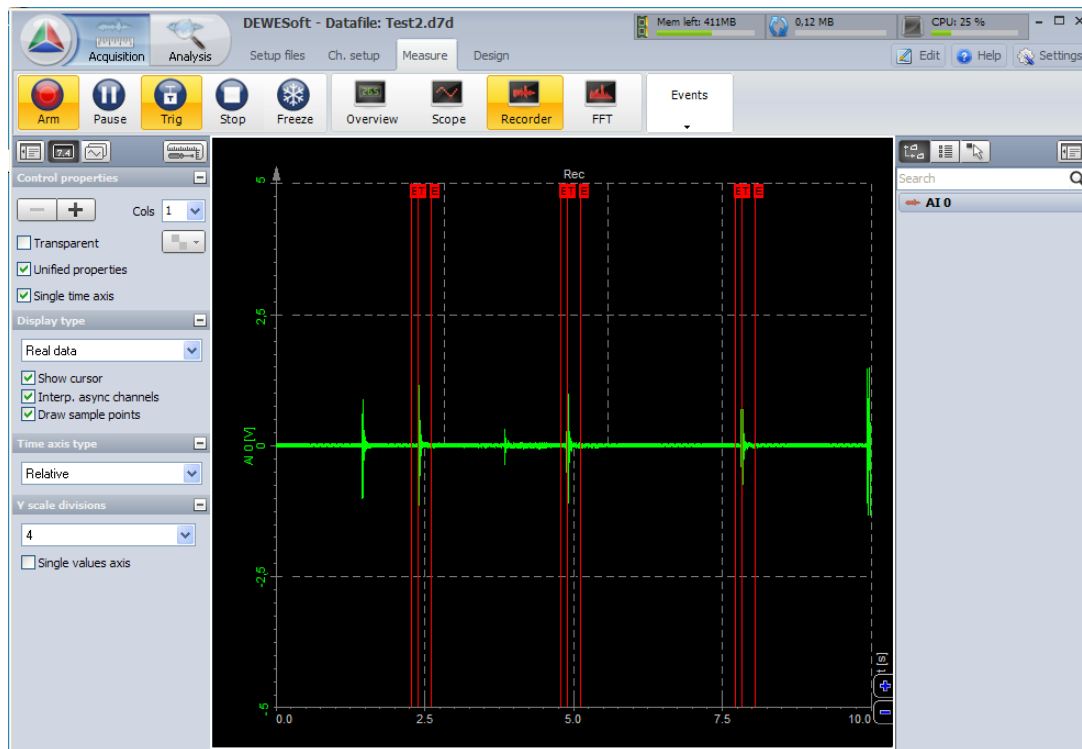
In the trigger setup window, we can choose *channels* for triggering. We can select *several* channels for triggering, but since this example deals only with one, there is only one choice. Then, define the trigger criteria needs to be defined. The user can trigger on time data, time or FFT.

Time triggering includes *edge, filtered edge, window, pulse-width...* on *real data, average* or *RMS values*. For this simple application, only the simple edge with a trigger level of 0.5 will be selected. This means that when the value crosses the 0,5 V limit, it will produce a trigger. One can already test the behavior of the trigger from the scope in the lower left side of the setup window.

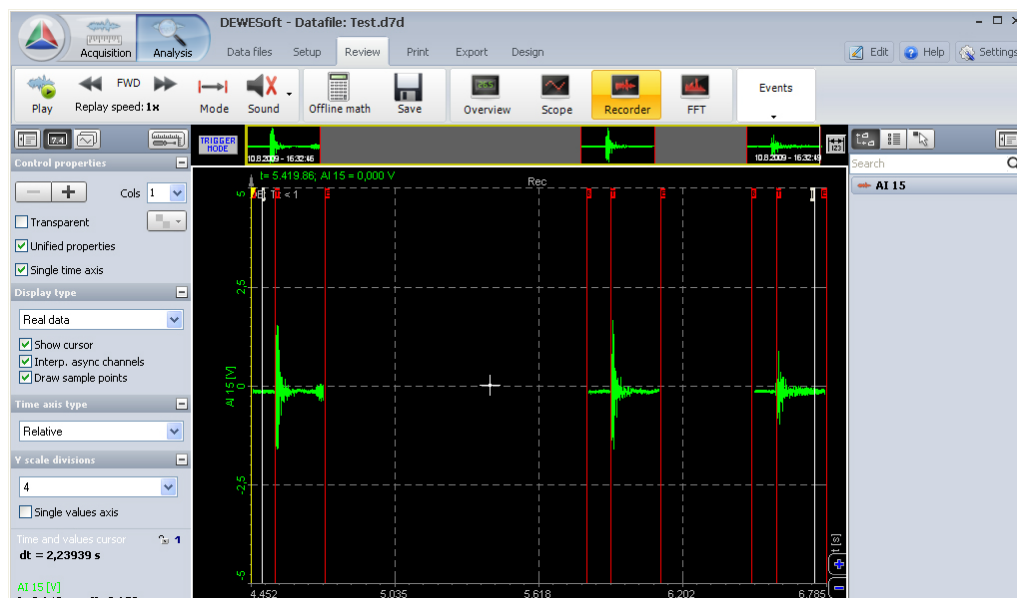


Now let's take some **measurements**. The user just needs to hit the microphone to produce the necessary trigger. We can see from the *recorder* that the first shot was not high enough, therefore we hit it harder. That did it, and one we can see the *beginning of the storage event, the trigger event* and the *end the of storage event*.

Note that the **Store** button *changed* the name to **Arm**  and there is an *additional* **Trig** button . This is manual trigger to issue a to trigger even without an event.



Let's **review** the data being stored. We can see that *only the trigger events* are stored, yet for the rest of time the data is blank. Note that there is a new button, called **TRIGGER MODE**, in the data preview. This gives us a chance to review the trigger events *without zooming* in on the data. If it is clicked, the first trigger event is *automatically zoomed in*.



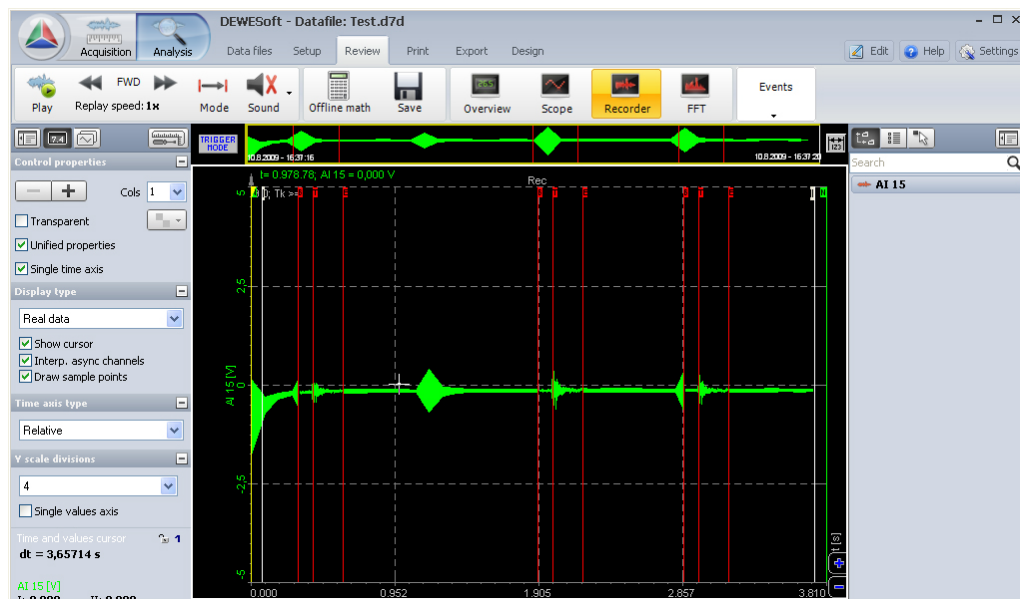
The **trigger mode** button changes to an "arrows" button, where we can browse between the events. If those two buttons are clicked, the recorder shows the trigger events *one by one*. On the data preview we can see the currently selected trigger event. Right click on the recorder to *zoom out* to the full region and to *leave* trigger mode.



Note that there was had one peak in the beginning of storage. It was not enough to be stored with the trigger, but sometimes it is still nice to see what the values in the regions without the trigger event were.

1.4.4 Fast on trigger, slow otherwise

To be able to acquire data with two speeds, we need to use a different strategy: *fast on trigger, slow otherwise*. All the settings for this mode are the same as for *fast on trigger*. It should be noted, however, that if the user acquires and reloads similar data with this strategy, the data is also *reduced* for the regions without the trigger event. This can be seen on the picture below.



By *zooming in* on the data, one can see the reduced, stored data *before* the trigger, where only the maximum and minimum of the signals is seen and then for a *region with* trigger the *full speed* data can be seen.



2 Measurement tutorials

Measurement tutorials give us an overview of some **basic technical measurements**. It will help to choose the right *sensor* and *amplifier* for a specific task.

| | | |
|---|-----------------------------|--|
|  | Voltage and current | Voltage and current measurements of grid voltage and consumed/produced currents |
|  | Sensors with voltage output | Sensors with <i>voltage/current or digital output</i> - we need to measure the output and convert it to the appropriate units |
|  | Temperature | Temperature measurements with different sensors: thermocouple, thermistor, thermal resistance... |
|  | Vibration | Vibration measurement for the measurement of surface vibration of certain elements |
|  | Strain gage | Strain gage measurement principles of measurement strain and stress of the materials |
|  | Counter | Counter channels are used to perform counting and frequency measurements |
|  | Frequency measurement | Measuring the frequency with counters, analog and PAD with differences |
|  | Video acquisition | Video acquisition provides important information in addition to <i>other acquired data</i> for data analysis |
|  | CAN bus measurement | "Listening" and acquiring data on CAN bus in cars and other vehicles |
|  | GPS acquisition | To determine the position on earth, precisely calculate the speed and synchronize remote systems |

2.1 Voltage and current

Voltage and **current** measurements are very common.

Voltage measurements can be split into several sections: high voltage grid measurements (in **kilovolts**), where we need voltage *transducers*, direct low voltage grid measurements (**120/230 V**) and low voltage measurements (up to **50 V**). The high voltage converters convert kilovolts voltage signals to a measurable range - up to one kilovolt. If you have a good amplifier (Dewsoft has SIRIUS HV), there is not much to be said about voltage measurements.

Current is similar - we can measure high currents with a Rogowsky coil or current clamps; or low current measurements which is often done with shunt resistors.

A *Rogowsky* coil can be used for **AC** current measurements. Directly it measures the derivative of current, therefore an integrator circuit or software filter module must be used. Current clamps work on the Hall effect principle, and it outputs the voltage proportional to the current. Both principles include a phase shift of the output.

The *shunt resistors* are very useful for the measurements of small currents. The theory behind their operation is simple - with known resistance and the a constant current flowing through the whole measurement chain, the voltage is directly proportional to the current. This will be explained a little bit later when calibrating the shunt resistor in the software.

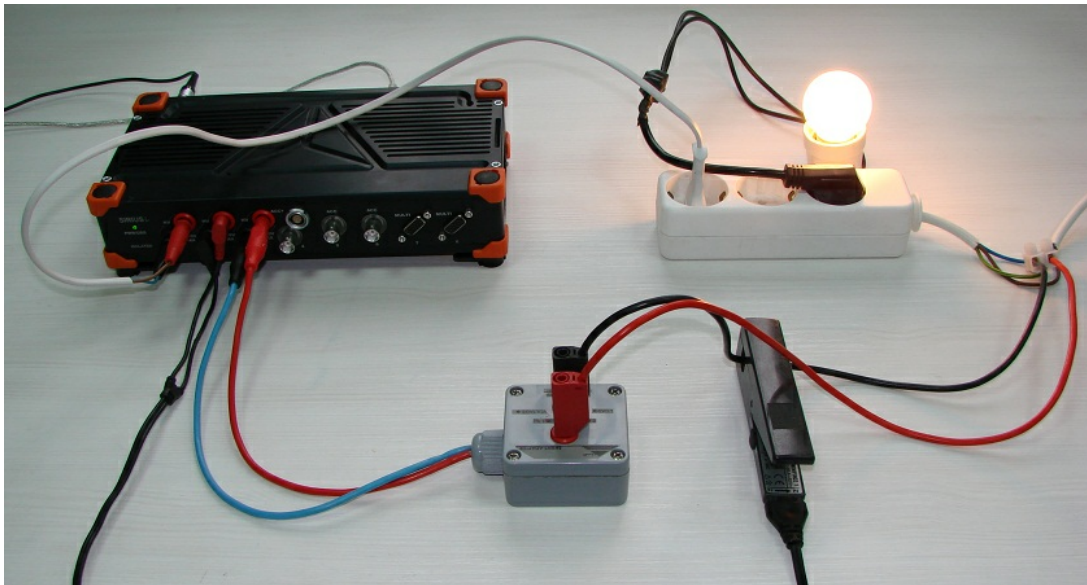
In all cases, it is highly recommended to use *isolated amplifiers*. Even if the voltage itself is not too high for the not isolated card input, this voltage might have a high potential to ground, which might kill the card. A good example is measurement with shunt resistor, which is basically just the high precision high current resistor, on which the voltage drop is measured. Even though there might be only a few volts measured on the output, the potential might be as the high as grid line voltage. This would - if connected directly to AD card - surely kills the card.

Dewesoft signal conditioning

Let's take a look at one typical example of "low" voltage and current measurement. We will measure the grid voltage and the current consumed by a *40 W light bulb*.

We will use three high voltage (HV) modules on isolated Sirius.

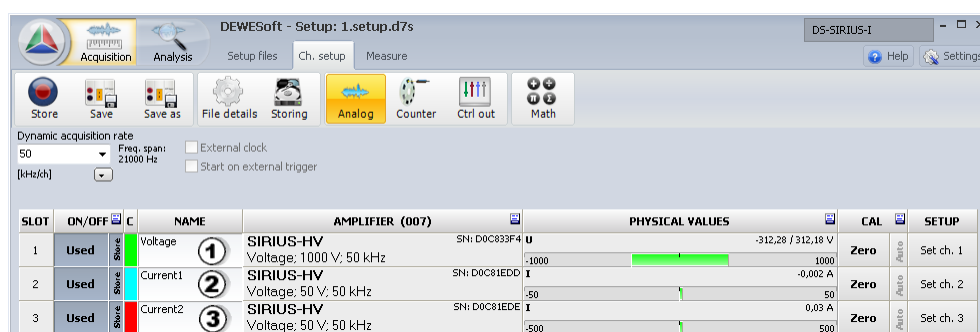
The **currents** will be measured using two principles. We will use a current clamp, which can be easily mounted since it can be opened. The second principle is the shunt measurement, where we need to cut the wire to include the shunt in the *series*. One should also be very careful not to *exceed the* maximum current of the shunt; otherwise we can measure a nice fire. For the moment, let's focus on voltage and current.



2.1.1 Channel setup

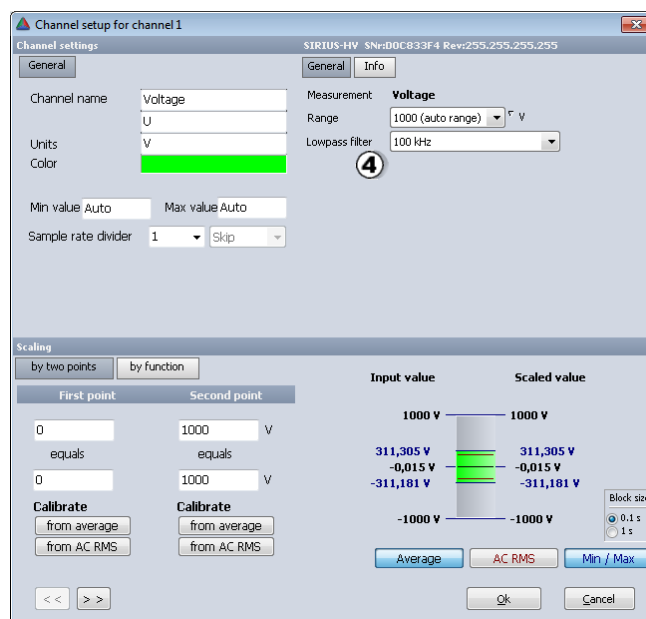
| | |
|-------------------|---------------------------------|
| Required hardware | Sirius Isolated with HV modules |
| Required software | Any license, except LT |
| Setup sample rate | At least 5 kHz |

Let's take a look at how to make the **DEWESoft setup** for the configuration shown on the previous picture. We will use first three Dewesoft programmable channels - one for voltage measurement ① and two for current measurement: ② and ③:



Since the *input voltage* can be ± 1000 V, the voltage can be measured *directly* without any of additional transducers. We expect voltage $230 \text{ V}_{\text{rms}} \cdot \sqrt{2} = 325 \text{ V}$ peak.

We need to take special care for the settings of the **Lowpass filter** ④. If this setting is *lower* than half of the sample rate, it will *cut* the signals already in the range of the measurements. Sometimes this is needed, but more often this filter is set to the low range by mistake, and then the measurement results will be invalid.



Now let's **calibrate** the *current*. Below is the picture from the current clamp sensor. Since we have only the light bulb, **10 A Range** on the current clamp will be more than enough. With Sirius, since we have dual core ADC, measurement range is not so important. We can choose highest range and have no worries we will get an overload.



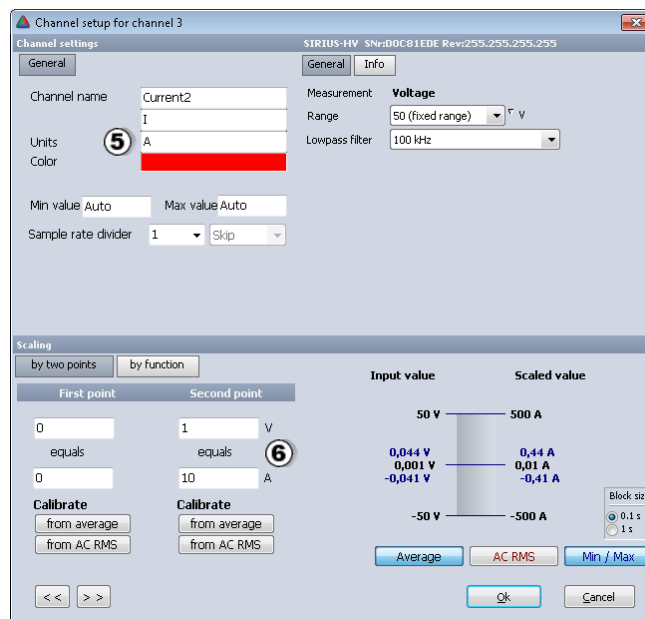
At this range, the sensitivity of the sensor will be **1 mV/mA**, which means that 1 mA at *input* will *output* 1 mV at the output. In the **channel setup** we enter that the measurement of current **I** is in unit **A**, and leaving the *scaling factor* as **1**. That's all there is to this one. Let's take a look how to scale the shunt resistor.

There is an easier way to **scale** the shunt resistor - which will be covered later, but let's look at the theory of how to scale the shunt resistor. This is a 0.1 Ohm shunt. Since it is connected in sequence, the current will be the *same on all* elements. The voltage drop will be measured on the shunt resistor. From the Ohmic law, the current for the voltage drop is calculated as follows:

$$U = R \cdot I = 0.1 I$$

So, there will be **0.1 V** at the *output* for **1 A current** or **1 V** for **10 A**.

It is time to enter the *scaling factor*. We enter the unit of measurement as **A** ⑤. Then we go to the **second point** scaling and enters that **1 V** measured equals **10 A** ⑥. That's all.



We also have an *additional option* to define the shunt type, for example on Sirius - multi module. For sensors with current output, it might happen that the current is only the transfer mechanism and the real measurement value is different (like pressure). There, we would need to **combine scaling factors** from *real measurement (pressure)* to *current*, and then from *current* to *voltage*.

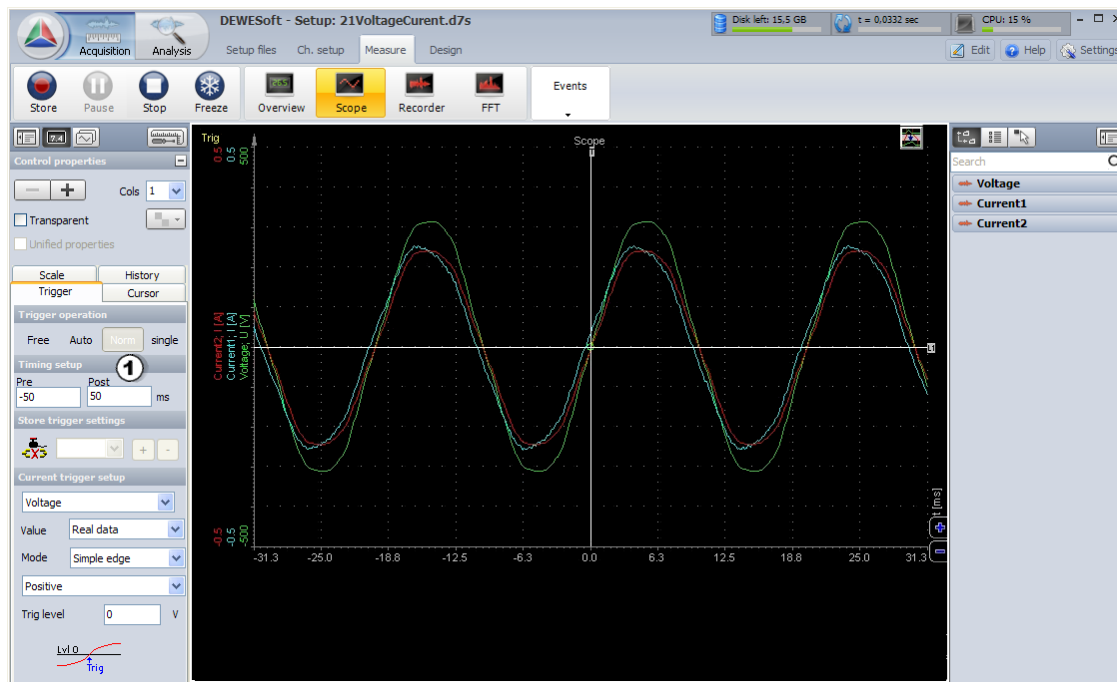
Therefore, it is convenient to *define* the shunt resistor as the *part* of the amplifier. In the section of the **amplifier**, we can define the **shunt type**. Standard Shunt1 is a precise 50 Ohm resistor, used to measure 4÷20 mA or 0÷20 mA current loops. The shunt 2 is 0.1 Ohm resistor, used to measure currents up to 5 Amps.

We will learn how to use the shunt resistors in the next tutorial → [Sensors with voltage/current/digital outputs](#)

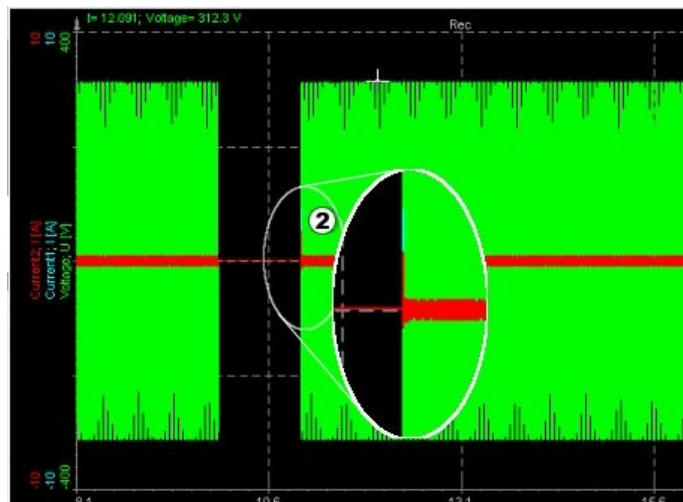
2.1.2 Measurement

Now, let's do some measurements. We have three *channels* - one is high voltage and the other two are currents. The best way to **observe** the waveform is in the *scope*. When we first start the scope, we can see almost nothing since the scope is running in the **Free run** mode. We need to "*hold*" the measurements by using **Trigger** → **Norm trigger** ① and defining the trigger *source* and trigger *level*. For now it is ok to leave the trigger as it is - trigger source is the **Voltage** channel and the trigger level is 0. We can see the difference between the measurements of current from the shunt resistor and the current clamps. The ohmic load from the light bulb should have the current which *follows* the voltage curve *exactly*. This can be seen nicely as the **red** current measured from the shunt, while the **blue** current is distorted by the phase angle errors from the current clamps.

This will result in *wrong* measurements of power. The following sections where the power module is described, the user can see how to compensate for this error, by defining a sensor transfer curve.

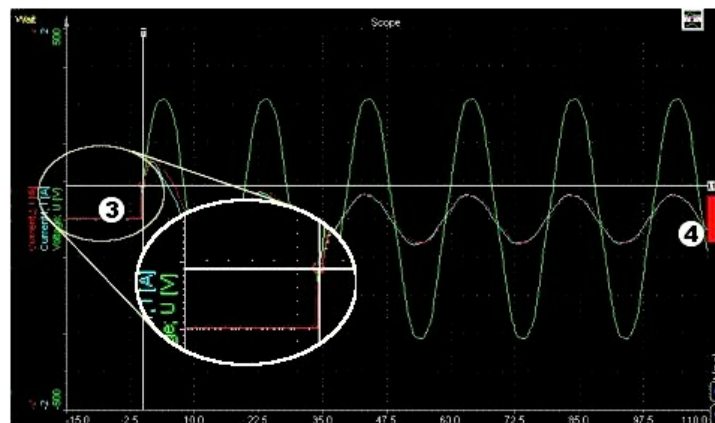


Let's look at the *recorder*. If the power supply is unplugged and plugged back in, we can see in the recorder (red curve) that the current raises *above* the normal consumption ②. This happens because at a lower temperature, the light bulb has a lower resistance than at operation temperature.



Let's catch this event in a better way by going back to *scope* and going back to the **trigger setup**. We can't really trigger on voltage, because the voltage levels *doesn't change* from startup to normal operation. Instead, let's **trigger on current**. The user should define the trigger *source* as Current and define a *level* which is normally not exceeded (for example **0.36 A** ③ in our case). When the *scope* is not triggering, the bar ④ on the right side shows the *current levels* of the signal so we can *optimize* the trigger level according the normal values. We can also use the **Auto** trigger mode. When the trigger is lost for several *seconds*, the data will be shown as *non - triggered*.

Returning to **Norm** mode - when we issue another trigger, the *scope* will show the *current event*, which will remain, until the *new event* occurs.



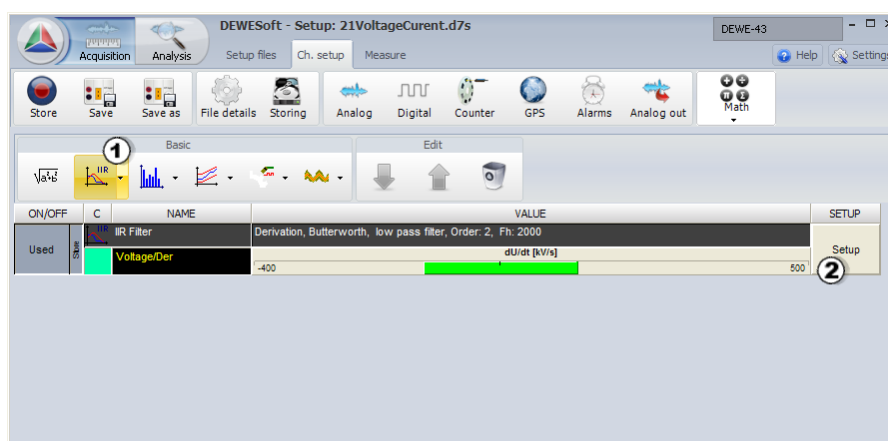
But what if we want to trigger on a *glitch*? This might not be easily noticeable since the signal could drop by only few percent and it is not possible to define a level where we would catch all the glitches.

2.1.3 Glitch triggering

For this purpose, it is very nice to use a **derivative** as the *source* of the **trigger**. The glitches will be seen much better on a derived signal (dU/dt), since the derivative is very *sensitive* to differences in the sine wave.

With the pure 50 Hz sine wave and 220 V, the basic derivative is $315 \cdot 2 \cdot \pi \cdot 50 = 100 \text{ kV/s}$ maximum. This way, the glitches are producing higher levels - from 300 kV/s on up. Thus, it becomes easier to set up the appropriate triggers.

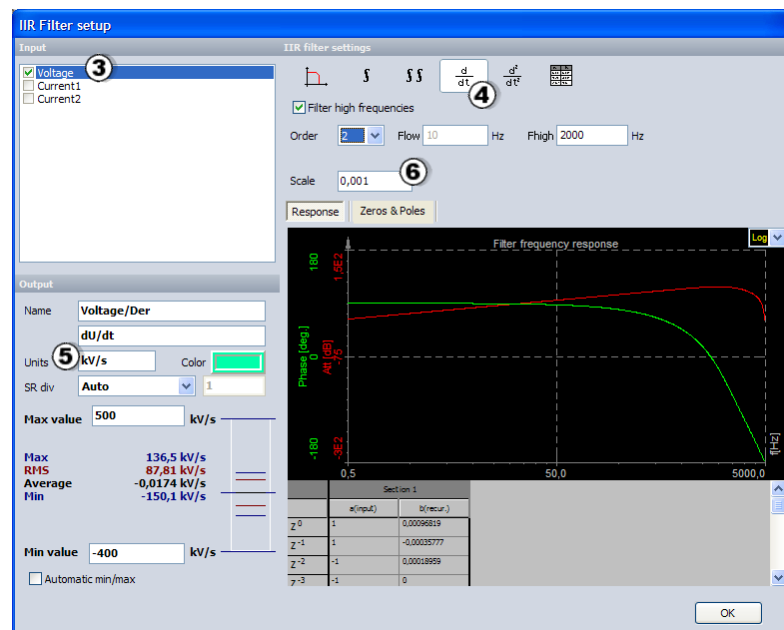
Returning to the **setup** screen, select the **Math** tab and define the *IIR filter*. If the **IIR** icon ① is not visible, it's either that the filter last used was a FIR or FFT filter. Select **IIR** from the drop down menu.



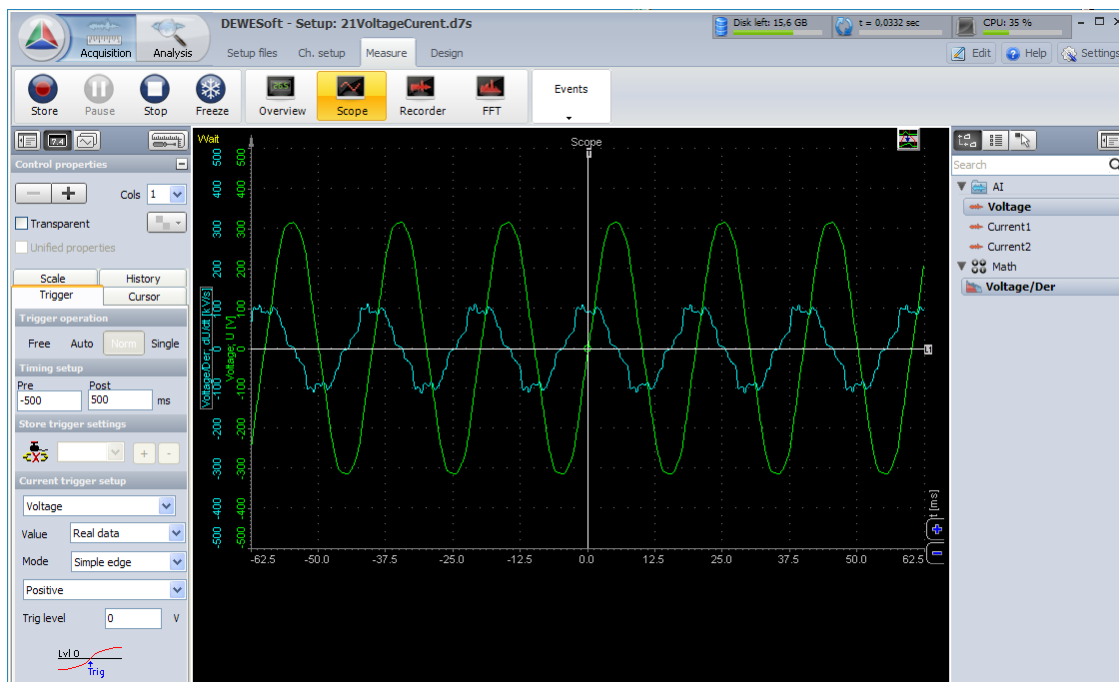
Now let's click the **Setup** button ② to define the filter. We select **Voltage** ③ as the *input*, then selects the **d/dt** icon ④ for the calculation of the derivative. Then the user defines the *units* as **kV/s** and defines in **Scale** field the *scaling factor*. Normally, if the input unit is **V**, the output will be in **V/s**. Since the values will be simply too high, we set the unit to **kV/s** ⑤ and define the scaling factor as **0.001** ⑥.

The *high frequency filter* (**Filter high frequencies**) is not mandatory, but it helps to *smooth out* the signal. If this filter is not used, the derivative will be calculated as a simple difference *from current* to the *previous* sample. This often adds lots of

noise, so we could *add an additional low pass filter* (**High** field) to *smooth* the signal out at the output.



If we now observe the *result* in the *scope*, we can see what appeared to be a nice sine wave, but is *not* really a pure sine wave. The derivate of a pure sine wave is *again* a sine wave, with a phase *shift* of 90°. This, however, is far away from it.



Let's repeat the same *scope* practice with **filtered** data. Was it noticeable in the picture on the first page that there is a big capacitor for electric motors? This is not included by sheer chance. We can use it for making *voltage glitches*. When the capacitor is charged, it takes lots of power from the grid, so there should be a *local voltage drop*. In this example, there is just the capacitor, which is connected to the normal line plug, where it is charged by inserting it BRIEFLY into the power supply.

To try this, please remember the two most important things: first, the user should insert the capacitor *only for a short*

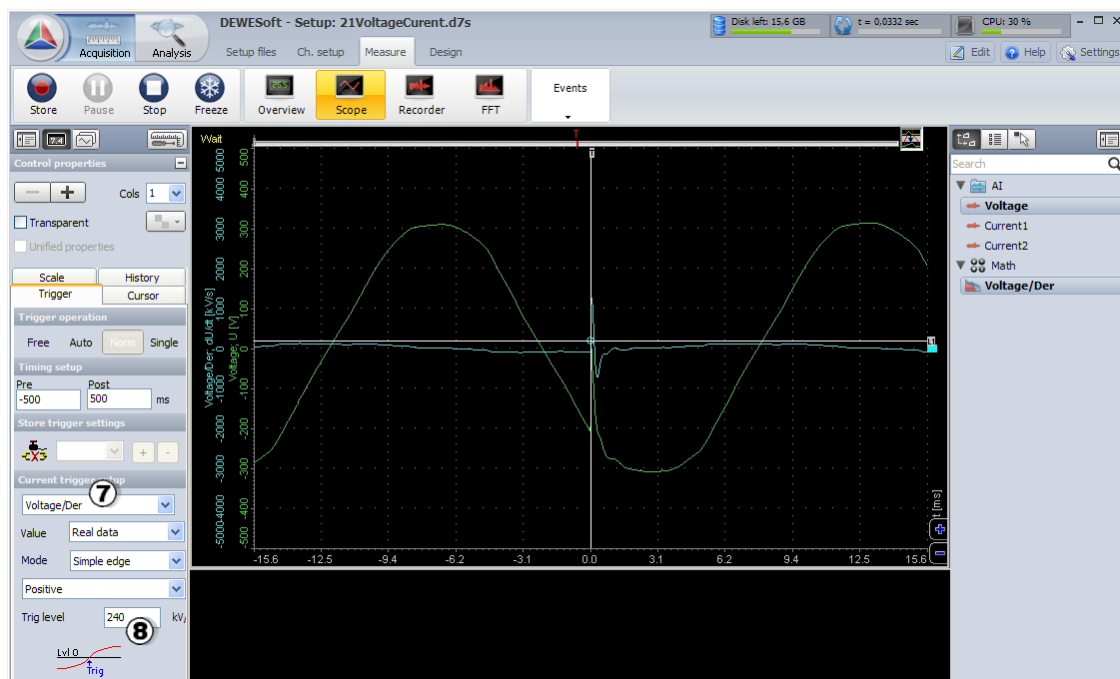
amount of time, else it could blow up. Secondly, **ALWAYS DISCHARGE** the capacitor on a piece of metal! Since these capacitors can hold substantial amounts of power, it is not very wise (and potentially quite dangerous) to discharge it on yourself.

If you try to do this, please remember two most important things: first, you need to insert the capacitor *only for a short interval*, otherwise it might blow up. Second, **ALWAYS DISCHARGE** the capacitor on a piece of metal. Since these capacitors can hold substantial amount of power, it is not very wise to discharge it on yourself.

WARNING *Touching both poles of such a capacitor when charged is equal to holding the line voltage, which can be extremely dangerous!*



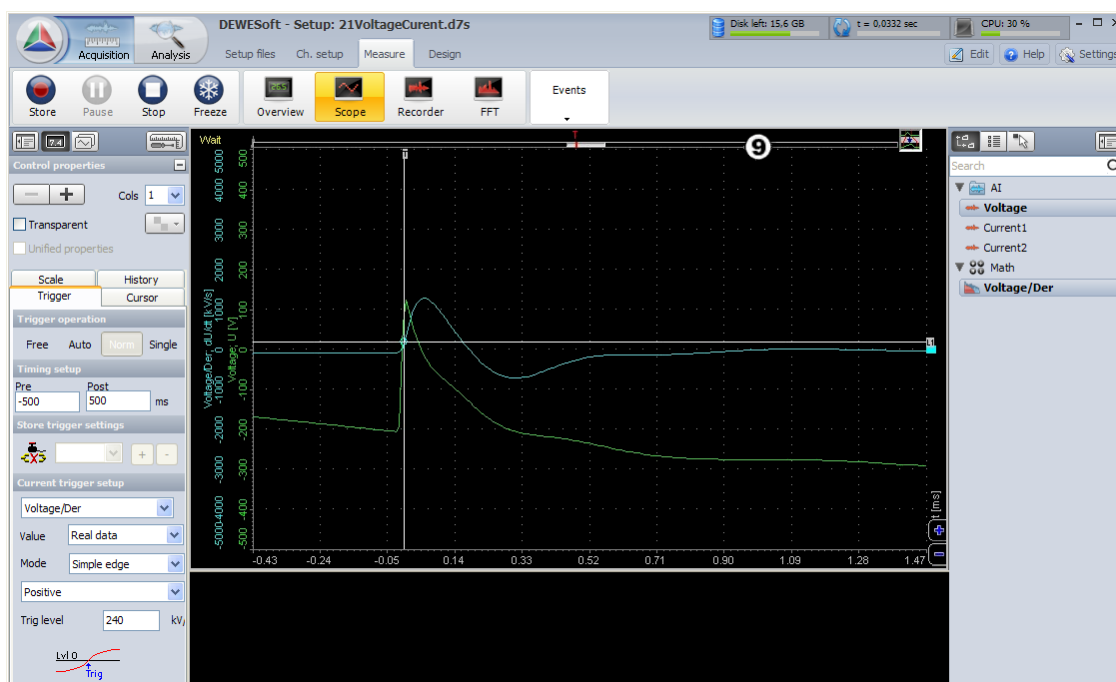
To **set up** the scope - the *trigger* parameter is changed to the **Voltage/Der** ⑦ channel and the **Trig(ger)** level is changed to **240 kV/s** ⑧. Now we create some glitches where we easily see how the scope catches these events.



The next logical thing is to be able to **zoom in** to see a glitch with a better resolution. If we click "zoom", we will lose the current event. There is, however, a mode which zooms in the scope *without losing an event*. On the upper right side of the

scope there is a **"zoom"** button ⑨ which enables an *additional zoom* window.

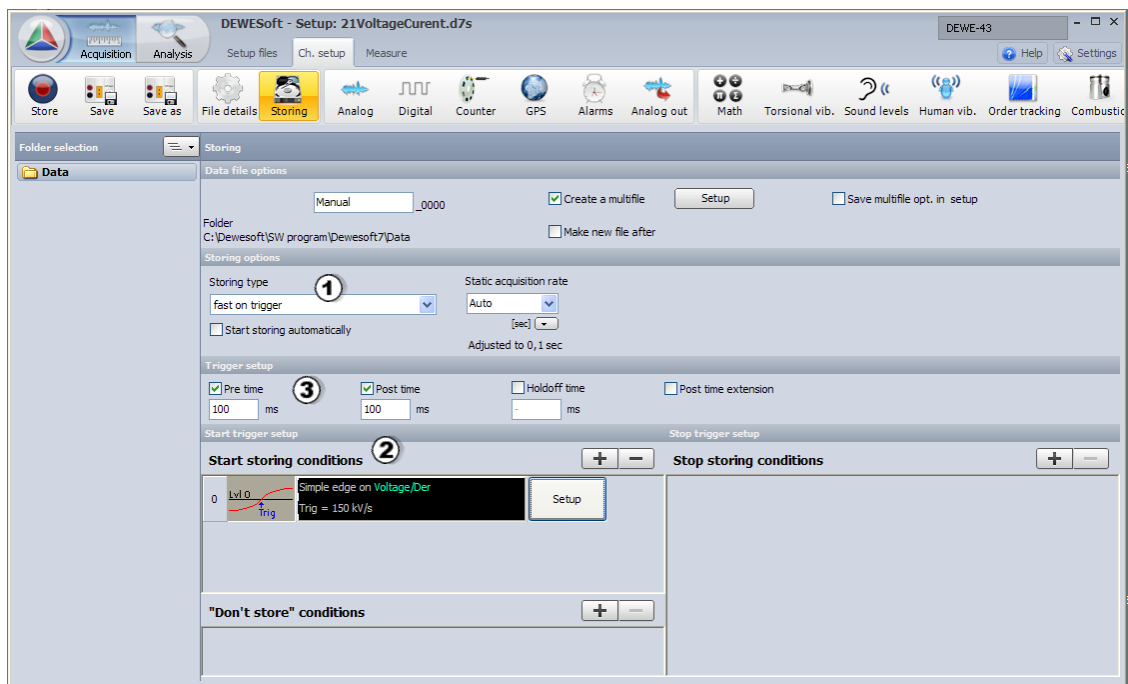
On the top, we can see the **bar** of the *current position in the event* and with the **zoom in/out** buttons; one can now zoom in on the *specific region*. We can also *move* left or right in the scope picture to see some part of the trigger shot.



Let's learn how to *store* this data.

2.1.4 Triggered storing

It is necessary to have **high sampling rates** (see previous chapter [Channel setup](#)) for the detection of glitches. These high sampling rates are related to the total waste of disk space. If the data is *always stored fast*, the user might end up with gigabytes of data, of which there might be only few seconds of useful data. Therefore, it makes sense to *store the data only when something is happening*. To do this, we should select the **Fast on trigger** ① storage option. The user can define the **"start storing condition"** ②, which is the **level trigger** when the voltage derivate exceeds **150 kV/s**, as well as including **Pre time** and **Post time** ③. Pre time is the time that the data will be stored *before* the trigger occurred while the post time is the time *after* the event. If the post time is not defined, data will be stored until there is a **manual Stop** or the **Stop storing condition** occurs (in this case, none was defined). For glitches, it is enough to have only some data before and after the event happened.



This is the *manual* way of setting the **trigger**. We can achieve the same by simply clicking the **Lock trigger** button

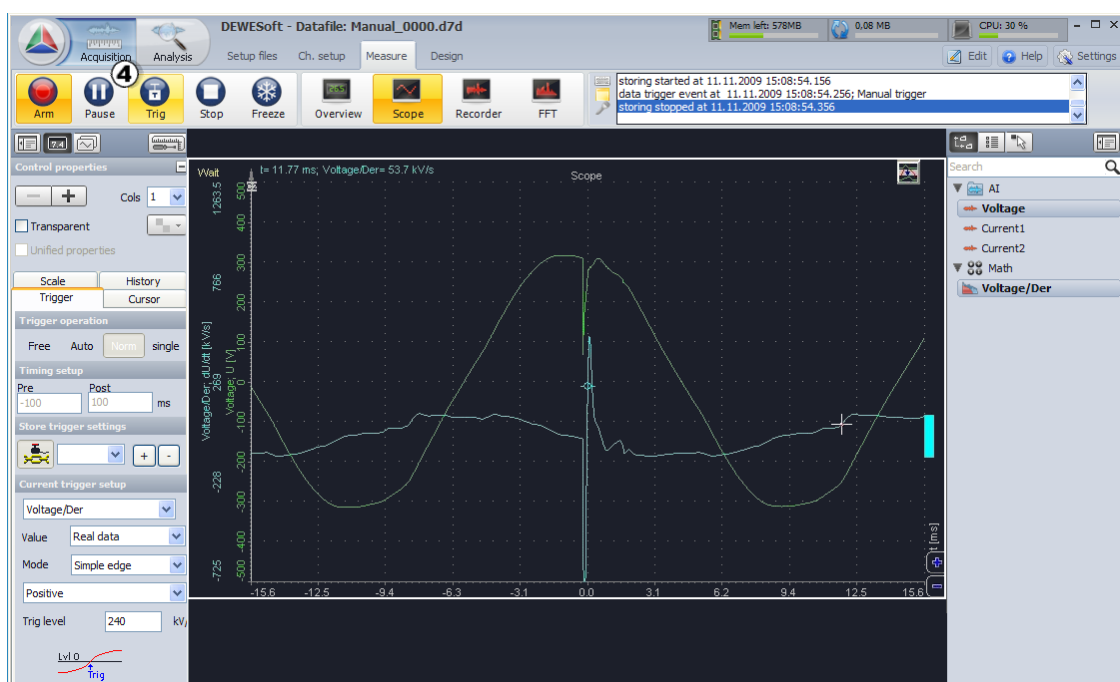


in the **Trigger** section of the *scope*. This will use the current *pre* and *post time*, trigger *source* and trigger *level*.

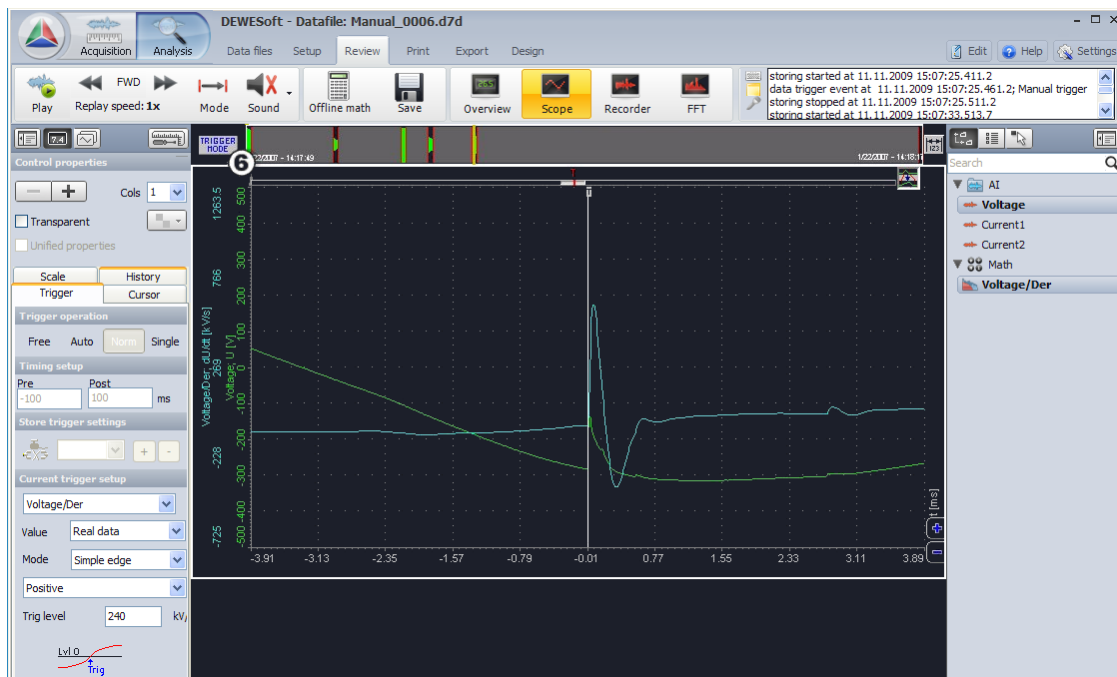
Either way, we now have to set specific *trigger criteria*. Now it's time to **store** some data. When we click **Store** an additional button **Trig** ④ becomes visible, which tells us that triggered storing is being used.

We can also click this button to a issue *manual* trigger.

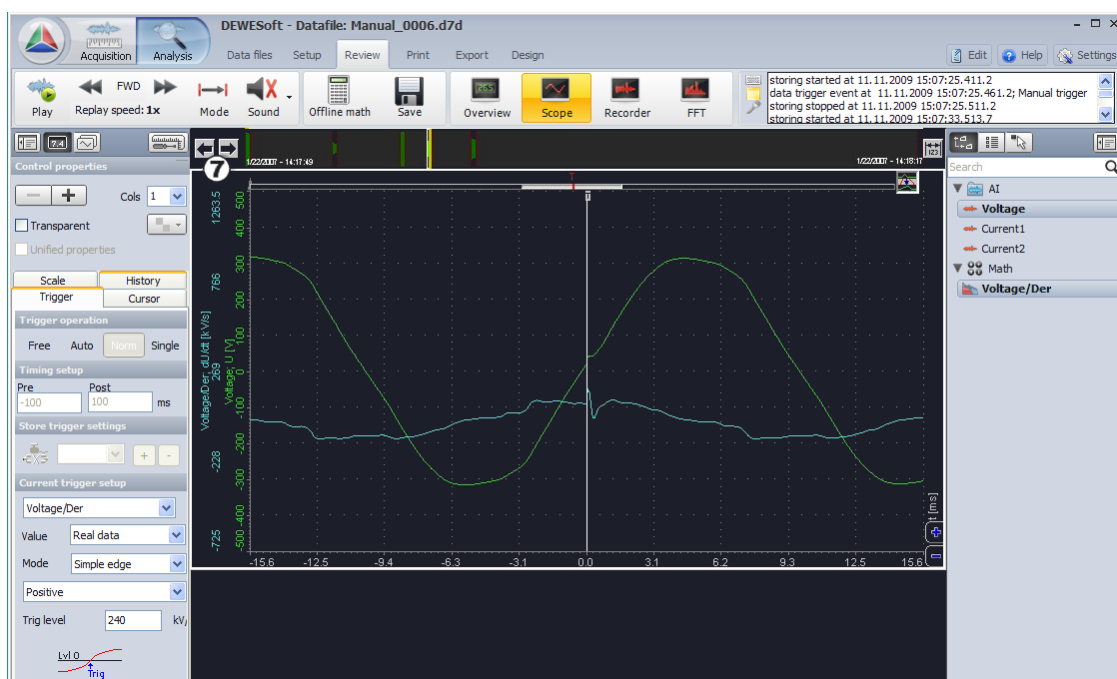
Now let's make some trigger shots. These will appear in the scope, and the **Trig** button *flashes*.



When we **open** this file, the *first trigger* is immediately shown in the *scope*. However, the *whole* zoom area is selected (in the *recorder*, for example). Now, the user has to click the **TRIGGER MODE** ⑥ button to view the data *trigger by trigger*.

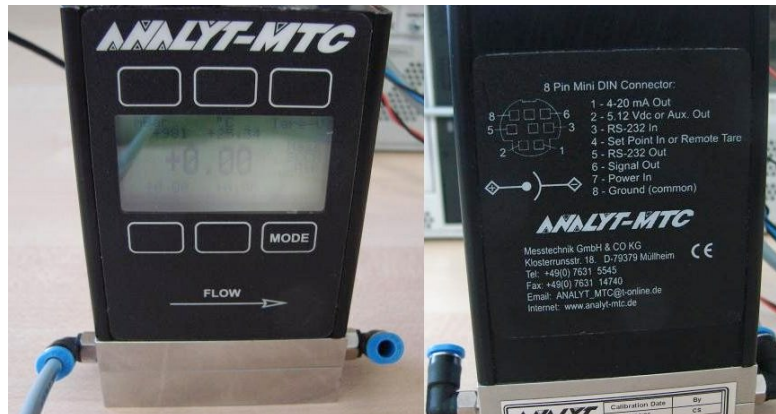


Then this button *changes* to a "**left**" and "**right**" button ⑦. By clicking these buttons, the user can **browse** through *trigger events*. One can also use the *scope zoom* to zoom in on specific regions of the trigger. The example below shows how a trigger on derivate reveals a small distortion in voltage really nicely.



2.2 Voltage/current/digital output sensors

In the previous tutorial we saw how to measure **voltage** and **current**. Often, there is a **sensor** which has voltage or current *output* and we need to measure that voltage and *convert* it back to real engineering units. Let's take a look at one simple example - an *air mass flow* sensor. This sensor has a **4÷20 mA** current output, a **0÷5 V** voltage output and also a **RS232** interface.

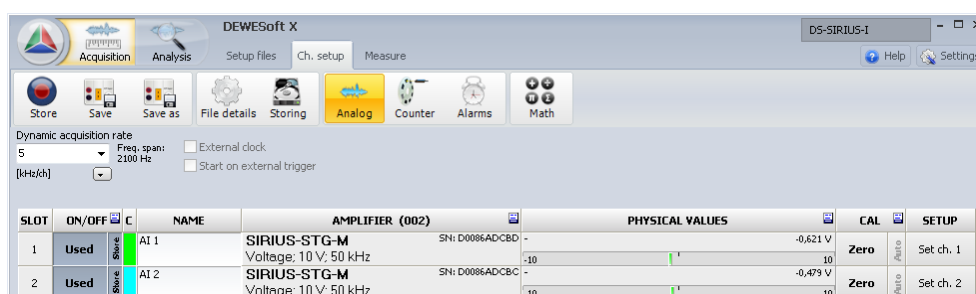


All three interfaces are connected. The **blue** and **black** wire is *voltage output*, and it goes to the Sirius MULTI module. The **9** pin voltage modules have the added benefit of also providing voltage for charging the sensors. If the sensors don't exceed current consumption specs, they can be charged *directly* from the module. The **red** and **black** wire is the *current output* and it goes to the same MULTI module, but in the front, we can see a shunt resistor. The thick **gray** cable is the RS232 connector, which goes to the RS232 port. In **DEWESoft**, the **plug-in** is written to **read** that data.

2.2.1 Channel setup

| | |
|-------------------|---------------------------------|
| Required hardware | Sirius MULTI, ACC, STG, DEWE-43 |
| Required software | Any license, except LT |
| Setup sample rate | At least 1 kHz |

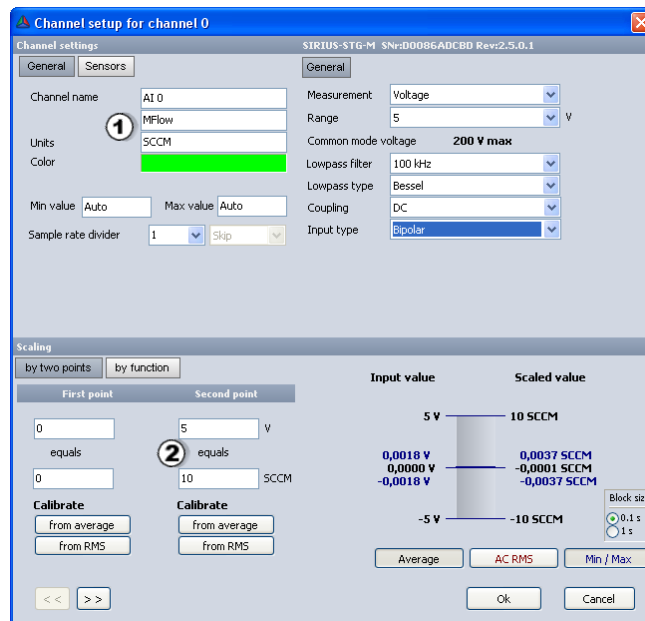
This example uses a simple **channel setup** with two channels - the first one is the *voltage input* and the second one is the *current input*.



First, let's **scale** the *voltage input*. From the specification we see that the maximum **range** is **10 SCCM** (10 standard cubic centimeters per minute). The *voltage output* has **0 V** for **0 SCCM** and **5 V** for **10 SCCM**. For the *current output*, **4 mA** is **0 SCCM** and **20 mA** equals **10 SCCM**.

| Analyt-MTC | | | |
|--|---|---------------------|----------------|
| Calibration Data Sheet | | | |
| Certification Number: 0008080 | | | |
| Sales Order #: | SO107933 | | |
| Serial #: | 24448 | | |
| Catalog #: | 35804 | | |
| Software Version: | GP07R48 | | |
| Process Gas: | Selectable | | |
| Calibration Gas: | Air | | |
| Range: | 10 SCCM | | |
| Gas Temperature: | 24.0°C | | |
| Ambient Humidity: | 30% | | |
| Calibration Procedure/Rev. #: | DOC-AUTOCAL-GASFLOW/Rev.12 | | |
| Calibrated By: | David Lashbrook | | |
| Calibration due 1 yr. after receipt: | 05/06/2005 | | |
| Calibration Date: | 05/06/2005 | | |
| Calibrators Used | | | |
| Flow Standard: | TOOL-MOLBOX1 | Flow Standard: | TOOL-MOLBLOC1 |
| Tool Due Date: | 07/13/2005 | Tool Due Date: | 07/13/2005 |
| Manufacturer/Model: | DHI / Molbox1 | Manufacturer/Model: | DHI / 1E1-VCR |
| NIST #: | 35865 | NIST #: | 35732 |
| All test equipment used for calibration is NIST traceable. | | | |
| Calibration: | | | |
| Required Accuracy: | +/- 0.50% of Full Scale = +/- 0.05 SCCM | | |
| Calibration Pressure: | NA PSIG | | |
| Actual | Indicated | Full Scale Error | Primary Output |
| 0.00 | 0.00 | 0 | 0.000 |
| 2.52 | 2.50 | -0.2 | 1.250 |
| 5.03 | 5.00 | -0.3 | 2.500 |
| 7.49 | 7.50 | 0.1 | 3.750 |
| 10.00 | 10.00 | 0 | 5.000 |

In the setup for the measurement of *voltage*, the user should define the **units** ① of measurement in **SCCM** and then sets the **Scaling**: **5 V** equals **10 SCCM** ②. That's all.



Channel setup for channel 0

Channel settings: SIRIUS-STG-M SNr:00086ADCB0 Rev:2.5.0.1

General

Channel name: AI 0

Units: **SCCM** (1)

Color: [Green]

Min value: Auto Max value: Auto

Sample rate divider: 1 Skip

Sensors

Measurement: Voltage

Range: 5 V

Common mode voltage: 200 V max

Lowpass filter: 100 kHz

Lowpass type: Bessel

Coupling: DC

Input type: Bipolar

Scaling

by two points by function

First point: 0 equals Second point: 5 V equals 10 SCCM (2)

Calibrate: from average from RMS

Input value: 5 V, 0.0018 V, 0.0000 V, -0.0018 V

Scaled value: 10 SCCM, 0.0037 SCCM, -0.0001 SCCM, -0.0037 SCCM

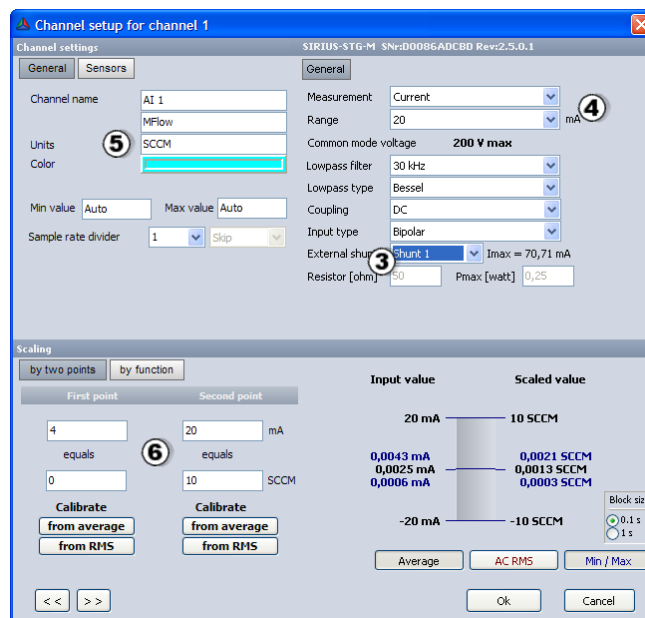
Block size: 0.1 s 1 s

Average AC RMS Min / Max

Ok Cancel

For *current input*, we first need to set the **Shunt** type. The shunt is *not recognized automatically* by the module, so we need to enter the shunt resistance. In this case, a standard **Shunt 1** with **50 Ohm** (3) is used, but a custom shunt could also be used. We only need to know the exact shunt *resistance*. Note that the input of the module changes from **V** to **mA** (4).

Then we enter the **units** of measurement (**SCCM**) (5) and the **Scaling**. It is very convenient to use **two point scaling**, where we enter the *current limits*: **4 mA** equals **0 SCCM** and **20 mA** equals **10 SCCM** (6).



Channel setup for channel 1

Channel settings: SIRIUS-STG-M SNr:00086ADCB0 Rev:2.5.0.1

General

Channel name: AI 1

Units: **SCCM** (5)

Color: [Cyan]

Min value: Auto Max value: Auto

Sample rate divider: 1 Skip

Sensors

Measurement: **Current** (4)

Range: 20 mA

Common mode voltage: 200 V max

Lowpass filter: 30 kHz

Lowpass type: Bessel

Coupling: DC

Input type: Bipolar

External shunt: **Shunt 1** (3) Imax = 70,71 mA

Resistor [ohm]: 50 Pmax [watt]: 0,25

Scaling

by two points by function

First point: 4 equals Second point: 20 mA equals 10 SCCM (6)

Calibrate: from average from RMS

Input value: 20 mA, 0.0043 mA, 0.0025 mA, 0.0006 mA

Scaled value: 10 SCCM, 0.0021 SCCM, 0.0013 SCCM, 0.0003 SCCM

Block size: 0.1 s 1 s

Average AC RMS Min / Max

Ok Cancel

2.2.2 Measurement

Now we can do some measurements. When performing this test, simply a *tube* where air was blown into was used, and the result was recorded. The **green** curve is the voltage input, the **red** curve is the current **input** and the **blue** curve is RS232 input.

Since this measurement is quite *slow*, we just simply store the results with no triggers. Basically the *recorder*, some *digital meter* and maybe a *bar graph* is enough to make a **visualization** of the results. Let's store some data and observe the differences between the traces.

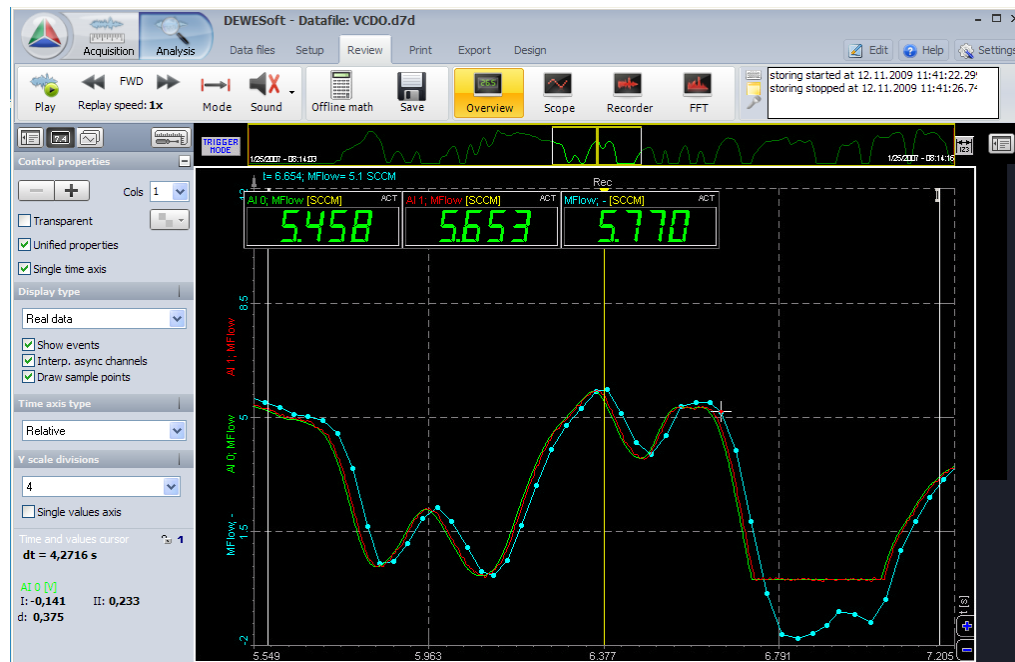


2.2.3 Analysis

When **loading** a data file and **zooming** in a specific portion, we can see that the curves don't overlap completely. There are three major differences. First we can observe the differences in *amplitude*. This comes from the fact that this **sensor** is already quite old already, and the calibration has period expired, therefore the *voltage output* might *not be exact*.

Secondly, we can see that the **RS232** data has a *delay*. This is quite normal, since the data has to be transmitted through a *slow* digital interface. A well-written *plug-in* (a driver for the device inside **DEWESoft**) can compensate for this delay. Also, the rate of data flow is *much slower* with RS232 than with the analog.

Finally, we can see that the *voltage* and *current output* doesn't output the *negative mass flow*, even though the sensor supports it.



So, what is better to use, *voltage* or *digital* interface? It would be incorrect to state that it is ALWAYS better to use one interface, because this depends a lot on the sensor itself. First, the user should look how the **sensor** is built. If it is an *analog* sensor in the first place and the *voltage* output is the *main output*, it is logical to use this output. The drawbacks for using analog output is that we can have *noise*, problems caused by long **cables** and the *voltage inputs* needs to be *recalibrated* over time.

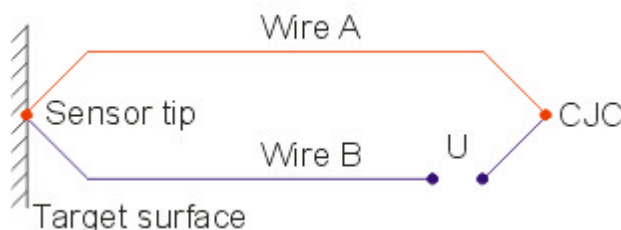
If the sensors have a *digital interface* as the default and they have DA converters to make *analog output*, then it makes *no sense* to use this analog interface since this results in nothing but losses. Of course, the use a of digital interface requires some extra work in *writing the drivers*, if the interface is not standard (like **CAN bus**, for example).

2.3 Temperature

A **temperature** measurement is one of the most common measurements. There are many different **sensors** available for this. Some of the most common ones are *thermocouple* (K, J type), *thermistor* (PTC, NTC), *thermal resistance* (RTD) and others. Let's look at the two most common types.

Thermocouple

First is the thermocouple, where we connect **two wires** together. Due to the *difference* in material *properties*, we can measure a small **voltage** (in a region of *millivolts*) on the open ends of the wires. We need to connect two different materials on the *hot* point as well as on the *cold* point, and then the voltage is almost proportional to the *temperature difference* between those two points. The cold point should have a *known* temperature. For this purpose it is necessary to measure the temperature at the cold point. This is called **CJC** or **cold junction compensation**. The *voltage* is also *not linear* with temperature and therefore we need to perform a **linearization**, either in the *hardware* or in the *software*. The *accuracy* of such measurement with "home built" sensors is usually not better than 1 deg C, but we can increase this by using thermocouple sensors.



The temperature **range** of thermocouples is huge. Depending on the type we can measure up to 1800 deg C. There are different types available, but the most commonly used one is K type, which has a range from -200 to 1200 deg C, while type E is better suited for *low* temperatures. Type N has similar properties as K type, but better stability and better resistance to *high* temperature oxidation.

B, R and S include platinum, and are therefore more expensive, but they offer better *high* temperature stability and are used for these applications.

T type consists of copper and constantan. Since they are both *non-magnetic*, they are often used to take measurements for *generators* with strong magnetic fields.

RTD

RTD stands for *resistive temperature detectors*. The theory behind this sensor is that the **resistance** of the *sensor* changes with temperature. These sensors are more expensive than thermocouples, and can only go up to 600 deg C (800 in some versions), but they are more *accurate*. The best known RTD is the Pt100. The name tells us that the base material is Platinum and nominal resistance is 100 Ohms at 0 deg C. The *accuracy* is better than thermocouples - below 0.3 deg C.

This tutorial will now demonstrate how temperature measurements can be performed in DEWESoft. On the next page, we will even be invited to a cup of tea!

2.3.1 Channel setup

| | |
|--------------------------|--|
| <i>Required hardware</i> | Sirius MULTI, STG, STG-M, DEWE-43 with MSI-BR-THK, CPAD, PAD |
| <i>Required software</i> | Any license except LT |
| <i>Setup sample rate</i> | At least 5 kHz |

The picture below shows a typical configuration for temperature measurements. We can see two temperature sensors coupled to Sirius via MSI BR THK.



For this example, there is also a cup of tea, just to make high temperature difference visible in the response of the channels. The measurement of temperature in *liquid* is one area of measurements, but an even bigger area is the measurement of *surface* temperatures. The mounting of the **sensors** is done with strapping, screwing, gluing or any other method providing a *good contact* between the surface and the sensor. This could often be a problem if the surface has some voltage potential, but since the Sirius is isolated, this is not a problem at all. *Isolation* of Sirius is **1 kV**.

Go to the **channel setup** and click on *Set ch.* button of a temperature module you are using. Select appropriate range ① from the list. Since the *linearization* is done in the software on this choice, the user needs to set the module correctly; otherwise the *scaling* will be *wrong*. For the US market, we can select to set up the *conversion* from **deg. C** to **deg. F** under **Settings** → **Global setup...** An additional button ② appears in **Scaling by function**. When clicking this button, an *automatic unit conversion* is done (scaling factor **1,8** and offset **32**).

Channel setup for channel 7

Channel settings: MSI-BR-TH-K (SIRIUS-MUL) SNr:374208 Rev:1-A

General

Channel name: AI 7

Units: °F

Color: [Yellow]

Min value: Auto

Max value: Auto

Sample rate divider: 1

Measurement: Temperature ①

Range: -200 .. 1370

Lowpass filter: 100 kHz

AO Setup

Scaling

by two points

Scale (k factor): 1.8

Offset (n factor): 32 ②

Set zero

Output = k * Input value + n

Input value

Scaled value

| Input value | Scaled value |
|-------------|--------------|
| 1370 °C | 2498 °F |
| 20,59 °C | 69,07 °F |
| 20,50 °C | 68,91 °F |
| 20,40 °C | 68,73 °F |
| -200 °C | -328 °F |

Average AC RMS Min / Max

Block size: 0.1 s

Ok Cancel

Since there are two modules in the system, the user needs to set *both* and then the following channel setup appears.

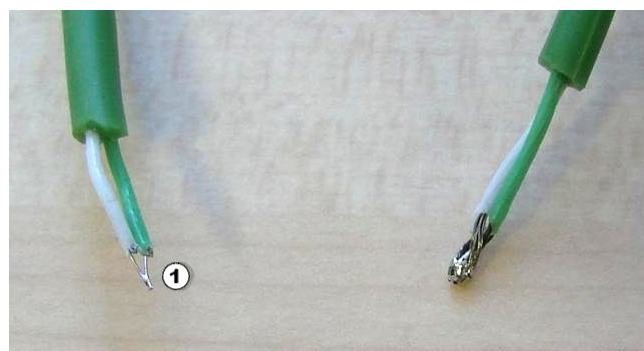
| | | | | | | | | | | |
|---|------|-------|------|--------------------------|-------------|------|---------|------|------|-----------|
| 7 | Used | Stone | AI 7 | MSI-BR-TH-K (SIRIUS-MUL) | SNr: 374208 | - | 69,4 °F | Zero | Auto | Set ch. 7 |
| | | | | -200 .. 1370 °C | | -328 | 2498 | | | |
| 8 | Used | Stone | AI 8 | MSI-BR-TH-K (SIRIUS-MUL) | SNr: 374149 | - | 21,4 °C | Zero | Auto | Set ch. 8 |
| | | | | -200 .. 1370 °C | | -328 | 2498 | | | |

There are two sensors installed in the system, so the user needs to set *both* of them to **Used**.

2.3.2 Measurement

Now let's do the measurement. It is best to start, since the tea is already cooling down. There are two **sensors** - one has quite a big *volume* on the tip, and another one has only two thin wires connected together ①. One important note at this point - the picture depicts an *amateur made* sensor. In reality, the two tips should not be soldered together, but rather *pressed*. This is because the third added material changes the properties and *only two original* materials should be connected together for highest precision.

But for this example, this is sufficient - we can see very nicely the difference between the sensor with the larger and smaller volume of the tip. The small tip volume will *increase* the *bandwidth* and *reduce* the *reaction time* of the sensor. Let's see how this works.

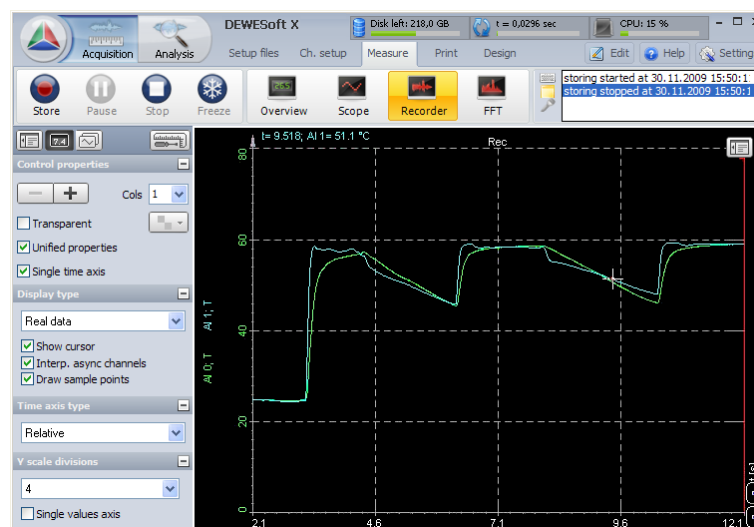


Let's put both sensors in the tea together and observe the results. We have already selected both channels in the channel setup.



In the *recorder*, we should **select** both *channels* while using the same axis for both channels (**single value axis**). Then we can change the values of **y scaling** to fit the measurement **range**.

The **blue** line is the *thin* sensor while the **green** line is the *thick* one. The reaction of two the sensors differs quite a bit when put in the tea. The temperature rises on the *thin* (blue) one almost *instantaneously*, while the green one *takes a few seconds* until the correct temperature is reached. On the other hand, the *thin* (blue) sensor is *more sensitive* to any ambient changes.



The temperature measurement is a *slow process*, and here we can use the **sample rate divider**. If there is a high speed measurement, combined with low speed temperature measurements, it is very useful to use the sample rate divider to *reduce* the amount of disk space used for storing. This is only important for measurements with MSI-TH, since the data from the PAD modules are asynchronous channels, and already **stored** with the *acquisition rate*.

2.4 Vibration

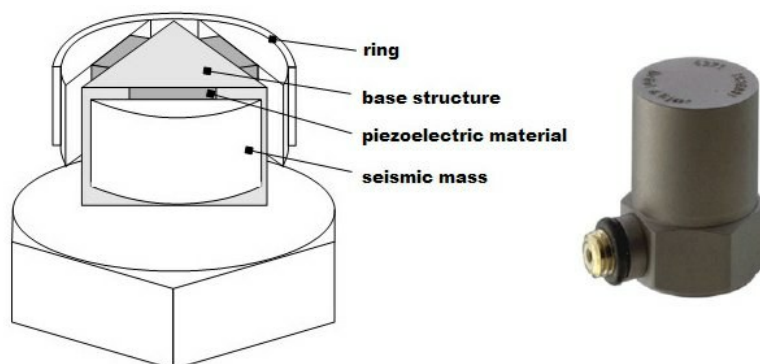
Vibration measurements are useful for the measurement of **surface vibration**. Many types of **sensor** can be used for this purpose. The most commonly used ones are *piezoelectric sensors* and IEPE sensors (with integrated signal conditioning). There are other brands for *low frequencies* (like *piezoresistive*, *capacitive* or MEMS sensors) and *non contact probes* (*eddy current probes* and *laser sensors*).

Piezoelectric sensors

Piezoelectric sensors work on the principle that a piezoelectric material is built between the bottom of the sensor housing and the seismic mass. When a sensor is moved, this mass *compresses* the piezoelectric material which *produces very small voltages*. To transfer those small electrical values through the cables require lots of knowledge and expensive cabling, therefore lately these sensors have often been replaced with the IEPE sensors with *integrated amplifiers*.

However, there are still applications areas where these sensors are very useful. These fields are especially *high acceleration and high temperatures*. The *amplitude measurement range* of such sensors can be *thousands of g*. One can find single axis as well as triaxial sensors.

Sirius MULTI, STG or DEWE-43 with MSI-BR-CH can be used, but please look at speed that dynamic range is sufficient for your application.



IEPE (integrated electronic piezoelectric) sensors

IEPE sensors need a *power supply* of 4 or 8 mA and they typically give out a 5 volt signal, thus it is much easier to transfer these signals over longer cables. Also, the *amplifiers* for those sensors are much easier to build, and therefore cheaper. The amplitude measurement **range** is quite *limited*. We can hardly find a sensor which measures more than 100g. We can find *single axis* as well as *triaxial* sensors. Lately, really nice **sizes** have become available - one can find a triaxial sensor as a cube measuring as little as 10 mm, and with the weight as light as 5 grams.

Again we can use Dewesoft Sirius or DEWE-43 to measure with these sensors. Sirius ACC can directly connect IEPE sensors. While STG, STG-M or DEWE-43 needs MSI-BR-ACC adapter to measure these sensors.



Static acceleration sensors

Both sensor types have a common *limitation*: they can't measure *static acceleration*. They usually start to measure from **0.3 Hz** to **10 Hz**, depending on the sensor. For *static* or *very low frequency* measurements, the user needs to use a different kind of sensor. Lately, a very popular type is the MEMS sensor. This is actually a microchip which has a mechanical structure (a cantilever beam or seismic mass) that changes its *electrical property* (usually capacitance) related to the acceleration. Not far in the past, such sensors were very special, since they were used to measure *earthquakes* or any other *slow movements*. But with the development of *airbag* technology, there was a big need to make a low cost sensor which measures the static acceleration. Therefore, the single chip solution emerged for this purpose. Lately these sensors are used also in low cost *gyro* systems and we can find sensors which also have quite good *bandwidth* up to **several kHz** and quite *low noise level* (though still bigger than IEPE sensors with the same measurement range).



Choosing a correct sensor

So what are the key points to consider when choosing the correct sensor?

Low frequency range

Usually for vibration measurements the requirement is that the sensor has a lower high pass cutoff than for the more interesting frequencies in devices currently being tested. With a rotating machine with 50 Hz, we can choose a sensor with a **5 Hz** cut off. It is also better not to have too little bandwidth, the lower the bandwidth gets, the longer the recovery times from shocks or overloads. Also, the amplifier should follow the bandwidth of the sensor. It is nice if the amplifier has at least two ranges in order to be more flexible in measurements.

A typical application for low frequency measurements are the *paper mill rolls*. They have a frequency of 1÷5 Hz, where the user would need a sensor with **0.3 Hz** or less bandwidth. For those applications *charge* or *IEPE* are the best.

If we need to measure **DC**, then we need a different sensor technology, like MEMS sensors is needed.

Bandwidth (high frequency range)

Bandwidth also depends on the application. If we are not interested in higher frequencies or we know there is not much going on there, it is alright to choose sensors with lower bandwidth. However, some applications like *bearing condition monitoring* require higher bandwidth up to **20 kHz** or more. *IEPE* or *charge* sensors are needed for those kinds of

applications. Also note that the sensor *mounting* affects the bandwidth. Read following section for more information on this subject.

Amplitude range

Charge sensors have the biggest amplitude ranges (up to 100 000 g or more), but IEPE are also fairly high (up to 1000 g). MEMS sensors usually have limited range (up to few hundred g). For general purposes, it is best to use IEPE, whereas for high levels piezoelectric sensors are better. Sometimes (for example for *seismic* applications) an accelerometer with *high sensitivity* is required (2 g or lower range).

Maximum shock level

The charge sensors are the least sensitive to shock. They can sustain up to 100 000 g of shock, while IEPE can usually take not more than 5 000 to 10 000 g. MEMS sensors are even *more sensitive* to shock.

Noise level

The residual noise level defines the *lowest amplitude level* of what the sensor will measure. This is also the reason why we should take a sensor with the optimum measurement range, because sensors with higher a range will also have a higher noise level.

Temperature range

All the sensors that include electronics have a *limited high* temperature range, up to 130 deg C. The temperature range of charge sensors is much higher - even up to 500 deg C. Please note, however, that this would also require high temperature *cable*.

Weight

In some applications, like *modal testing*, weight can be a big factor due to the *mass loading effect*. The added mass to the structure changes the dynamic behavior, so ideally a sensor should have no mass at all.

In other areas of predictive monitoring in harsh environments, a sensor needs to be *robust* enough to sustain possible damage or, as my good old friend would say, that you can still find it in the dirt.

Mounting consideration

There are sensors available with *holes* (usually threaded) or with a clip mounting.

Sometimes it is very important to use the *isolation*. Since IEPE amplifiers are usually not galvanic isolated, we need the isolation on the *side of the measurement*. There are also sensors readily available with a housing which is isolated to the connector ground.

NOTE: *The examples above are only for a basic understanding. There might be special sensors of specific types which exceeds those limits.*

Sensor mounting

The sensors can be mounted in different ways. Especially the *bandwidth* of the sensor is *affected* by the mounting.

Clearly, it is the best to drill a *hole* in the test specimen and *affix* the sensor to the surface with a *screw*. This should not affect any sensor property. Obviously, in some cases a customer might not be particularly thrilled to do this, for example, to his brand new prototype of an airplane wing. Another type of mounting, which doesn't affect the bandwidth that much, is *thin double sided adhesive tape* or bees wax (this is, however, limited in its temperature range).

A very widely used mounting technique for machine diagnostics is to mount the sensor *on a magnet*. This will still produce

a good bandwidth, but of course, the surface must be ferromagnetic (no aluminum or plastic). On sensors where we can use the mounting clip, we can glue the mounting clip up front and then just attach the sensor itself.

A "quick and dirty" solution is also to hold down the sensor with the a *hand on a rod*. This is useful for some places which are hard to reach, but the bandwidth will be *cut* to 1÷2 kHz.

2.4.1 Channel setup

| | |
|--------------------------|--|
| <i>Required hardware</i> | Sirius ACC or MULTI, STG, DEWE-43 with MSI-BR-ACC or MSI-BR-CH |
| <i>Required software</i> | PROF or DSA version |
| <i>Setup sample rate</i> | At least 5 kHz |

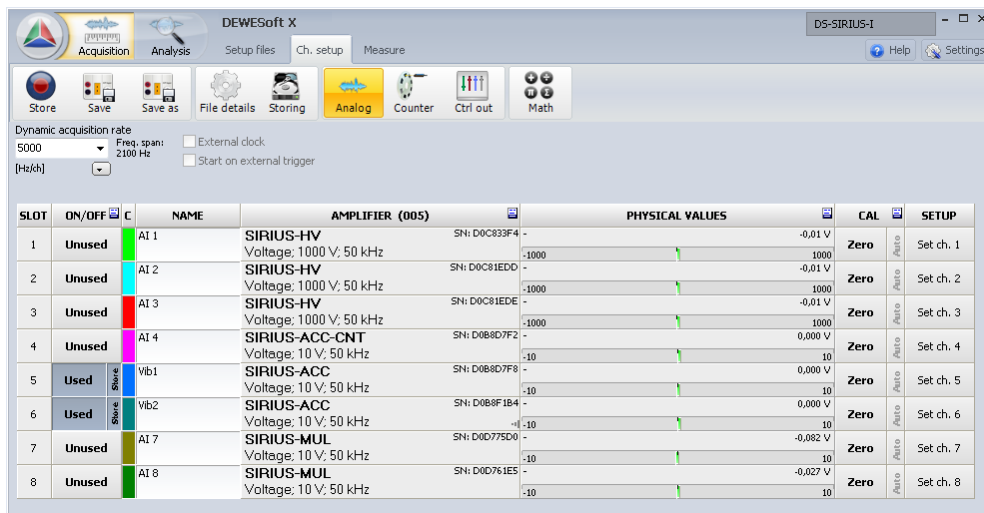
Let's do some **vibration** measurements in **DEWESoft**. Since vibration is rather difficult to visualize and, since there were lots of questions about the difference between *acceleration*, *vibration velocity* and *displacement*, it is helpful to actually **show** the vibration.

This example has a *shaker* with an attached light plastic structure that has a *low natural frequency*. At the same time, a *video* of the movement of this beam was taken with a *high speed camera*. This helps to really see the vibrations, as they were measured with the accelerometer.



It is always advisable to use a measurement device with *anti aliasing* filter. Otherwise, we can never be sure that the measuring is correct. Quite often acceleration in high frequency range (around 20 kHz) is very high. If a device without anti aliasing filters is used, and samples with lower sampler rates are taken, those high frequencies will be mirrored in the lower range. Especially for the measurements like *modal analysis* this is the most important criteria.

Below is the **analog channel setup**. There are two ACC modules we will use for the measurement of vibrations. Let's look how to scale the measurements.



Sensor setup

There are three ways to perform the **setup** of the sensor - first, the user can to enter it from the calibration sheet, second, the user can calibrate it with the calibrator, and third the user can use TEDS technology to read out calibration values.

1. Entering the setup from the *calibration sheet*: It is helpful to first take a look at the sensor calibration sheet. There is a sensitivity of the sensor, expressed either in mV/m/s^2 or mV/g (or both) for IEPE sensors and in pC/g for piezoelectric (charge) sensors. The picture below shows the calibration data sheet for *triaxial* sensor. The *Reference sensitivity* is the key value to enter in the **DEWESoft** setup.

| | | X- | Y- | Z- | axis |
|---|---------------------------|----------|-----------|-----------|---------------------|
| Reference Sensitivity ¹⁾ at 159.2 Hz ($\omega = 1000 \text{ s}^{-1}$), 20 ms ⁻² RMS, 4 mA supply current and ...23... °C: | | | | | |
| | | 9.863 | 9.988 | 10.14 | mV/ms ⁻² |
| | | 95.72 | 97.85 | 98.41 | mV/g |
| Frequency Range | Amplitude ($\pm 10\%$): | 0.2-5.5k | 0.25-3.0k | 0.25-3.0k | Hz |
| | Phase ($\pm 5^\circ$): | 1.5-3.0k | 1.5-3.0k | 1.5-3.0k | Hz |
| Mounted Resonance Frequency: | | 18 | 9 | 9 | kHz |
| Transverse Sensitivity: | | | | | % |
| Maximum (at 50 Hz, 100 ms ⁻²): | | < 5 | < 5 | < 5 | |
| Transverse Resonance Frequency: | | 9 | 9 | 9 | kHz |
| Calculated values for TEDS ²⁾ : | | | | | |
| F_{res} | | 18.7 | 9.52 | 9.56 | kHz |
| Q | | 8.88 | 1.68 | 1.67 | |
| Amp. Corr. | | -2.03 | -2.26 | -2.25 | %/dec. |
| F_{up} | | 0.010 | 0.010 | 0.010 | Hz |
| F_{sp} | | 3.00 | 3.00 | 3.00 | kHz |
| Measuring Range: +500 ms ⁻² peak (+50 g peak) | | | | | |
| Polarity of the electrical signals are positive for an acceleration in the direction of the arrows on the drawing. | | | | | |

First, as usual, we should enter the **Units** of measurement. In this case we use m/s^2 ①. Then it is the best to go to the **Scaling by function** section. We check the **Sensitivity** box ② and enter 9.863 mV/m/s^2 in the sensitivity field. Also do not forget to set IEPE measurement ③.

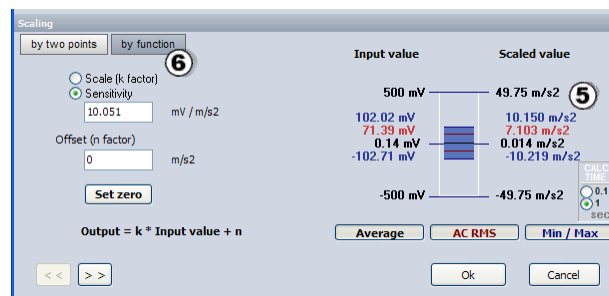
2. The second way is to *make a calibration*. We can use a standard old *Accelerometer calibrator* which *outputs* 10 m/s^2 peak level acceleration ($7,07 \text{ m/s}^2 \text{ RMS}$). The sensor is attached to the calibrator, and the acceleration level is adjusted to the sensor mass.



Then we enter in **Scaling by two points** the acceleration level of $7,07 \text{ m/s}^2$ and click calibrate "**from RMS**" ④. The current measured voltage level in **mV** is written to the second point scaling.



Then, we can already see if the calibration was successful or not. In the *data preview* we can see that the *peak level* is approximately 10 m/s^2 and the RMS is around $7,07$ ⑤. It is advisable to change the **CALC TIME** option to *one second* in the data preview to have more stable data. We can also select the **Scaling by function** ⑥ and *compare* measured sensitivity to the calibration data sheet.



3. The third, quite new way of sensor setup, is the use of an *electronic calibration sheet* - **TEDS**.

With a TEDS sensor, it is quite easy to select settings. *Plug in* the sensors in Sirius ACC, run **DEWESoft** and the sensors should be *recognized immediately*. TEDS works only if the amplifier is in IEPE mode (it doesn't work in the voltage mode). If this is set up later (after the first scan) or if we plug in the sensor when **DEWESoft** is already running in the **setup** screen, the TEDS sensors need to be **rescanned**. This can be done by clicking on the **AMPLIFIER** column caption on the basic **setup** screen and selecting the **Rescan modules** option. Note that with MSI-BR-ACC, TEDS will not work.

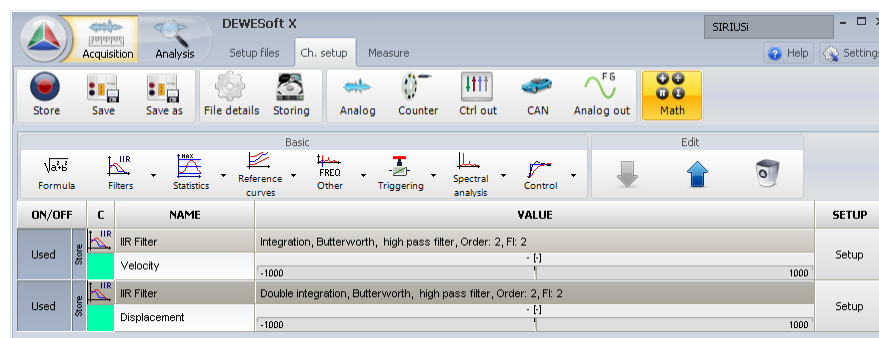
When a sensor is correctly recognized, *scaling factors*, *sensor serial number* ⑦ and **Recalibration date** ⑧ will be read from the sensor. In the setup screen, the user doesn't have to enter the sensitivity, since it is already filled in from the sensor. This principle is easy and straightforward, and it prevents user errors.



2.4.2 Math setup

The second step is to **calculate** the *vibration velocity* and the *displacement*. This can be achieved in the **math** section with the **filter**, since the integrator is actually nothing more than a filter.

We enter two **IIR filters** in the setup - first will be the *integrator* (for calculation of vibration *velocity*) and the second one will be the *double integrator* (for measurement of the *displacement*).



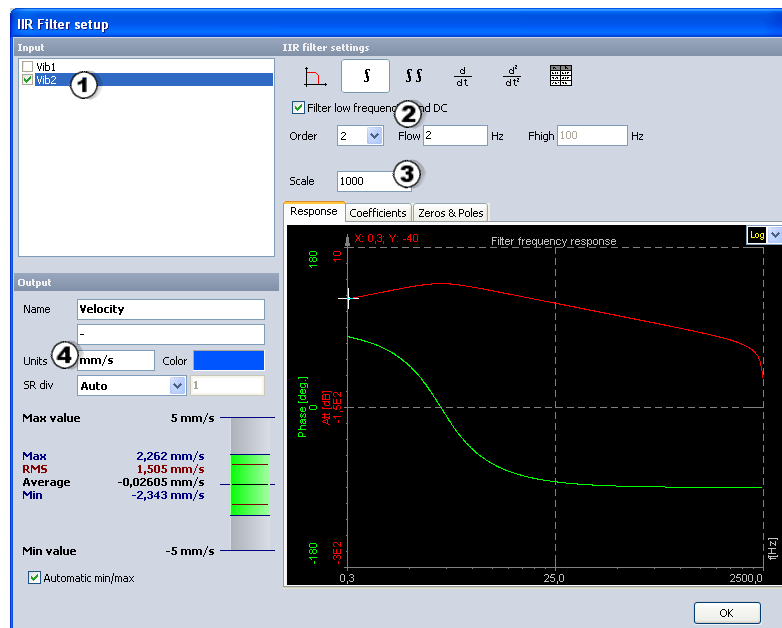
Let's go to the channel setup of the first filter. First, we need to choose the *input channels*. Since earlier the second channel was the upper *accelerometer*, we must deselect **Vib1** and select **Vib2** ①. It is quite a common error to forget to choose the correct input channel, so it is advisable to do this step first.

Then we should choose the *integrator*. Since the **DC** offset is merely an error in measurement and calculation, we need to set up the *high pass filter* (in **Flow** field) to cut off the DC offset. For *single integration*, the **Order** of the filter needs to be at least **two** ② (if filter order is one, there will be static offset left in the result, if there is no filter, it will drift away).

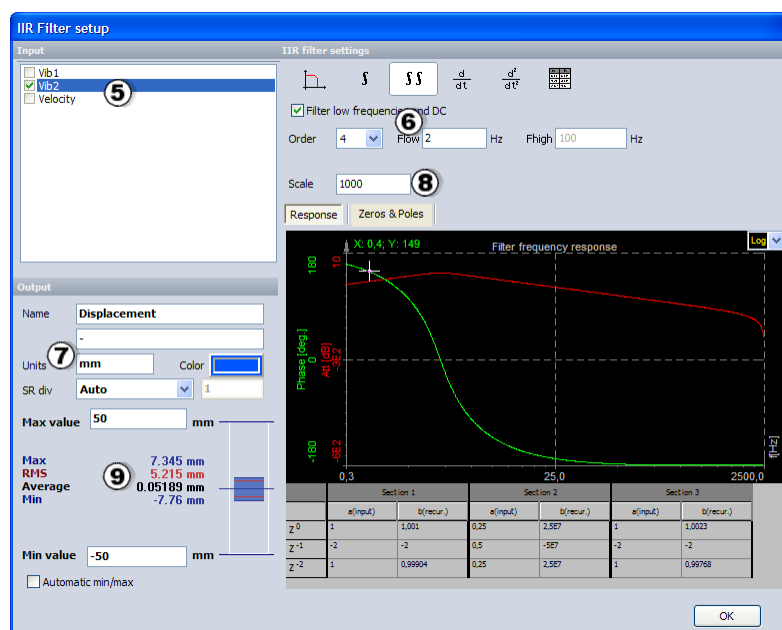
Next, we enter the **units**. If the integration is from acceleration to velocity and the acceleration unit is **m/s²**, the output unit is normally **m/s**. Since this unit will be too large, it is helpful to enter the *scaling* factor of **1000** ③ and then have **mm/s** ④ as

the units.

As can be seen in the preview, the vibration on the shaker is quite large. On normal machinery (where the vibration is an unwanted side effect), the vibration value of velocity should not exceed $10 \div 15 \text{ mm/s}$.



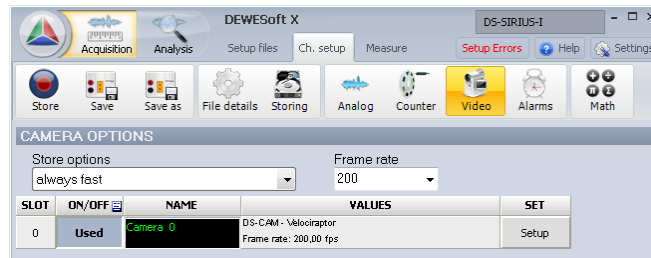
Since the movement of the structure will be observed with the *camera*, it would be interesting to know the vibration *displacement*. For this, we should setup another channel by again selecting **Vib2** ⑤ and selecting **double integration**. Since the double integrator is in fact a second order filter, we need to set the **high pass filter** ⑥ to the **Order** at least three or higher. Usually the displacement caused by the vibration is not visible by the eye, and is measured in **micrometers**, but since this measurement has quite high values, the output **unit** was set to be in **mm** ⑦. The **scaling** factor is therefore again **1000** ⑧. We can already see in the *preview* that the **peak-peak** movement is around **15 mm** ⑨ and since this is a value which can be confirmed with the eye, we can be sure that the scaling factors and the settings are correct.



Now the video camera needs to be set up and then we can see what's going on.

2.4.3 Video setup

We have fast DS-CAM camera connected in this example. First we set it to **Used** on **Video** tab of the **DEWESoft-Setup** screen, and then create a setup for this camera by clicking the **Setup** button.



We set the **Frame rate** to **200** ① to have enough speed to see the vibration. The most important setting after the frame rate is the **Shutter** speed ②, because this defines the time it takes for the camera to acquire the picture. If we acquire fast movement, shutter speed needs to be faster, but then we need a stronger light.

Then, we can set the **Gain, Brightness** and other advanced settings ③ according to current light conditions.



Now let's acquire some data.

2.4.4 Measurement

Let's set up the display to have a *recorder*, *scope* and *camera picture* ③ in one display.

On the *recorder* ① we see nicely how the acceleration, velocity and displacement are *phase shifted* by **90 deg** one to another. This is in accordance to the theory - if sine function is integrated, cosine results.

We use the *scope* ② to see the *base* acceleration of the shaker (green curve) and the *top* acceleration. Since the natural frequency is **90 deg** phase shifted to the base movement, we can use this scope to set the correct **frequency** of the *function generator* to have maximum response.



For the vibration measurement one of the most useful displays is the *FFT* screen ④. *FFT* decomposes the waveform to the series of sine waves and there we can clearly see the *cause of the vibrations* (on a rotating machine we can clearly see unbalance, misalignment, bearing faults and other errors). With *modal* testing, we can easily see *natural frequencies* in *FFT* and it can be used in a special form (*3D FFT*) as *order tracking* for *run-up's* and *coast downs*.



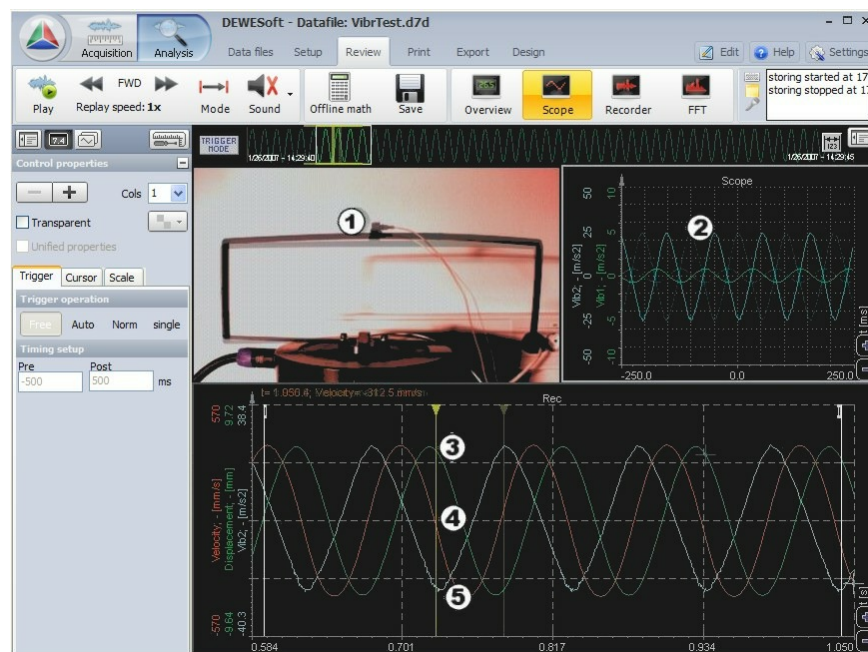
With this test, the only point of interest thing to see is the non linearity of the excitations (errors of the sine wave), since the shaker output should be a pure sine wave.

2.4.5 Analysis

In the analysis mode we can **look** through the **data**. Here, one *picture* is put on top of another to see the *movement* of the beam. The first picture below is the *upper* point ① of displacement.

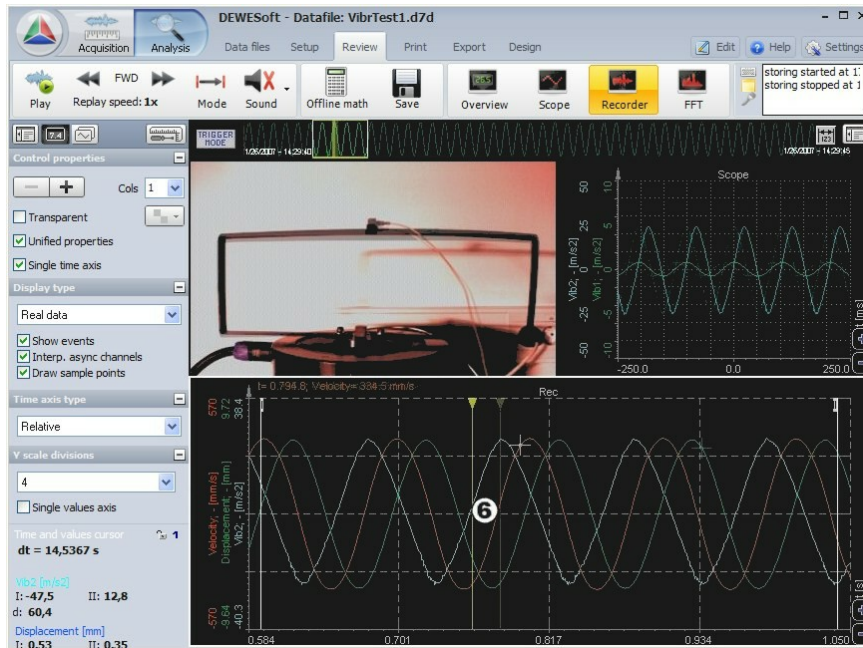
On the *scope* on the right, we can see nicely that the Vib1 and Vib2 are phase shifted for **90 deg** ②. Thus, the user is in the clear region of the *natural frequency* of this structure.

On the *recorder* graph below, we can analyze the acceleration, the velocity and the displacement. The displacement (*green* curve) is in the *upper* position ③, which can be visually confirmed with the *video* ①. The velocity (the *red* curve) is *zero* ④ - this is also clear because the upper point is a *turnaround* and before reaching this point on the top, the velocity is decreased and at the top point, the velocity is zero. The acceleration (*blue* curve) at the top is at *maximum* in the *negative* direction ⑤. Acceleration is the rate of *change of the velocity*. We can see from the velocity curve that the rate of change is at a maximum at the *top*; therefore the acceleration is as its *maximum at the top dead point*.

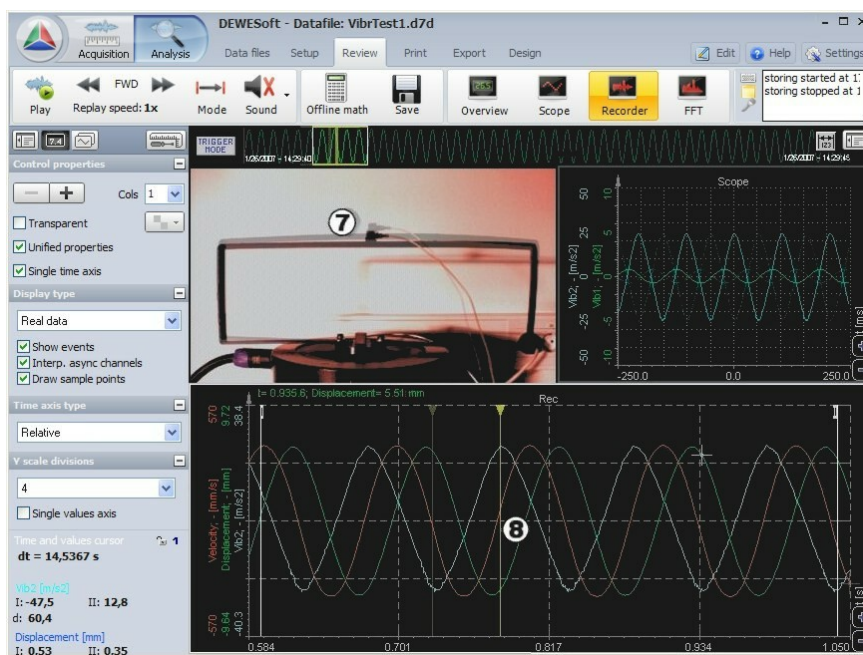


Now let's go to the next significant point of the movement - the *center* point. We can see that it is the center point because the displacement is in the *middle* ⑥. The velocity in the center is at a *maximum in the negative* direction ⑥. This makes sense: the beam is reaching the middle point with maximum velocity, and it will slowly start to decelerate.

Acceleration at that point is *zero* - when a body is standing still or moves with constant velocity, its acceleration is zero ⑥. This can be confirmed by observing the *blue* acceleration curve.



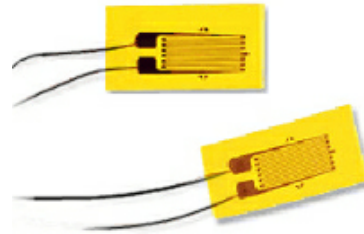
The third significant point is the *bottom* point. Here, a top point is shown in background for reference ⑦. The displacement is at the *lowest* point ⑧, velocity is *zero* ⑧ and will continue to increase, the acceleration is at a *maximum in the positive direction* ⑧ - the speed is changing at a maximum rate at this point.



Here, we conducted a simple experiment to get a better feeling about the vibration measurement. In practice, the vibration measurement would surely look different, but we would use the same basic principles as shown in this example.

2.5 Strain measurement

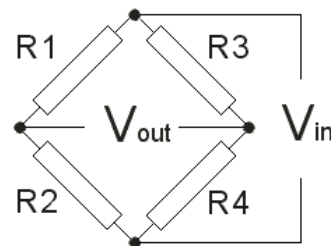
The strain gage measurement is a very nice principle of measurement **strain**, and with that also **stress of the materials**. Strain gages are basically *foil resistors* which are using the principle that the line resistance of the wire is proportional to the length and inversely to the area of the cross section. A typical strain gage material is *constantan* (copper-nickel alloy), which has a *constant resistance* over a big temperature range.



The change, however, of resistance with strain is very *small*, so we need a good *amplifier* and measurement principle to measure such small differences.

Wheatstone bridge

The most widely used principle to measure strain gage is the *Wheatstone bridge*. This bridge consists of four resistors. On the *inputs*, we supply the so-called excitation voltage and **measures** the voltage of the *outputs*. If the bridge is balanced, the output voltage will be zero.



There are now several ways to mount the strain gages. This depends a lot on the specific application. For a better understanding on how to use the software and what the values mean, let's take a look at the basic equation of the bridge.

$$V_{out} / V_{in} = \frac{R1 \cdot R4 - R2 \cdot R3}{(R1 + R2) \cdot (R3 + R4)}$$

One of the more common configurations of the bridge is a *quarter-bridge*. In this case, *only one* resistor is a strain gage while the other three are *fixed* resistors with equal resistance, which use the nominal resistance of the strain gage (this shows that the bridge is balanced). Therefore let's say that $R4 = R + dR$ and all the rest are equal $R1 = R2 = R3 = R$. Now let's put this in the equation above.

$$V_{out} / V_{in} = \frac{R \cdot (R + dR) - R \cdot R}{(R + R) \cdot (R + R + dR)} = \frac{R \cdot dR}{2 \cdot R \cdot (2 \cdot R + dR)} = \frac{dR}{4 \cdot R + 2 \cdot dR} = \frac{dR / R}{4 + 2 \cdot dR / R}$$

The $2 \cdot dR / R$ part is quite *small* (1% of elongation gives a 1% error), so this can simplify the equation to:

$$V_{out} / V_{in} = \frac{dR / R}{4}$$

The basic parameter of the strain gage is a so-called gage factor ***k***. This gage factor tells us the amount of change of the resistance in relationship to the strain.

$$k = \frac{(dR / R)}{(dL / L)} = \frac{dR / R}{\varepsilon} \quad \rightarrow \quad \text{therefore the } dR/R = k \cdot \varepsilon$$

If we insert this in equation above, we get:

$$V_{out} / V_{in} = \frac{1}{4} \cdot k \cdot \varepsilon$$

If the user has a different bridge configuration, this factor needs to be entered in the DEWESoft program. This is the so-called *bridge factor*. The equation thus changes to:

$$V_{out} / V_{in} = \frac{1}{4} \cdot B_F \cdot k \cdot \varepsilon$$

The bridge factors are given in the table for all bridge configurations.

The measured value from the strain gage is therefore the strain:

$$\varepsilon = dL / L$$

The strain is usually presented in **um/m**, so the ratio of elongation in micrometers comparing to the length of a specimen in meters. So what does that really mean if we measured a value of **2000**? First of all, we can also express this in *percent*. Strain in um/m divided by 10000 is elongation in percent. In the case of 2000, the elongation will be **0.2%**.

We can also judge from this value how close the material is from its limit of elasticity. For steel, this limit is app. 2% (this depends heavily on the type of steel), so were exceed 20000 um/m, the specimen would be over its tolerance of elasticity and would be permanently stretched. The full bridge example will demonstrate how to calculate stress and force from the strain.

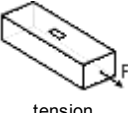
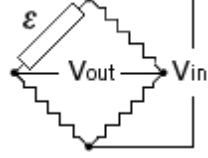
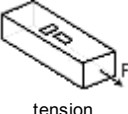
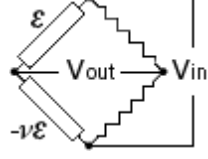
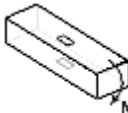
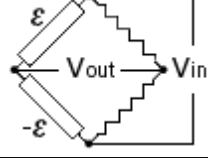
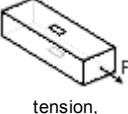
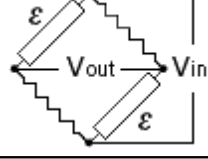
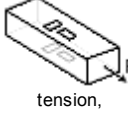
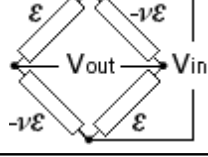
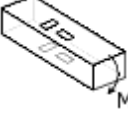
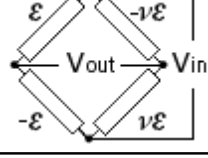
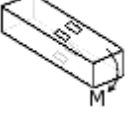
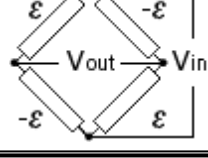
Bridge configurations

There are several configurations for basic measurements. First, we need to know the special effects of the materials - that is when the material is stretched, the material (usually) gets thinner in the other (two) directions. The ratio of *transverse* strain to extension strain is called Poisson's ratio ***ν***.

When strain gages are positioned 90 deg. towards each other, it becomes very important to know the ratio of transverse strain and to include this in the equation.

The Poisson's factor is **0.27** to **0.31** for steel (usually **0.3** is used) and **0.33** for aluminum.

The following table shows several basic configurations for the gages. Basically the configurations are divided in *quarter*, *half* and *full* bridge circuits.

| MEASURES | TYPE | BRIDGE | EQUATION V_{out}/V_{in} | BRIDGE FACTOR | LINEAR | DESCRIPTION |
|---|---------|---|---|---------------------|--------|---|
|  tension, compression | quarter |  | $\frac{k \cdot \epsilon}{4 + 2 \cdot k \cdot \epsilon}$ | 1 | no | Single gage measuring tension and compression - basic configuration |
|  tension, compression | half |  | $\frac{k \cdot \epsilon \cdot (1 + \nu)}{4 + 2 \cdot k \cdot \epsilon \cdot (1 - \nu)}$ | $(1 + \nu)$ | no | One gage in principal direction and one in transverse direction - usually used for temperature compensation |
|  bending | half |  | $\frac{k \cdot \epsilon}{2}$ | 2 | yes | Two gages with opposite strain - usually used for measurement of bending |
|  tension, compression | half |  | $\frac{k \cdot \epsilon}{2 + k \cdot \epsilon}$ | 2 | no | Two gages with same strain - usually used for bending cancellation |
|  tension, compression | full |  | $\frac{k \cdot \epsilon \cdot (1 + \nu)}{2 + k \cdot \epsilon \cdot (1 - \nu)}$ | $2 \cdot (1 + \nu)$ | no | Two pairs of gages where one is in the principal direction and the other one is in transverse direction used in temperature compensation and bending cancellation |
|  bending | full |  | $\frac{k \cdot \epsilon \cdot (1 + \nu)}{2}$ | $2 \cdot (1 + \nu)$ | yes | Two pairs of gages where one is in the principal direction and the other one is in transverse direction used in temperature compensation and tension cancellation |
|  bending, torsion | full |  | $k \cdot \epsilon$ | 4 | yes | Two pairs of gages in opposite strain - usually used for measurement of bending |

Sensor mounting

The mounting of the strain gage is a process where we need to *clean the surface*, *glue* the strain gage, *connect lead wires* and *protect* the strain gage. The detailed process is beyond the scope of this manual; please refer to the web pages or handbooks of strain gage manufacturers for specific information. Since the different configurations must be connected in the correct way, as shown in the above picture, it is very important to use *special care* when connecting the gages to the amplifier.

A little story about the strain gage connections is very fitting in this context. Most people are likely familiar with Murphy's Law. It originally states that "Whatever can go wrong, will go wrong". This is very well known, but what is not as well known is that it originates from the strain gage measurements. The "inventor" of this law, Capt. Ed Murphy, made a strain gage measurement system for a g-force testing system at Edwards Air Force Base, where the maximum g force that the human body could take was to be tested. As a side-note, a real human was used, and the maximum force was 40 g.

The result of the first measurement was zero, simply because the strain gages were connected in such a way that they gages canceled out each other. Capt. Ed Murphy blamed his assistant for the error, who had connected the gages in the wrong way. The other, even more interesting part of the story for the process of measurement was that Murphy simply declined a verification of the system, which was offered to him before performing the test.

The point of this story is this: Connect - CALIBRATE - VERIFY - Measure. If Capt. Ed Murphy had followed this procedure Murphy's Law would not have been invented (at least not on that occasion).

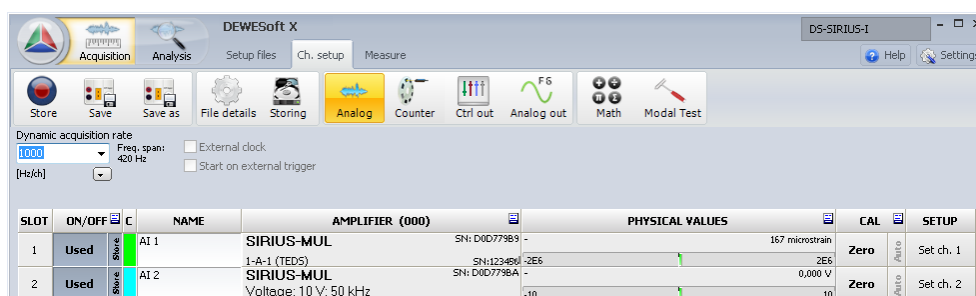
2.5.1 Experiment setup

| | |
|--------------------------|-------------------------|
| <i>Required hardware</i> | Sirius MULTI, STG |
| <i>Required software</i> | SE, PROF or DSA version |
| <i>Setup sample rate</i> | At least 1 kHz |

I have prepared two common types of *strain gage* configuration. First we have a tuning fork with a single gage attached and an off-the shelf load cell with a full bridge connection.



From the amplifier side, we use two 9 pin MULTI modules with isolated input.



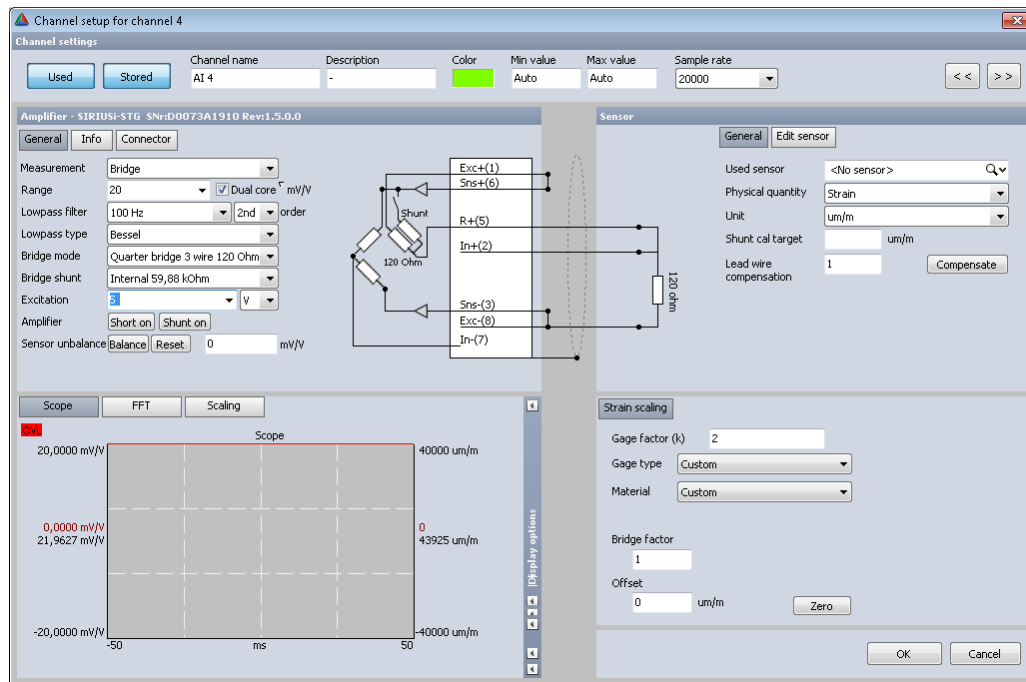
2.5.2 Quarter bridge setup

Let's take a look at a quarter bridge setup in **DEWESoft** using DAQP-BRIDGE-A. A single strain gage is attached to the tuning fork. From the specification, we can see that the resistance of the strain gage is **120 Ohms** and the gage factor **k** is **2,07**. When selecting strain gages, we can typically choose from **120** or **350** Ohms. 120 Ohms gages will have less power consumption and less heating, while 350 Ohms will have large signals and therefore work better with longer cables.



With this information, we can **set** the quarter bridge with **120 Ohms** as the *input type*. This means that the single bridge will be the *real* gage, while other three gages will be *internal precision* resistors. Now, the *input scaling* is in **mV/V** ①.

We can freely select the **Excitation voltage** ②. A higher excitation voltage will increase the signals, and therefore reduce the noise. This will also cause more gage self-heating and will increase the power consumption. All strain gages has a certain limit for excitation voltage, so check this prior to connecting the sensor not to burn it. Higher excitation voltages are therefore *recommended* for *longer* lines. The next step is to enter the **k factor** and **Bridge factor** ③ from the specification. Since this is only a quarter bridge, we can enter the bridge factor as **1**.



The scaling changes from $\mu\text{V/V}$ to $\mu\text{m/m}$. We can now enter the range in units of relative deformation.

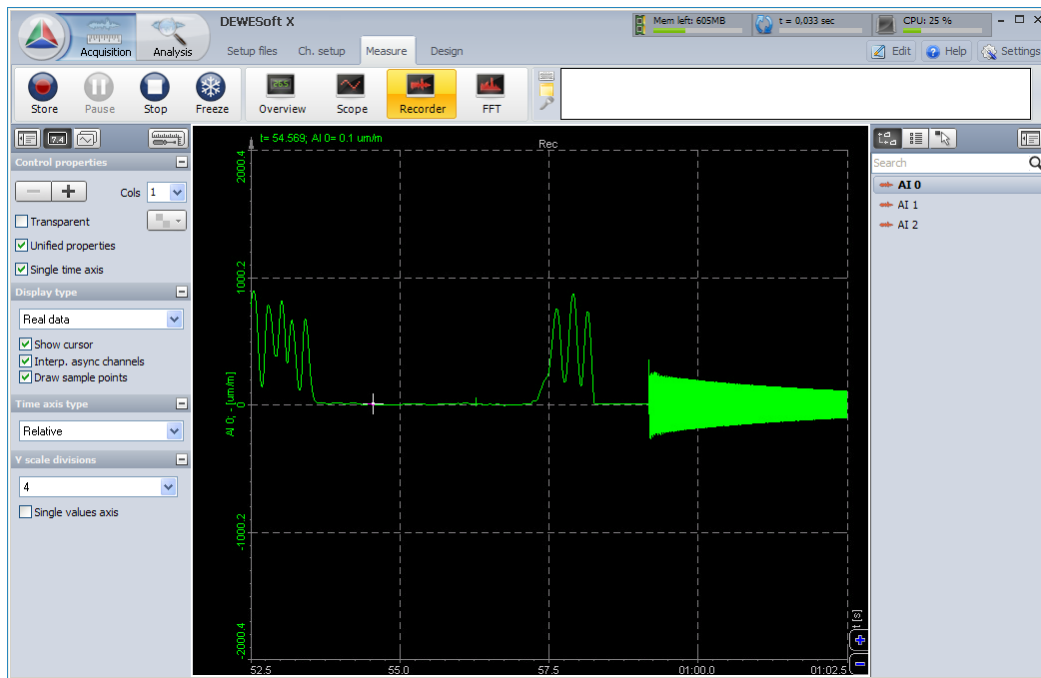
We have the tuning fork connected with *three* wire connection. The three wire connection allows us to cancel out the effect of wire resistance. Especially with long cables, the wire can significantly reduce the sensitivity of the gage. The third wire combined with the shunt calibration allows us to measure this resistance and correct this error.

The *shunt calibration* can be used to measure and correct the resistance in this case. We need to choose the **Bridge cal** scaling and click the **Compensate** button. Shunt calibration takes a while since it has to set several configurations and measure back the results. After that, the wire resistance measurement shows in the drawing, the **Corr(ection). factor** is shown and, if possible, the **Excitation** voltage level is raised to still meet the required sensitivity of the gage.

As the last step, we set the **Lowpass filter** of the amplifier to **5 kHz** to be able to see the *tuning fork* natural frequency, which is **440 Hz** (A4 tone).

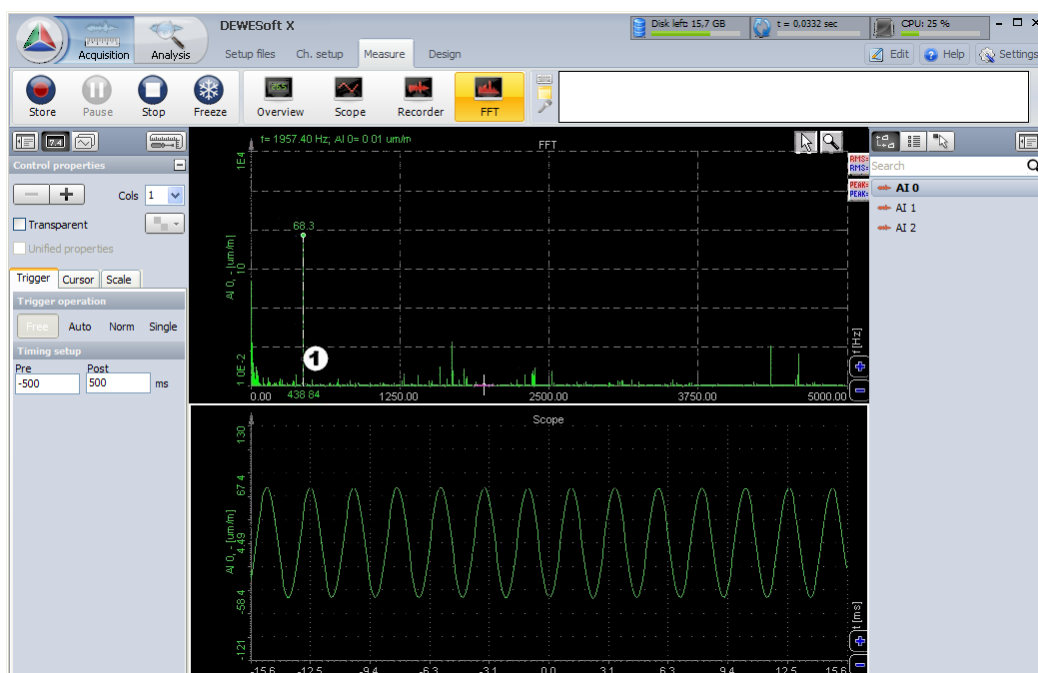
2.5.3 Quarter bridge measurement

Now let's do some more measurements. Let's take a look at *recorder*. If a *static force* is applied on the tuning fork, we can see the *changing offset* of the signal. We might also try *hitting* the tuning fork so it makes a sound to reflect the natural frequency of the tuning fork (the conventional way it is used). We can see this as a *high frequency* vibration with *falling amplitude* because of air friction and friction in the fork.



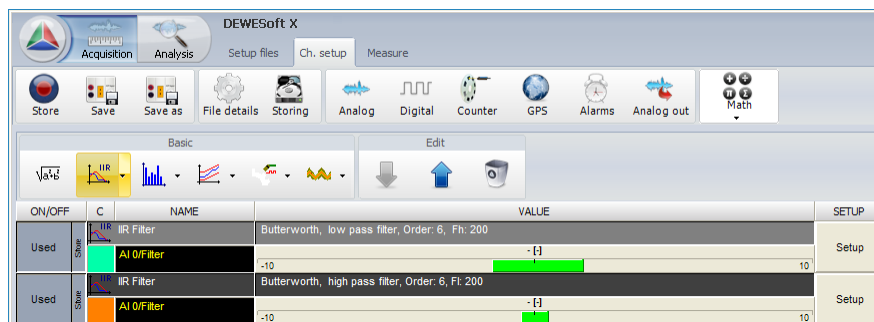
When looking at the FFT screen (let's change it to the *logarithmic scale* that can be used to see all the amplitudes in a nice way, we can see that there is an obvious peak at approximately **440 Hz**. We can also *place* a cursor at this point by simply clicking *on the peak* in the FFT. The frequency shown is **438.8 Hz** ①. It is not exactly 440 because the tuning fork used here is surely isn't what it should be after many years of use, but also because the FFT has a certain line resolution.

This line resolution depends on the *sampling rate* and the *number of lines* chosen for the FFT. If we want to have a faster response on the FFT, we would choose fewer lines, but we would have a lower frequency resolution. If the user wants to see the exact frequency, it is necessary to set a higher line resolution. This is well described in reference guide, but a simple *rule of thumb* is: if it takes **1 second** to *acquire* the data from which the FFT is calculated, the *resulting* FFT will have **1 Hz** line resolution. If we acquire data for **2 seconds**, line resolution will be **0.5 Hz**.

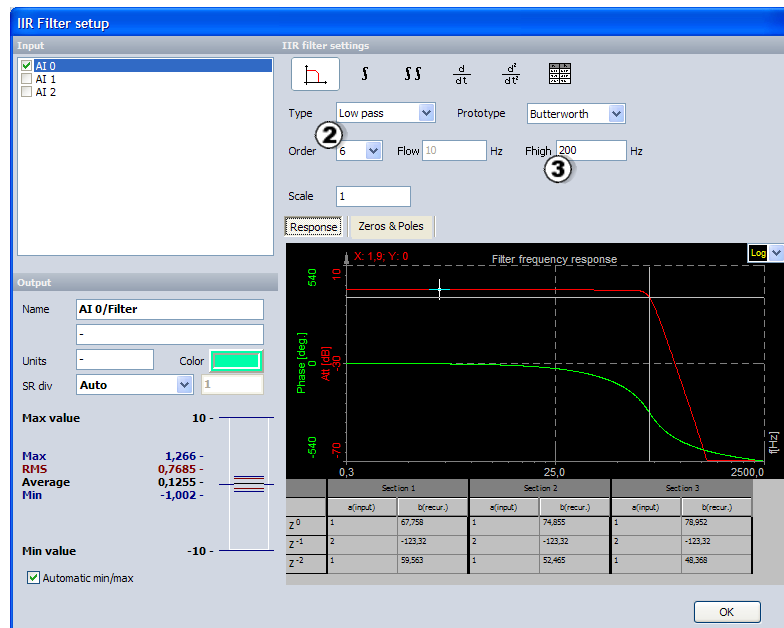


This is also a perfect example to take a look how to *use* the **filters** in **DEWESoft**. Clearly, there is one part of the signal in the form of the *offset* (static load) and one part in a form of *dynamic ringing* with a **440 Hz** frequency.

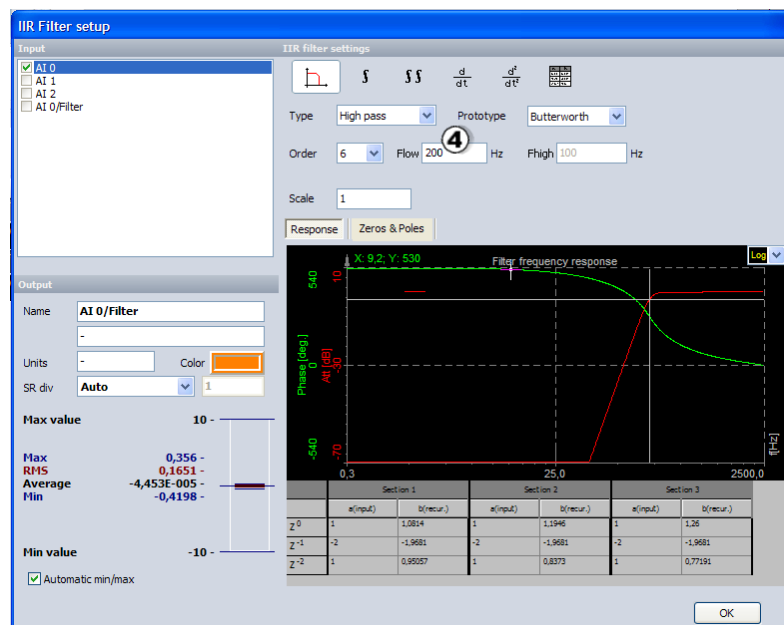
If we want to extract those two components from the original waveform, we need to set *two* filters - one low pass and the other high pass. So we add two filters in the **Math** section.



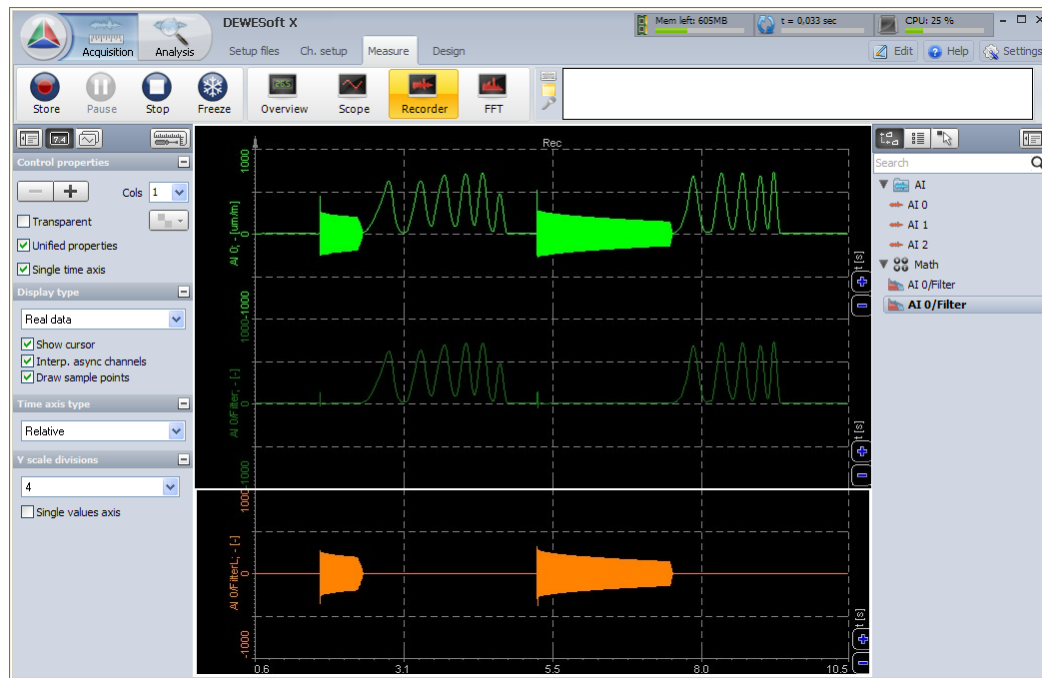
1. First, we set the *input channel* (AI 0) in this case. Then we set it to **Low pass**, 6th ② order filter and we set the *cutoff frequency* **Fhigh** to 200 Hz ③. This is so that all the signals below 200 Hz frequency will pass and all the frequencies above this will cut.



2. The second filter is set to **High pass**, 6th order with the same *cutoff* **Flow** frequency ④.



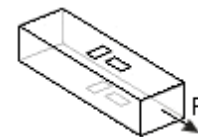
If we show those two filters on the *recorder*, we can see that the signal is nicely *decomposed* to the static load and dynamic ringing. The user can use this technology to *cut off unwanted* parts of the signal or to *extract wanted* frequency components of the certain signal.



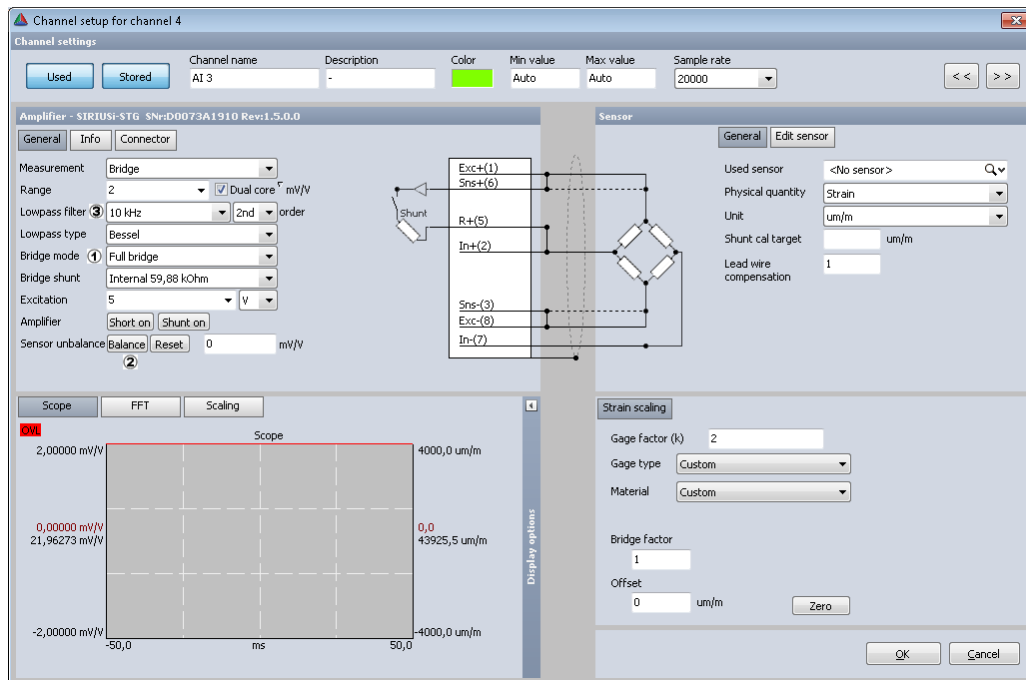
At this point it might be worth noting that **IIR** filters are used where we want *higher* calculation speeds and cutoff rates are needed. We can also use **FIR** filters if we don't want to have any *phase shifts*. More details can be found in the Filter comparison section of FIR filter in the user's manual.

2.5.4 Full bridge setup

For full bridge sensor we will use a "home made" **force sensor**. The sensor is built from two xy strain gages, where one gage is oriented in the *principal* direction and another one in *transverse* direction. This sensor is therefore *temperature compensated* and has *bending cancellation*.



Let's do a step by step **calibration** of such a configuration. First of all we change the **Input type** to **Full bridge** ① and **Balance sensor** ② is performed. If the bridge zero is not successful, we should choose the highest possible **Range**, perform zero, then switch to the wanted range and perform zero again. Then we set the **Lowpass filter** ③. Since this example will measure more or less static measurements in a low region of the probe, the results can be enhanced by using a *very low* lowpass filter. In this case it is set to **10 Hz** ③.



Next we set the **Strain scaling** to used and set the **k factor** of the gages to **2** and the **Bridge factor** to **2.6**. The value of **2** is read from the strain gages *data sheet*. The value of **2.6** is a *bridge factor* from the table of bridge configurations. Now the input values are strain in **um/m**.

Now we need to do the **final Scaling by function** since the force should be measured in **N**. Now it's time to do some calculations using following equations:

$$\text{Stress} = \text{Force} / \text{Area} \quad \text{and}$$

$$\text{Stress} = E \cdot \text{Strain}$$

Therefore the **Force = Area · E · Strain**. Given that the elastic modulus (Young's modulus) of steel is **210000 N/mm²** and the cross section of the sensor is **139 mm²** we get:

$$\text{Force} = 139 \text{ mm}^2 \cdot 210000 \text{ N/mm}^2 \cdot \text{strain} / 1\text{E}6 = 28 \cdot \text{strain}$$

Factor **1E6** is there because the strain is measured in **um/m** and we need to have a *scaling factor* for *this* unit.

Now we have the real data scaled in **N**. The last thing to do is to select the measurement **Range** based on the *input*. In this case the *lowest range* will be more than enough for the measurement.

The **Short** and **Shunt** button is for *checking* the strain gage. The **Short** is used to *short the pins* on the *input* and measure the bridge *offset*. The **Shunt** (which is also used in the *shunt calibration routine*) can be used to see if the bridge is *reacting* - so to check if the connections are working.

2.5.5 Full bridge measurement

Since writing tutorials can be quite boring at times, the DEWESoft team decided to make things more interesting for this example.

This *load cell* was a perfect chance. We added one *camera* to see the measurement live and there was an in-house competition trying to break apart the load cell to see who the strongest employee was. Since the *measurement range* of load cell is 30 kN or approximately 3 tons, no one succeeded, of course.

But since we are **measuring** the *force* on the cell, the results could be easily judged.

It turns out that the strongest man (as was expected) was the CAMCNC engineer, pulling 570 N ① or 57 kilograms.



Of course the biggest question of all was: how did the management perform? As one can see from the picture, there was lots of effort and many discussions, but not really a big effect...

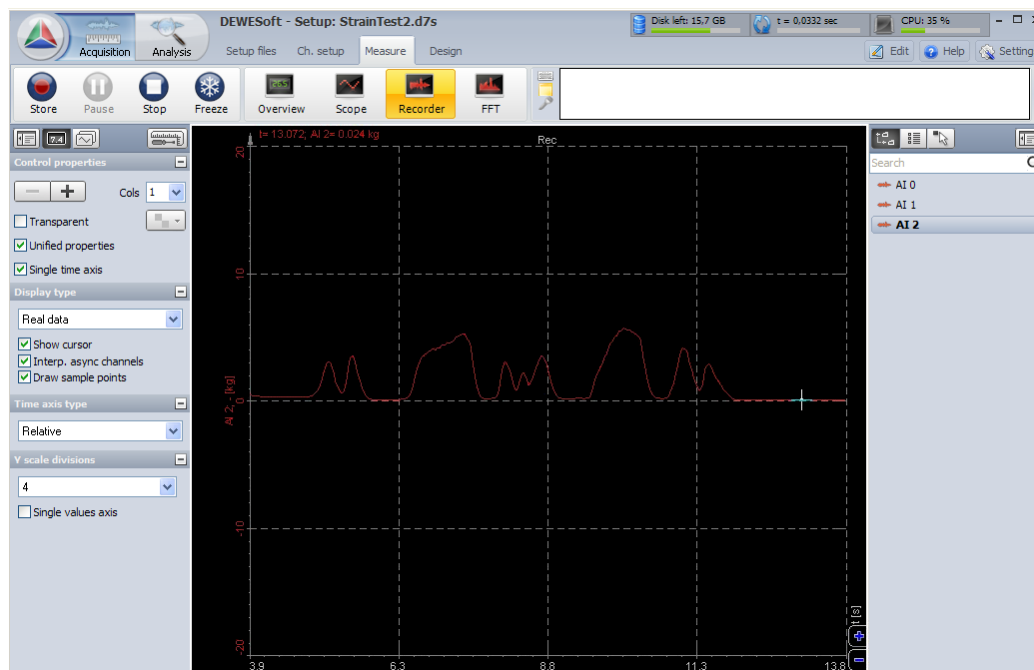


2.5.6 Load cell setup

The load cell **setup** is the easiest of all. The load cell used here was **Full bridge**, so we set the correct **Input type** and **Excitation** voltages. Then a **Bridge zero** was performed with *no* attached load.

The next step is to enter the sensor's **Scale** factor. The load cell is calibrated in **kilograms** and we can see from the spec sheet that it outputs **1 mV/V per 10 kg**. We enter this value in the scale factor and this concludes the setup.

There is not really much to see from this load cell in the measurement. An in-house competition "who weighs the most" could be held, but it wouldn't make much sense - surely management would win.



2.6 Counter

Counter channels are used to perform **counting** and **frequency measurements**. Typical applications are:

| | | |
|-----------------|--|--|
| Counting modes: | <ul style="list-style-type: none"> • Event counting | <ul style="list-style-type: none"> - Simple - Gated event - Up/down |
| | <ul style="list-style-type: none"> • Encoder counting | <ul style="list-style-type: none"> - X1, X2, X4 modes - Zero pulse |
| Timing modes: | <ul style="list-style-type: none"> • Period and pulse-width | <ul style="list-style-type: none"> - Period measurement - Pulse-width measurement - Duty cycle |
| | <ul style="list-style-type: none"> • Two pulse edge separation | |
| | <ul style="list-style-type: none"> • Frequency/super-counter | |
| Sensor modes | | |

Different cards offer different counting possibilities. By far the most enhanced counters are those on Orion 1624 or Orion 1616. A special mode, called super-counter, allows *exact frequency* measurement and *direct support* of the most commonly used *sensors*. All other counters offer only a specific subset of operation.

Basically the input to the counter should be a clean *digital signal*, where most of the cards support 5 V levels as the default (some cards offers higher input signals or even variable trigger levels). If the user has lower or higher signals, some *conditioning in front* of the counters should be used.

One of the practical applications where the counters are often used will be more closely examined:

- **Counters in automotive applications**

2.6.1 Event counting

Event counting is one of the simplest counter operations.

| | |
|--------------------------|-----------------------------|
| <i>Required hardware</i> | DEWE-43, Sirius ACC+, MULTI |
| <i>Required software</i> | Any version |
| <i>Setup sample rate</i> | At least 1 kHz |

There are two special modes of event counting available:

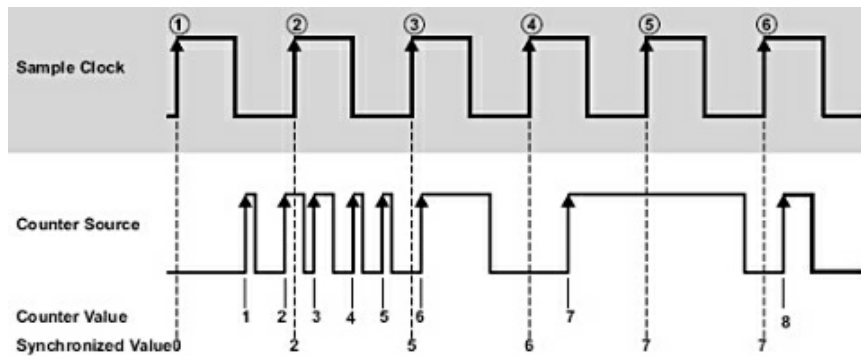
1. gated event counting, where events are counted only when a gate signal is *high*
2. up/down counting, where the events are counted *up* when the *gate is high* and *down* when the *gate is low*.

The following simple test configuration was made with *push button*, connected to the DEWE-43 counter.

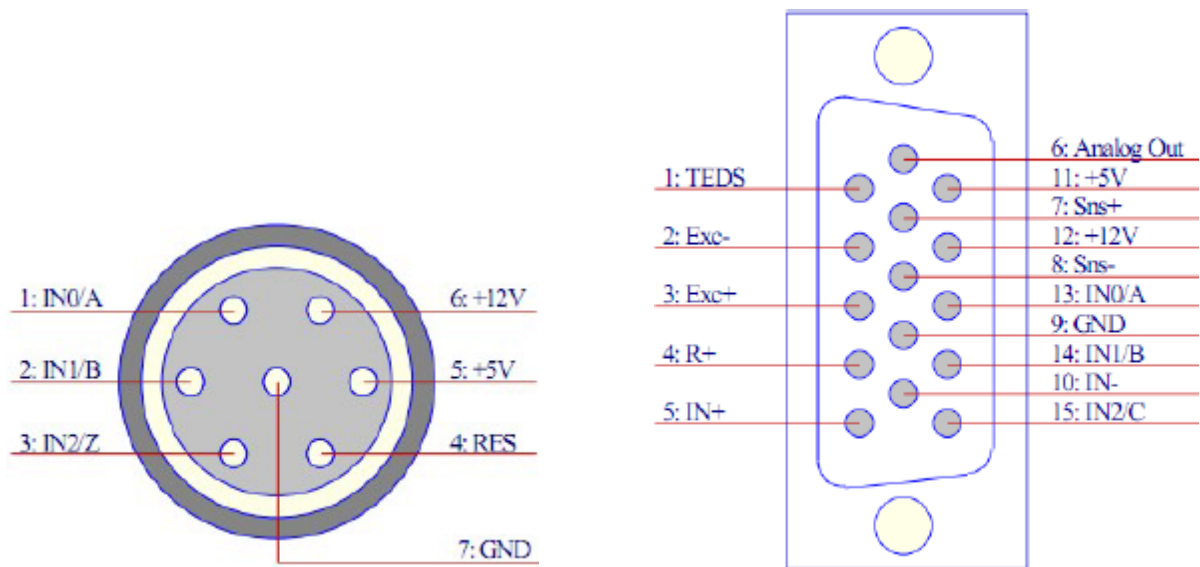


2.6.1.1 Simple event counting

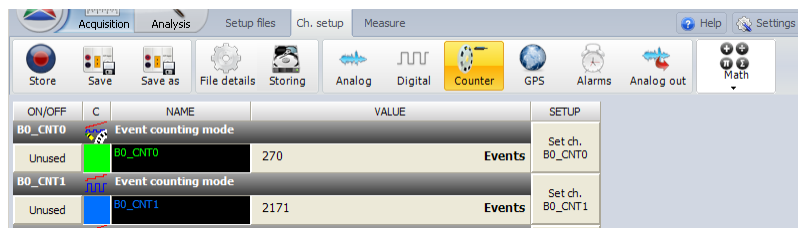
Simple event counting is the mode where we can **count** either *falling* or *rising* edges of the *signal*.



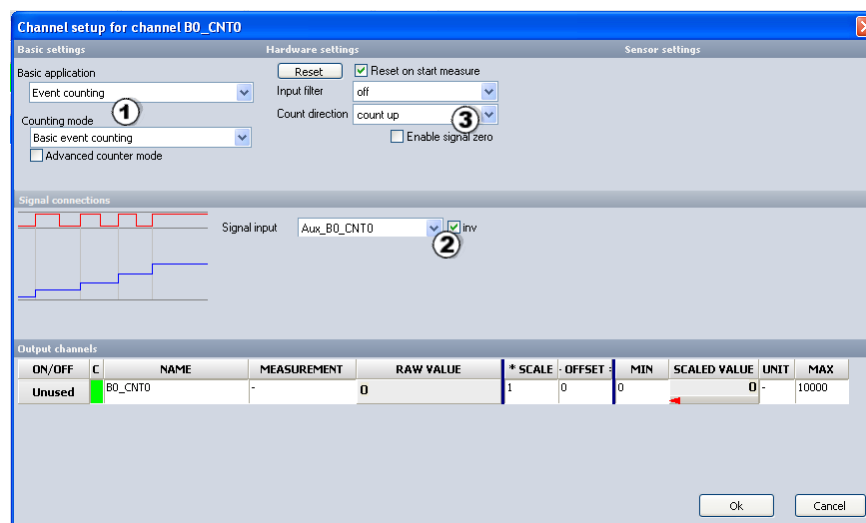
In this case we just add a *switch* between the counter input and the ground. Since all inputs have pull-ups, shorting the switch will give a *zero* and when the input is opened, it will be *one*.



Then we go to the **Counter** tab and select **CNT0**.



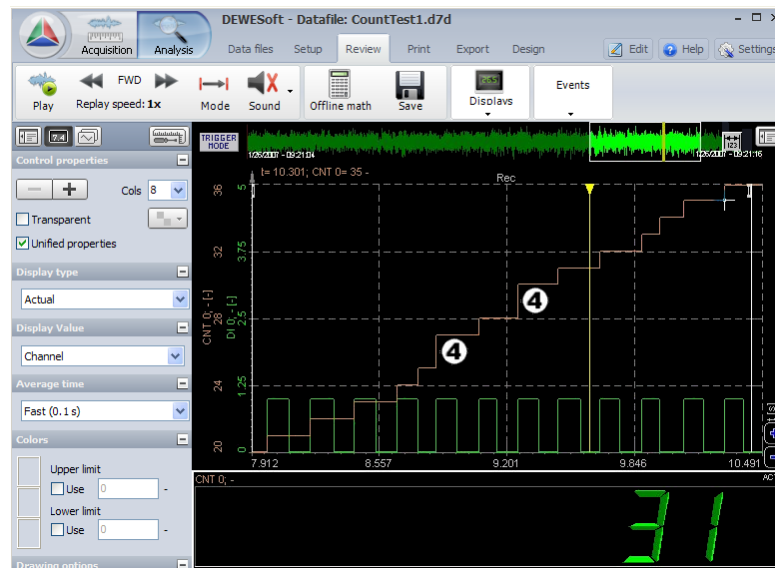
To **set up** the counter click "**Set ch.**" button. The **Basic application - Event counting** and **Counting mode - Basic event counting** ① are already selected by default. Then we choose the **Signal input**. Usually the signal input is **Source0**, but since the signal is connected to **Aux_B0_CNT0**, that signal ② needs to be selected as the input. The **normal** state (when the switch is not selected) is high, therefore it is nice to **invert** the signal by choosing the **inv** check box ②. This has two effects: firstly, the levels will **change**, so if the button is not pressed, the level will be low and consequentially the counter will count on **falling edges**. We can select the **Count direction** either to **count up** or **count down** ③.



The **filter** is also an important setting to prevent **double counts**. We need to set the filter to react a bit faster than what we expect the events are expected to be or, with a different logic; we need to set it a bit slower than expected frequency of

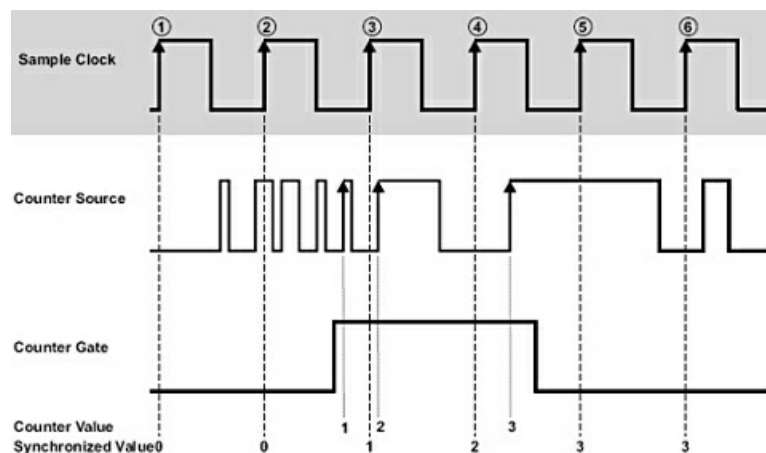
the glitches. With a manual switch 80 MHz base clock, some glitches we can for sure be expected. Let's look at the measurement.

The green curve shows the digital signal from the switch and the orange curve shows the counter value, while the digital meter shows numerical value of the switch. The counter value is *increased* by each transition from low to high (in this case, since the values are inverted, it is a real voltage transition from high to low). We can see at some points that the values are counted up *twice* ④. This is because a counter can see *every glitch* (even below 20 nanoseconds) in the signal. Therefore we need to use the a filter to *filter out* these glitches. For example we could use a highest possible filter of 5 microseconds, since the speed of switching is really low.



2.6.1.2 Gated event counting

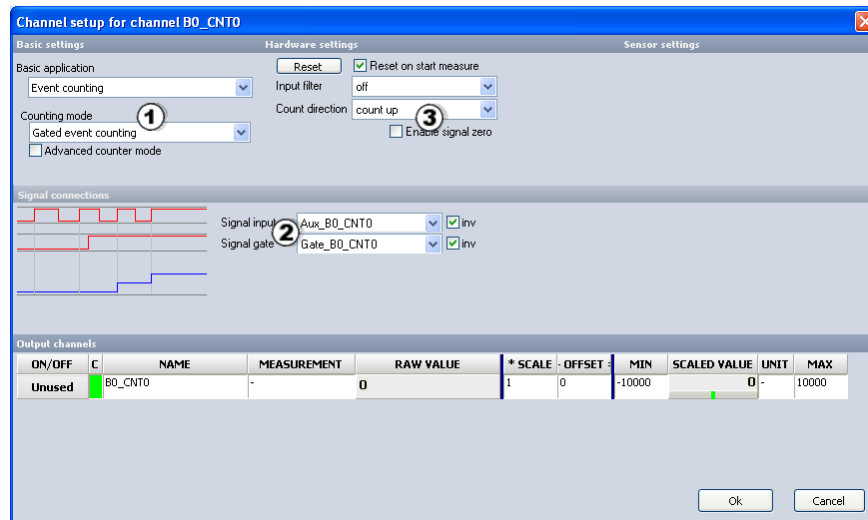
Gated event counting is a mode where the counter **counts** only when the *gate signal* is *high*. This tutorial is a continuation of the previous "*Simple event counting*" section.



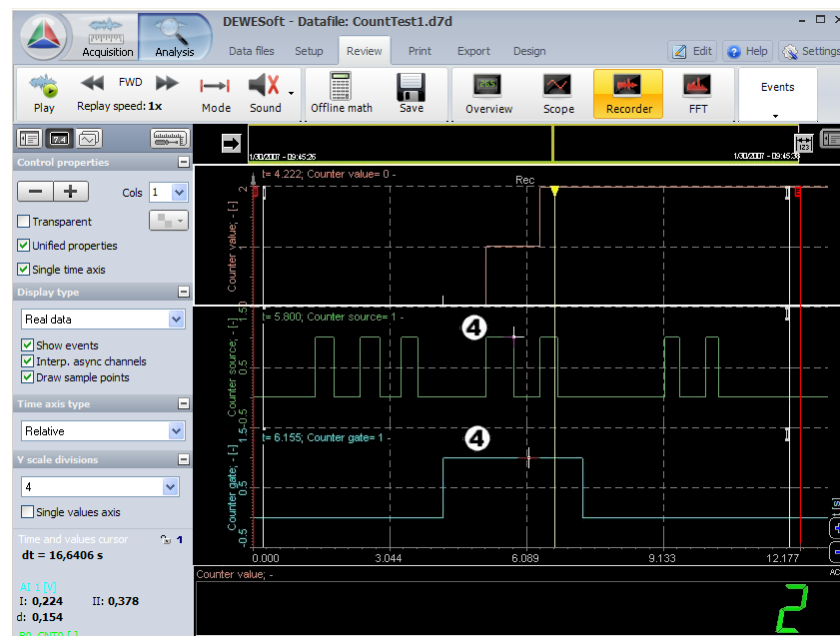
The counter signal is the same as with simple event counting, but we need to connect *additionally a second* signal - the *gate*. We should connect the *second switch* to GATE 0 as well as the DI_1 to see the values (of course it is not

necessary to do this when measuring real signal, this it is just for demonstration purposes).

Then we perform the **setup** of the *counter channel*. We should choose **Gated event counting** mode ① and set the *signal source*, which is **Aux_B0_CNT0**, and the **Signal gate** (**Gate_B0_CNT0**) ②. The counter will count the transitions from *low to high* only when the *signal gate* is *high*. Since the Signal gate is *inverted* ③ (normally it is high), it is necessary to also invert the *gate* signal so it will count only when a *button will be pressed*.

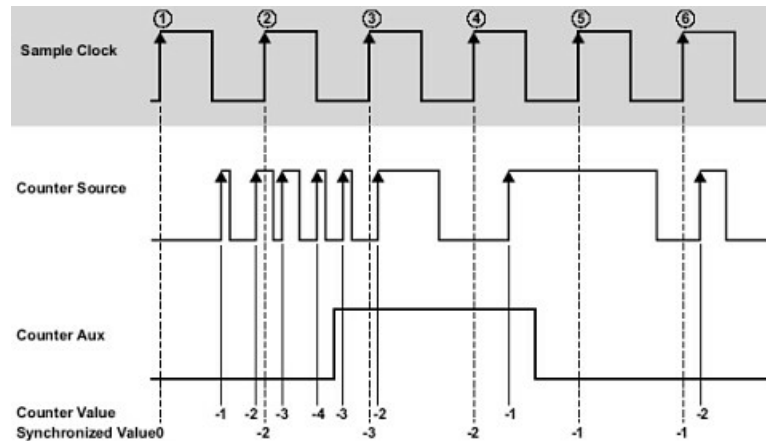


In the example below, we can see how this counter works. The **orange** signal (counter) counts up when the **green** signal makes a transition from *low to high* and when the gate signal (**blue**) is *high* ④.

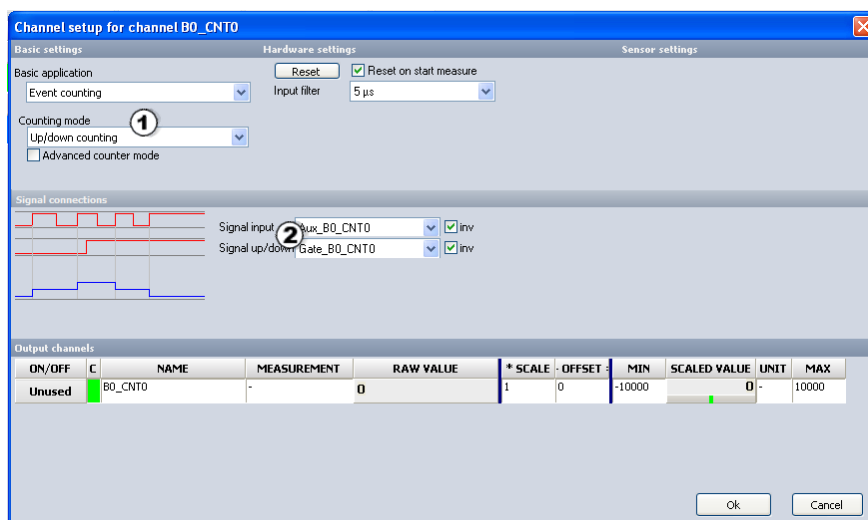


2.6.1.3 Up/down counting

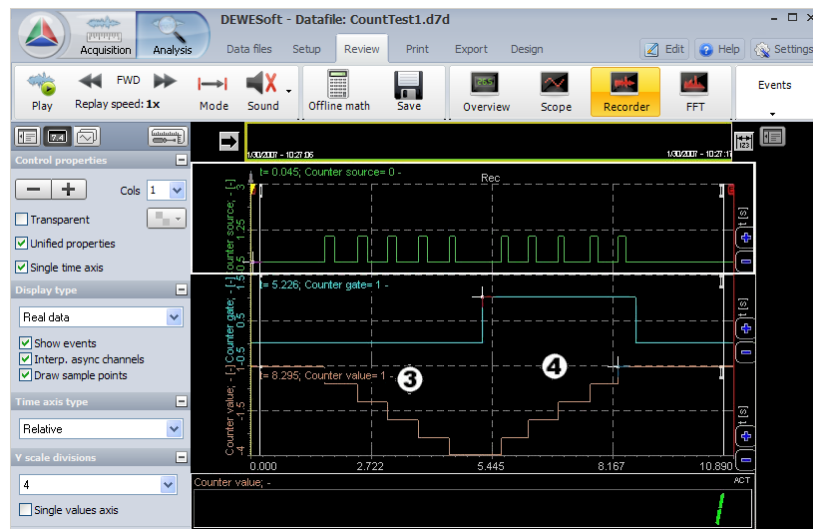
Up/down counting is a counter operation which **counts up** when the *gate* is *high* and counts **down** when a gate is *low*. The encoder operation is similar to this one, in fact we can use this mode to make **X1 encoder** measurements, but some more electronics are needed.



The connection is the same as in the previous example - *gated event counting*. We should choose **Up/Down counting** ① in the **Channel setup** and select the **Signal input** and the **Signal up/down** signal ②.



On the measurement, we can see that when the *gate* (blue signal) is *down*, the counter counts in the *negative* direction ③. When the value of the gate is *high*, the counter counts in the *positive* direction ④.

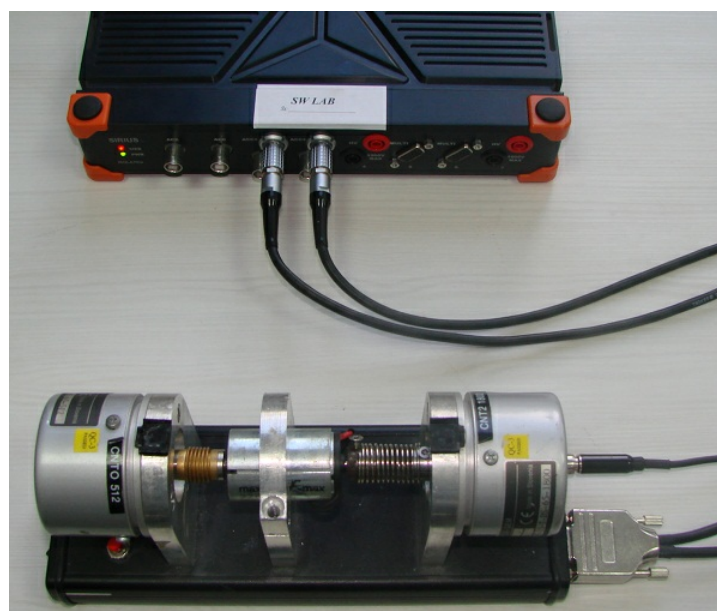


2.6.2 Encoder

An **encoder** is a *wheel* (or *linear bar*) with marks on it. Usually there are encoders having *two marks* with a *phase difference* of *90 deg* to one another - this is to determine the direction of movement, and a *zero pulse* - one pulse per revolution which can tell us the absolute position of the encoder.



The two *signals* (A and B) help to determine the direction of the rotation or movement. When input A *leads* input B, the value is *incremented* and when the input B *leads* input A, the value is *decremented*.

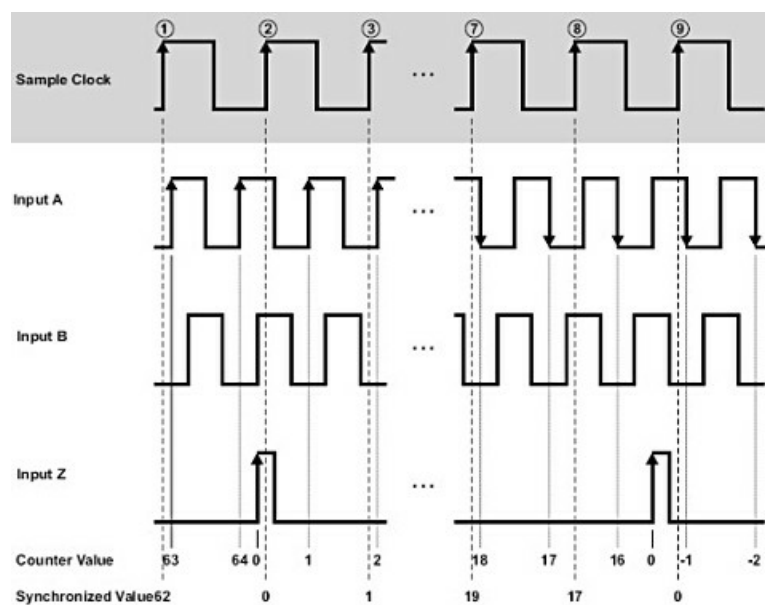


There are several encoder modes: The first one is used to measure only the *rising edges* of input A - this is the so called **X1** mode. The second mode - **X2** measures the *rising* and falling *edges* of input A while **X4** mode measures *rising* and

falling edges of *both* signals. X2 and X4 modes are extremely helpful if there is *slow* movement (for example with linear encoders), because it will actually *increase* the resolution of measurement by a factor of two or four.

| | |
|-------------------|------------------------------------|
| Required hardware | Sirius ACC+, MULTI 15 pin, DEWE-43 |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

If there is a fast *dynamic* measurement (like *torsional vibration*) it will sometimes introduce more errors if X2 and X4 mode is used. This is because those two modes assume that the *gap ratio* is exactly 0.5 and that the encoder *electronics* switches with exactly the *same speed* between *dark* and *light* areas. We can evaluate this error with *period* and *pulsewidth* measurements as described in the next section.

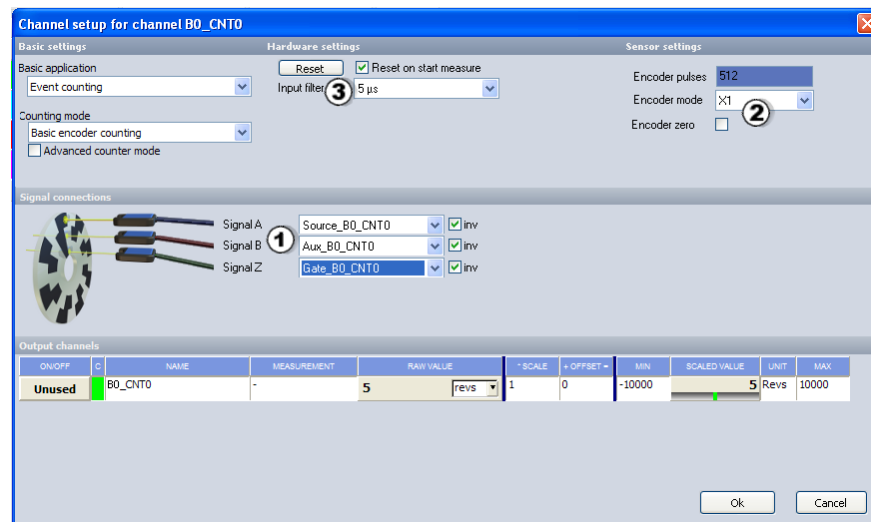


We connect the **A** signal to **SOURCE0**, the **B** signal of the encoder to **AUX_CNT0** and the **zero** pulse to **GATE0**.

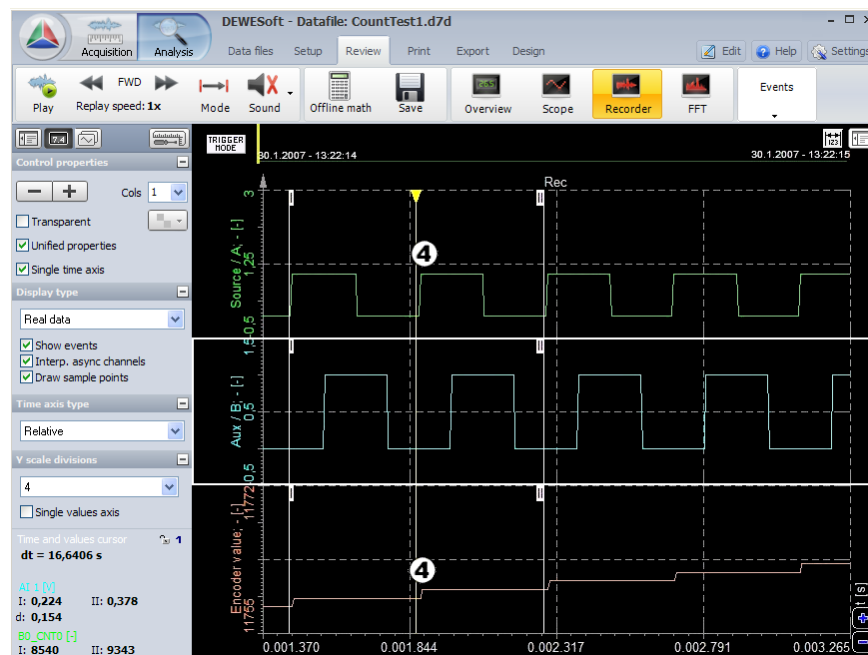
Additionally, just for *visual presentation*, we can also connect those signals to **DI_0**, **DI_1** and **DI_2**, but there is no need to do this for the measurement.

2.6.2.1 X1, X2, X4 modes

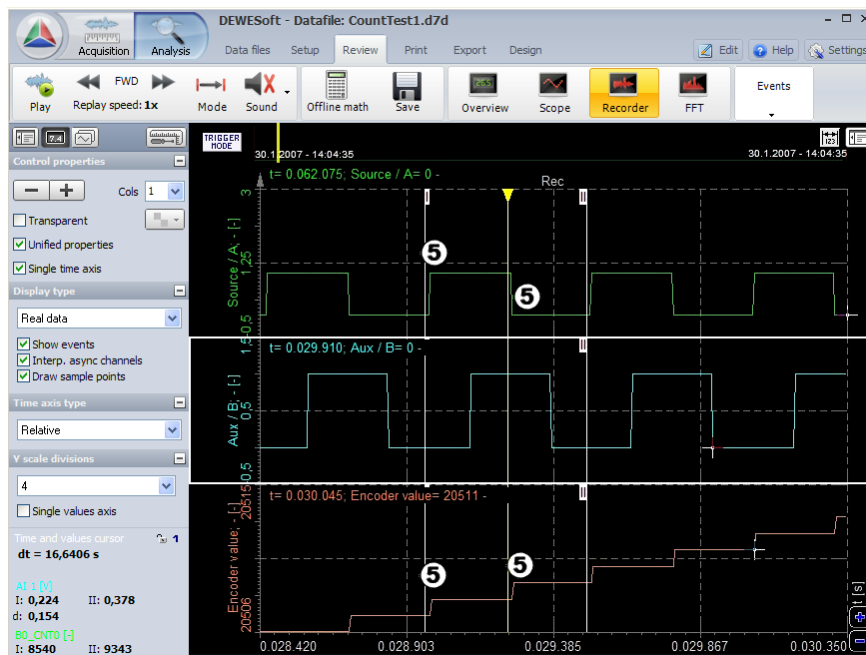
The simple encoder mode can be selected by choosing the **Event counting + Basic encoder counting**. First let's **set up** the encoder. **Signal A** is **Source_B0_CNT0**, **Signal B** is **Aux_CNT0** and **Signal Z** is **Gate_CNT0** ①. We set the **Encoder mode** to **X1** ② and the **Input filter** we set to match our *highest frequency* ③. Scaling is easy - simply select counts, revs or degrees from the drop down or enter any other scale factor.



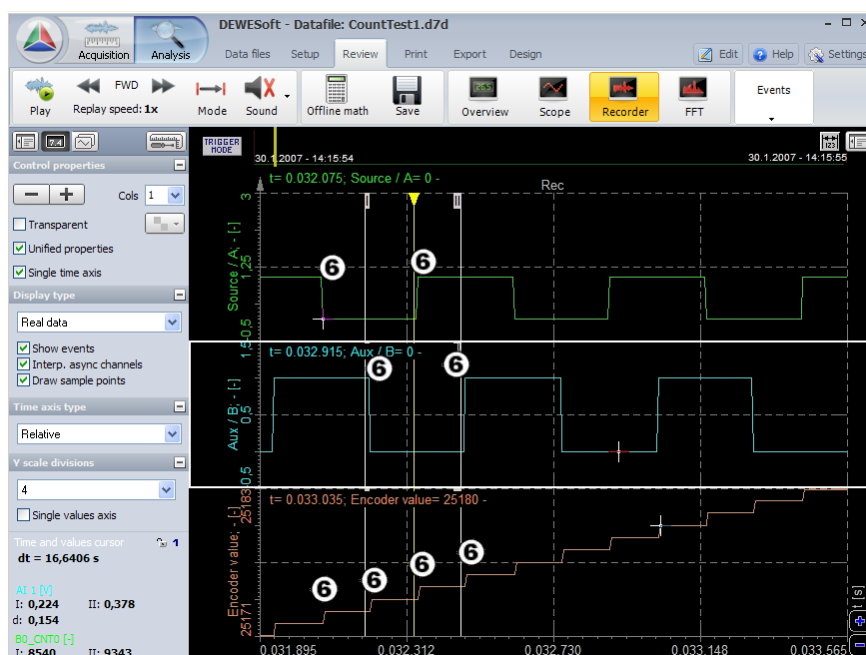
Now let's take some **measurements**. The *output* of this counter is the one that which counts *up* when *signal A leads* *signal B*, and counts *down* when *signal B leads* *signal A*. The *positive edges* of *signal A* is used to *calculate* the counts ④.



If we choose **X2** mode in the setup, the counter will count *raising* and *falling edges* of *source A* ⑤, therefore the resolution will be *increased* by a factor of 2. Everything else stays the same.



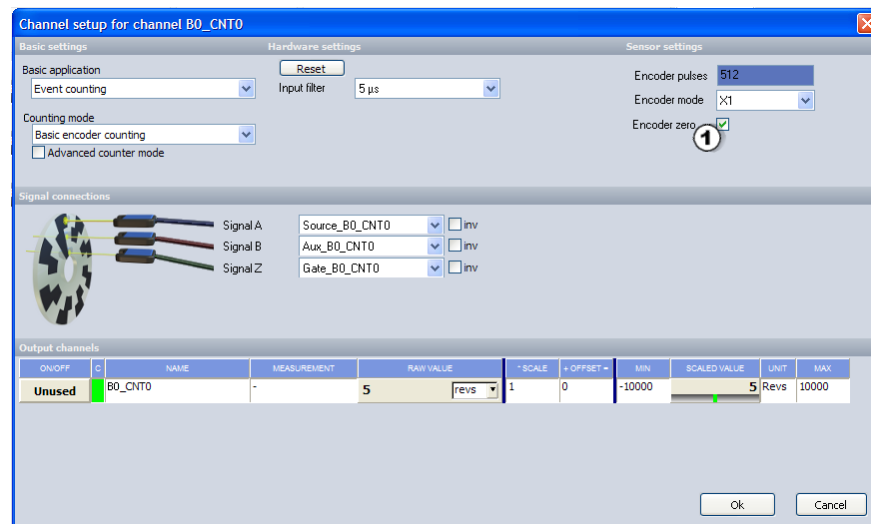
X4 mode counts the *rising* and *falling* edges of **signal A** as well as **signal B** ⑤. The resolution of the measurements is therefore *increased* by a factor of 4.



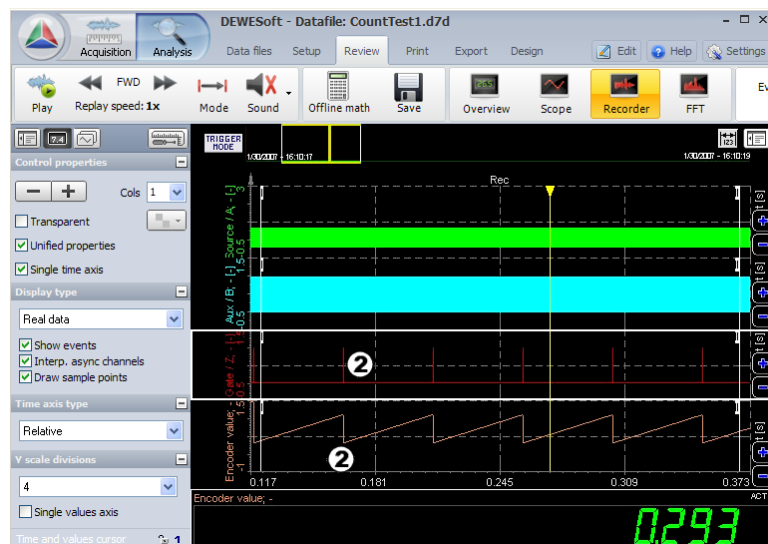
Please refer to the section "[Counter in automotive applications](#)" to see other ways to choose the counter sensors.

2.6.2.2 Encoder with zero pulse

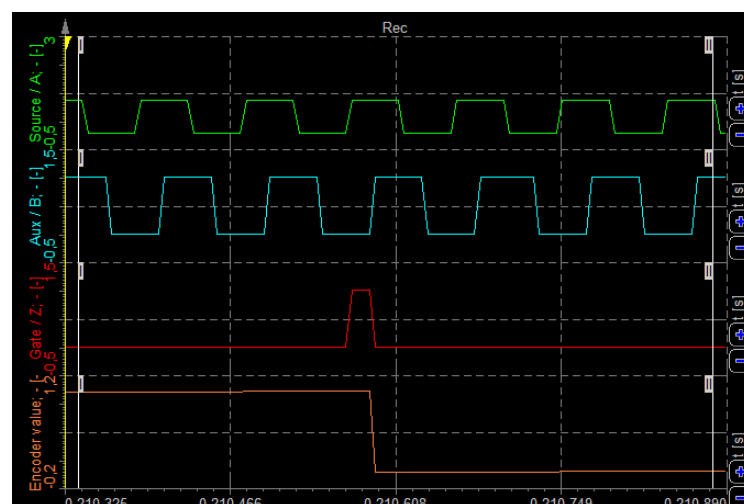
The zero pulse is used to **reset a measurement** when a **Z pulse** is *recognized*. The only change to the setup is to check the **Encoder zero** check box ①. This will *reset* the counter value to 0 when a *zero pulse* is *passed*. We also need to set the number of **Encoder pulses** for internal calculations (512 in this case).



The picture below shows the operation. The **red** curve is the *zero signal*, and the **orange** curve is *encoder output* the . When a pulse is *detected* on the *zero pulse input*, the *counter value resets* to 0 ②.



The picture below is of a zoomed region-in of the *recorder*. It shows that the *encoder* resets the value on the *zero pulse* and *continues to count up* on the *rising edges* of the *A signal*.



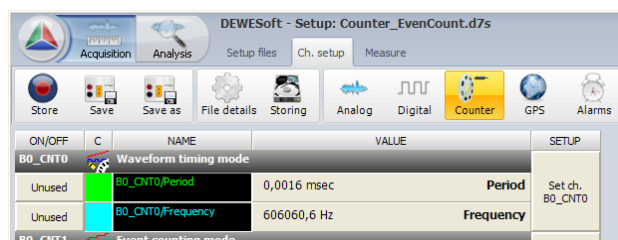
2.6.3 Period and pulse-width

Period and pulse-width measurements are similar in function. The **period measurement** measures the *time* between two *consecutive low to high transitions*, while the **pulse-width measurement** measures the *time* that the *signal* is high.

| | |
|-------------------|------------------------------------|
| Required hardware | Sirius ACC+, MULTI 15 pin, DEWE-43 |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

We will use the same configuration with *two buttons* for this measurement. The hardware connection is simple - we only connect one *switch* to the **AUX-CNT0**, and the same one for the *monitor* to the **DI_0** line.

Then on the **Channel setup** screen we should select one *counter* and go in the "**channel setup**" for *that* counter.

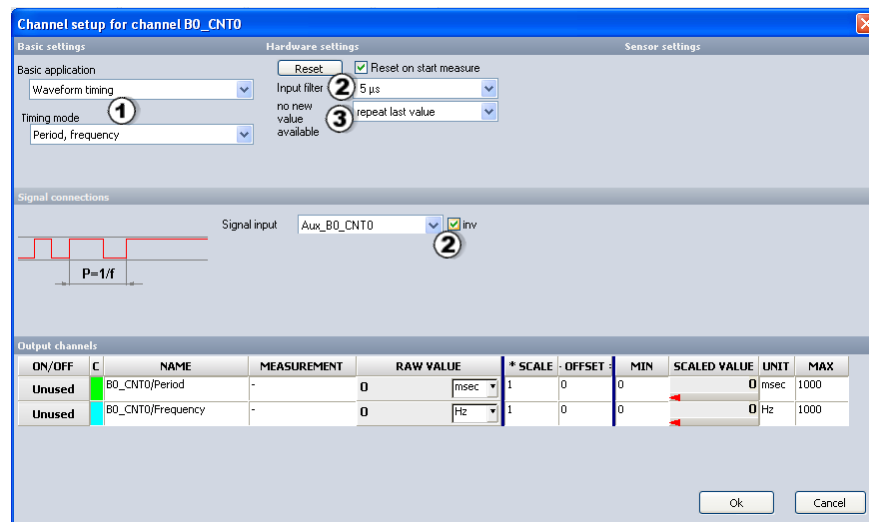


We can also use the period and pulse-width measurement *combined* to do *duty cycle measurement*.

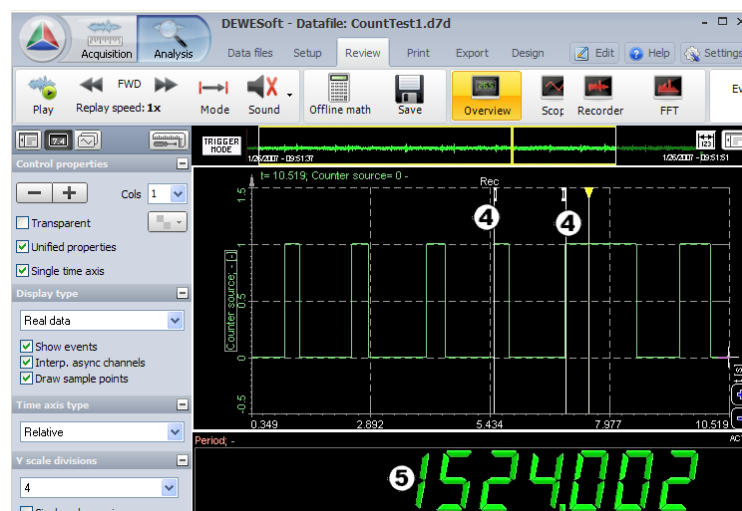
2.6.3.1 Period measurement

First we select the **Basic application** as **Waveform timing** and **Timing mode** as **Period, frequency** ①. We set the **Signal input** to **Aux_CNT0** and **inv(ert)** the signal, so it is *normally low*; we can also set the signal **Input filter** to *prevent glitches* ②.

Then there is one additional field to tell the software what to do when *no new value* is available. The new value is calculated *only* when a *signal changes* the *value* from *low* to *high*. Therefore the value can't be calculated most of the time. If we choose to **repeat the last value**, then the *same* value will be added *until a new transition* is made. Alternately, we can select to **output zero value** when *no new value* is available, so we will have *only spikes* at the points of *new data* and the rest of data the value will be zero ③.

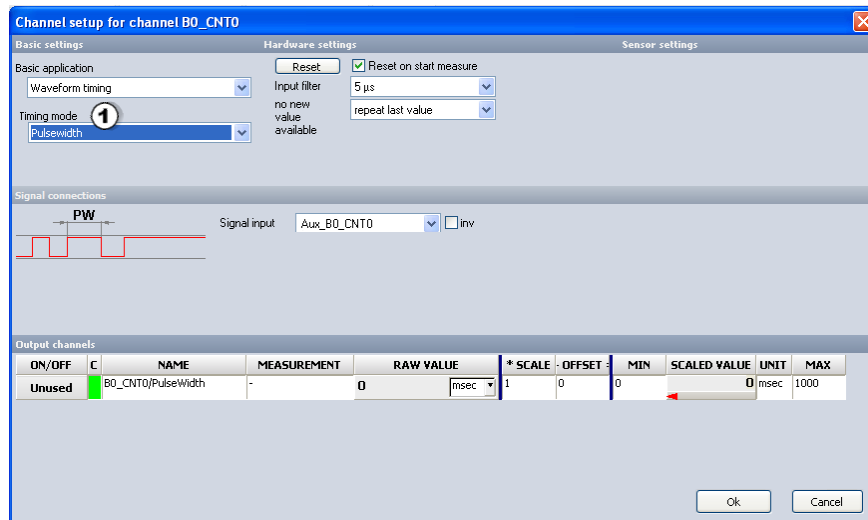


The example below shows how the **measurement** is performed. The two **white** cursors in the *recorder* show the **time difference** of the two *pulses* (it is shown on the left on the recorder setup screen ④). It reads **1.529 seconds**. The *counter value* is of course much *more exact*, as it shows **1524.002** ⑤. Since the *counters* are running with a **80MHz clock**, we have below a **microsecond** resolution. Out of the period, the counter also calculates a frequency, which is simple $1/\text{period}$. We can set the units of period values and frequency in the channel list on the setup screen.



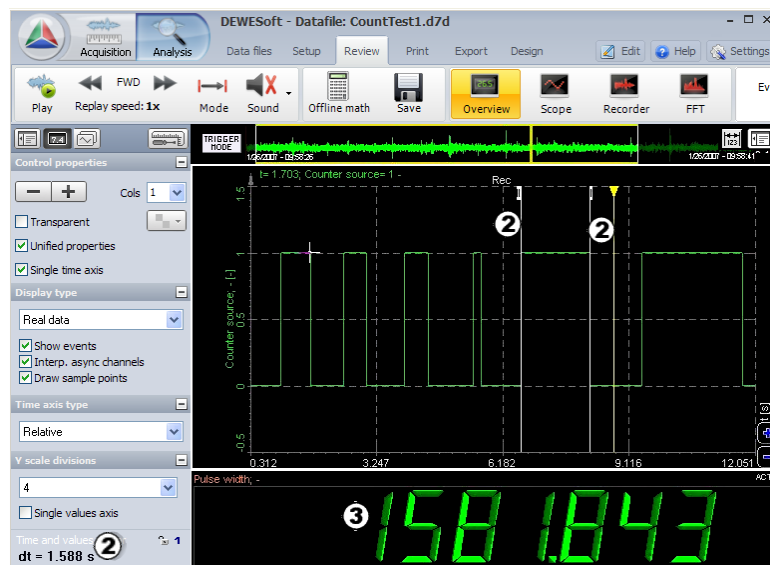
2.6.3.2 Pulse-width measurement

The pulse-width measurement setup is the same as what was previously described in the period measurement sample. The only difference is to select the **Pulsewidth** option in the **Timing mode** section ①.



The **readout** is now **updated** on each **high-to-low transition**. The **recorder** cursor readout shows **1.588 seconds** ② and the **counter value** shows **1581.843 milliseconds** ③.

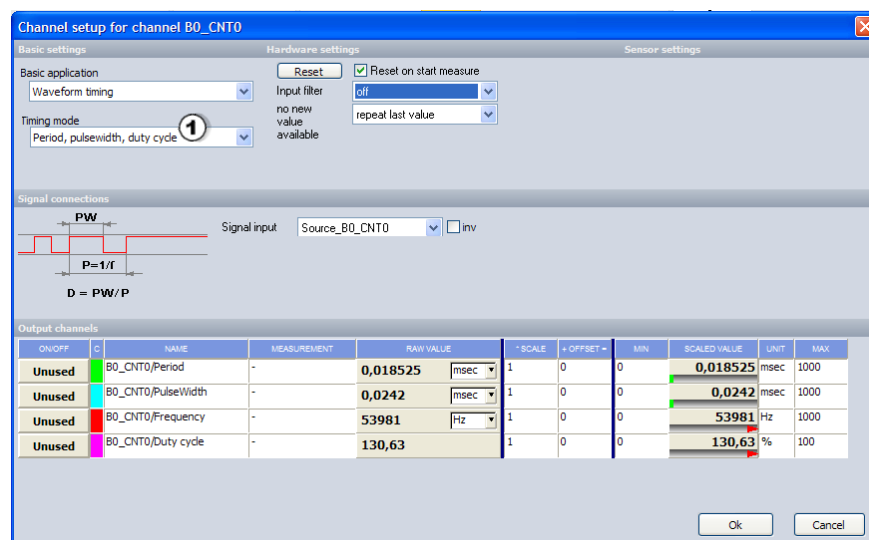
We can use the **super-counter** to measure the **low** and **high** time of each counter or to **combine** the period and pulsewidth to measure the **duty cycle** of the **signal**.



2.6.3.3 Duty cycle measurement

The duty cycle measurement is a procedure where the ratio between the **high** (or **low**) *pulse* of the *signal* and the *period* is **measured**. As was previously stated in the encoder tutorial, the decision to use X1, X2 or X4 mode depends on the quality of the encoder and the encoder electronics. The same encoder as was used for the **Encoder** tutorial will be used to measure its *quality*.

For this measurement we need to set the **Period, pulsewidth, duty cycle** mode ①. If this mode is not available, then the counter does not support it. Either it is a super-counter pair (only even counters of the Orion have this mode) or it is not an Orion card.



Then we can directly select the **Period, pulsewidth, duty cycle** and also the *frequency* output *channels*. The counters are set *automatically* for this operation.

Now let's look at the duty cycle measurements. The upper graph shows the *period* and *pulse-width* of the *signals*. In the lower graph, we can see the *duty cycle* for the few rotations of the *encoder*. We can observe nicely that there are some points where the encoder has a slightly larger error than in the rest of the data. The value there is approximately **0.49**, so this means that the encoder mode X2 will have around a **1%** of the *error*.



2.6.4 Frequency / super-counter

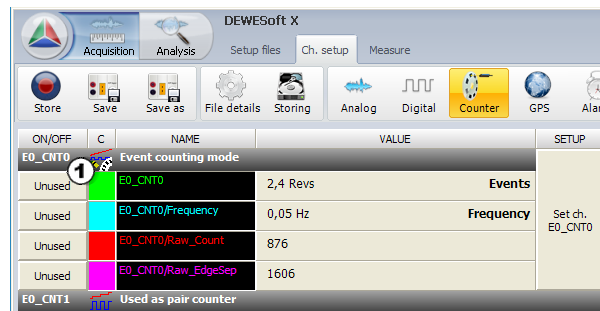
| | |
|-------------------|------------------------------------|
| Required hardware | Sirius ACC+, MULTI 15 pin, DEWE-43 |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

Finally let's talk about the mode, as it has many advantages over traditional counter measurements.

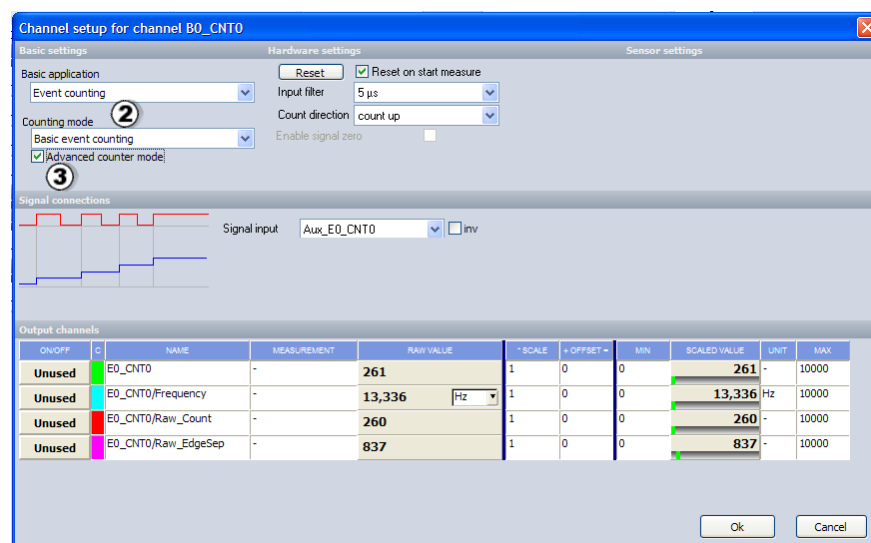
The problem with traditional counters is that the value of the counter is *latched only* at a sample rate interval. Therefore we only have discrete values on each sample. Since the second counter of Orion can measure **exactly** where the position of the *pulse* is between two *samples*, we can **calculate** two things out of this: the *exact interpolated position* of the *counter* at the sample point, as well as the *exact frequency* of the *pulses*.

So I suggest just to a look at how it works to get a better impression about this mode. The *hardware* configuration is as follows: we connect a *signal* (this could be from an *encoder*, as in example below) to SOURCE0. This is have also connected to the COUNTER4 input, just to be able to compare the results to the normal counter.

Now let's **set up** the *channels*. For the super-counter and the frequency measurement, we need to use *two* counters. Another limitation is that the *counter channel* needs to be an *even* counter (CNT0, CNT2 or CNT4) This is marked with the icon near the counter name ①. For example, the counter pair is CNT0 and CNT1, which is used in this case.



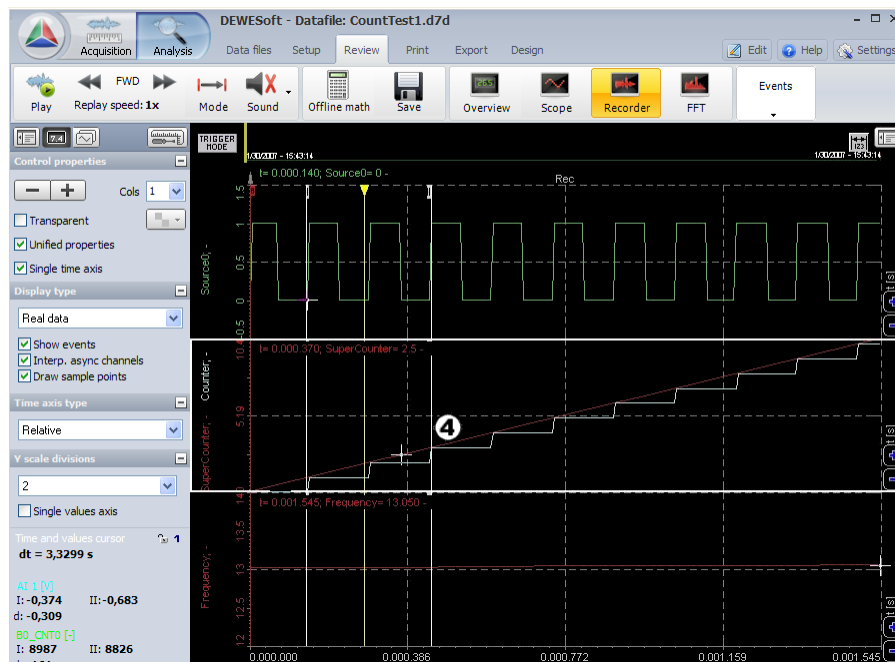
The counter 0 is set to simple **Event counting + Basic encoder counting** ②, and then the only thing left is to check the **Advanced counter mode** ③. Then the counter pair is set automatically and we have an exact count and exact frequency as the calculated output channels. The **Raw_Count** and **Raw_EdgeSep** are only for advanced purposes - they are raw values coming from the counters.



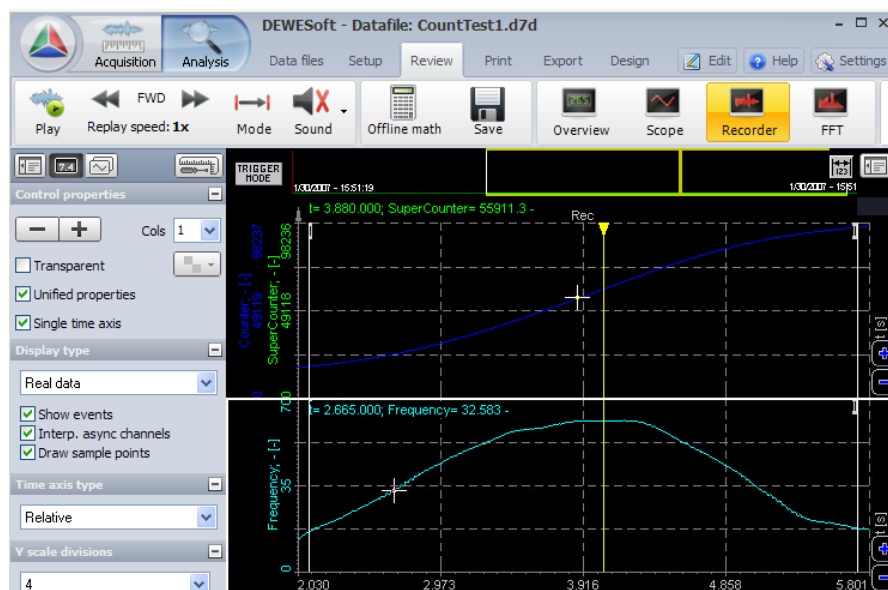
We have the Source0, also shown as a *digital line* and in the second graph the *blue* curve is the *normal* counter (raw counter values), which *increases* the value of *each* sample. Meanwhile the *red* one is the *super-counter* ④, where the values are *interpolated between the counts*, and even more importantly, also *between the samples*.

If you try this tutorial by yourself, try to set the *signal frequency* from *half* of the sampling rate up to *50% higher* than the sample rate. You will see *normal* counter staying the same for a the sample, then jumping, then staying the same again or jumping for two values. In short, the result will be really poor. But if a *super-counter* is used, the values will be *perfectly aligned* to the input as shown in the example below.

Also, the frequency measurement will be perfect in this case.



The data file below shows the *run-up* and *run-down* of the *test machine*, where the super-counter and the frequency are showing perfect measurement results. This is actually the recommended way of measuring all advanced DSA features like *order tracking*, *torsional vibration* and *rotational vibration*.



Please take a look at the [Frequency measurement](#) tutorial for a comparison of the different methods.

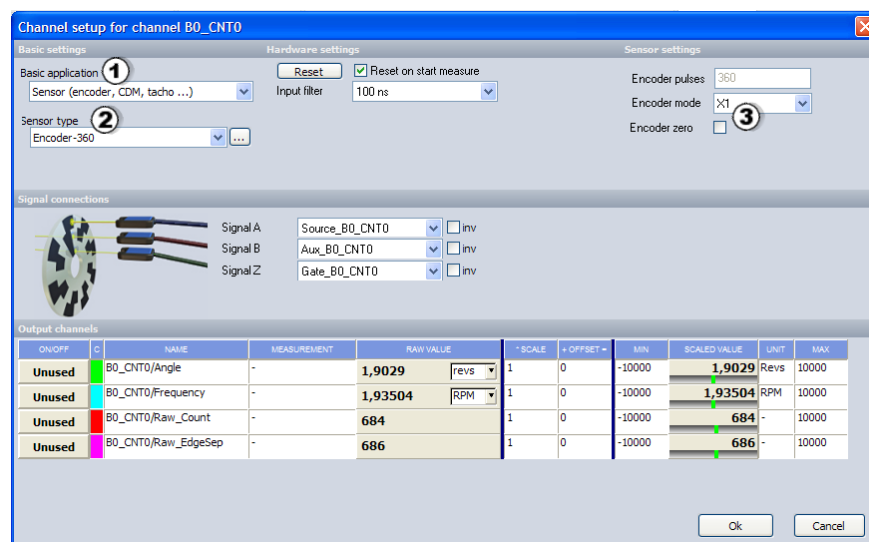
2.6.5 Sensor measurement

| | |
|-------------------|------------------------------------|
| Required hardware | Sirius ACC+, MULTI 15 pin, DEWE-43 |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

The super counter mode is also used in a special counter mode, called "**Sensor**" mode (selected from the **Basic application** drop down menu ①). This mode allows the direct use of the digital speed/position sensors as defined in the **Counter sensor editor**. You can choose *rotary encoders*, *linear encoders*, *CDM sensors* (angle sensors with zero reference), *geartooth with missing or double teeth* and *tacho probes*.

The only thing needed is to select an appropriate sensor from the **Sensor type** drop down menu ②. If the sensor is not yet defined, there is a three ellipsis button on the right side which opens the counter sensor editor. This is where sensors can be defined. The sensors will always run in the **supercounter** mode, showing the exact frequency and angle.

The benefit of using sensors is that scaling will be done automatically, so we don't have to worry about that anymore. There are still several options to choose from. For encoder, we can select the **Encoder mode** (X1, X2 or X4) ③ and either use the **Encoder zero** or not.



If zero is used, then there is a message telling us how many pulses between two zero points are seen, just to inform the user about possible setup or connection errors (as like shown in next picture).

For the **CDM** sensor, the only special setting is the *direction of the count* ④. The CDM sensor will count **up only** by default, but we can reverse that with this option.

The tacho and geartooth have no special settings, so they will depend solely on how the sensors are set up in the sensor definition.

2.6.6 Counters in automotive applications

| | |
|-------------------|------------------------------------|
| Required hardware | Sirius ACC+, MULTI 15 pin, DEWE-43 |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

We will look for two typical applications in automotive: *steering wheel measurements* and a *wheel speed sensor*.

Steering wheel sensor

In this case a measuring steering wheel with a *quadrature encoder sensor* is used to measure the **angle position** and the **angular velocity** of the steering wheel during test drives. The quadrature encoder used in our example has a *resolution of 1800 pulses per revolution*.

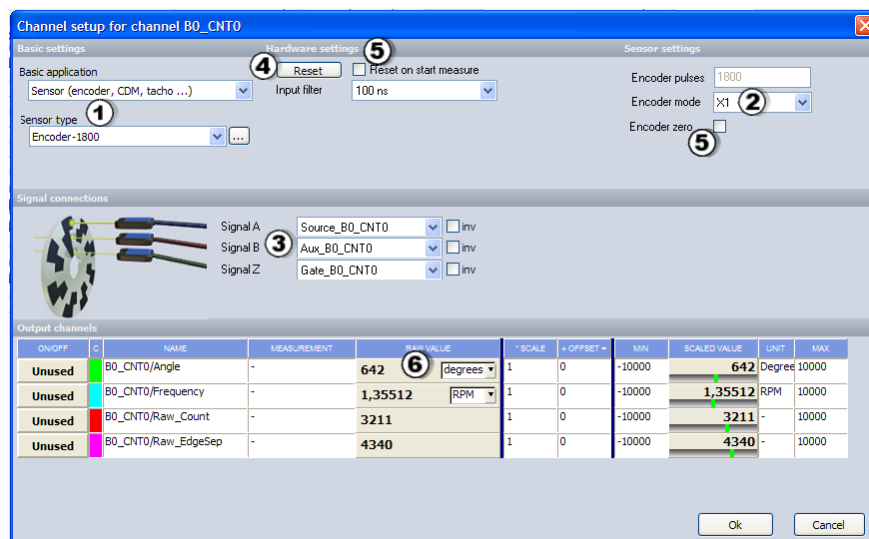


Channel Setup

Let's first set up the *steering wheel sensor*.

For the Counter mode, we should select the **Sensor (encoder, CDM, tacho...)** from the **Basic application** drop down menu to decode the signals of the quadrature encoder sensor and select the appropriate sensor (**Encoder nn**) from the **Sensor type** drop down menu ①. With **Encoder mode** the *resolution* of the angle measurement can be set. In this example "**X1**" ② is used and the counter *outputs 1800 pulses* per revolution. "X2" would output **3600 pulses** per revolution and "X4" would output **7200 pulses** per revolution. It is recommended to use an **Input filter** to avoid measurement errors caused by jitters or spikes on the signal (in this case **100ns**). The quadrature encoder sensor *normally* has *three* signals. Only **Signal A** and **Signal B** are used for this application ③.

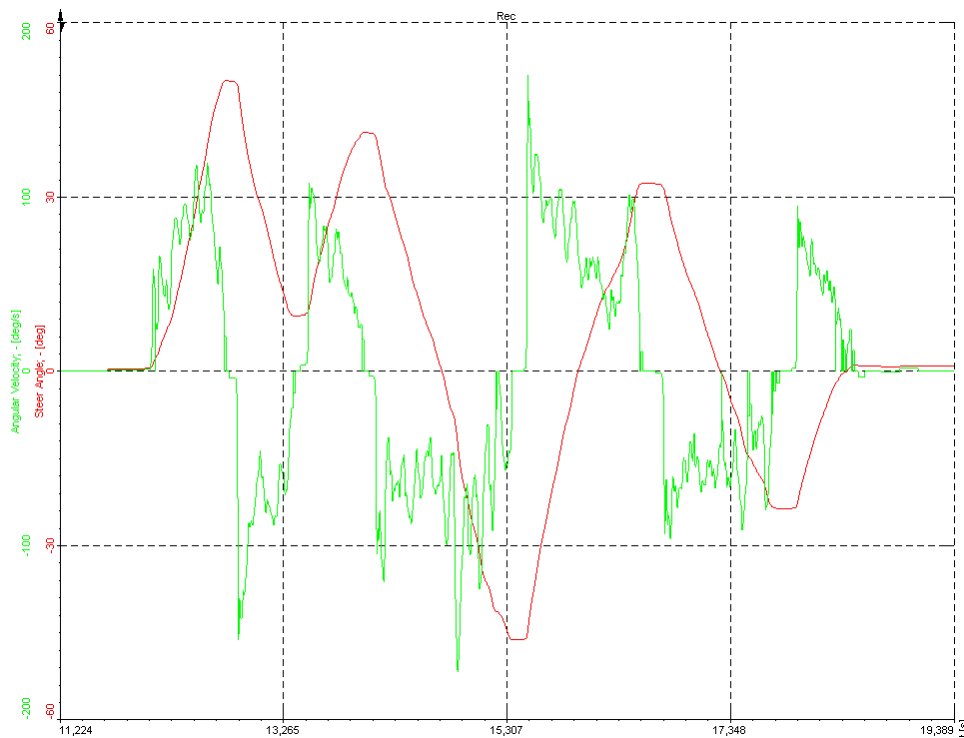
It is important that *Encoder zero* is *not chosen* if signal Z is connected to the counter input, else a jump in the frequency channel can appear and the manual zero point definition can be lost. The preferred way is to simply choose the sensor from the list, where all the scaling will be done automatically for us. The **Encoder zero** has to be *de-selected* and **Reset on start measure** should be *so as de-selected* not to lose the initial zero point correction.



The *first* output is the *angle*, where we can select **degrees** as units (this could also be **revs** or **counts**) and the *second* output is the *frequency*, which is already scaled in **RPM** or **Hz**.

Zero Point Definition

After all the settings in the channel setup are made, the steering wheel has to be **set to zero** for a specific *steering angle*. In most cases the steering angle is set to zero while the car drives *straight ahead*. Zero definition can be done by pushing **Reset** ④ in the channel setup. This way, the steering angle is set to zero in that steering wheel position. A *horizontal* test track is recommended for the zero point definition to avoid a steer angle offset error. It is very important that **Reset on start measure** ⑤ is *not selected*, else the counter value is set to zero at *every* measurement, and thus the zero point definition will be *lost*. The figure below shows the measurement results of the steering wheel.

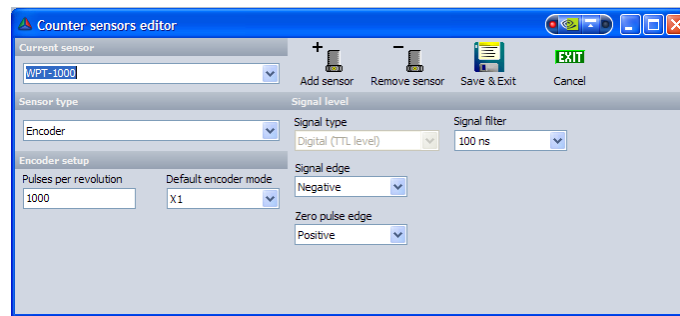


Wheel speed sensor

The wheel sensor is used for high precision **wheel speeds** and **rotations** at test drives. We can also measure traveled distance and speed, but for this we need to know the *dynamic wheel radius*. The wheel radius, however, depends on the operating state of the wheel, e.g. road condition, driving conditions, cornering ability, tire pressure and other factors. Therefore a *precise distance* measurement with dynamic wheel radius scaling is *not recommended*. Instead, we could use either a GPS solution like VGPS or radar/optical sensors.

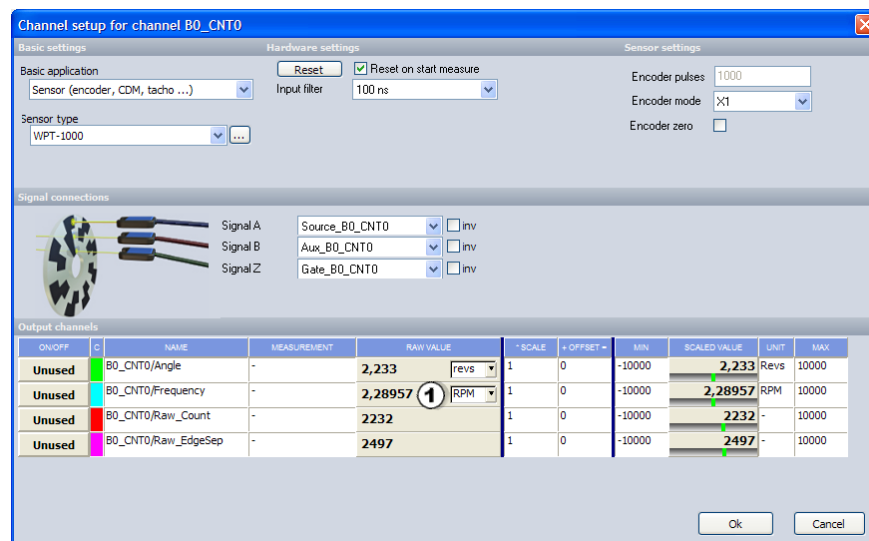


Again, we need to set the *channel setup* for the wheel sensor. This sensor is not defined as standard in DEWESoft, therefore we need to create it with the **Counter sensor editor**.



After the sensor is created, it just has to be selected in the sensor editor.

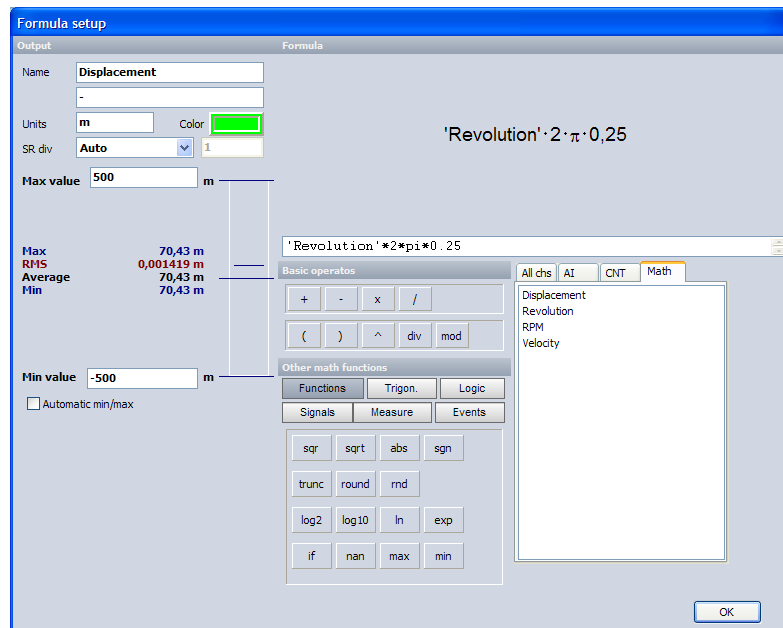
The *angle output* can be set to *revs*, *degrees*, or *count*. The unit for *frequency* can be set in Hz or RPM ①.



After the counter setup is finished, the **math** channel setup for traveled distance and velocity can be set. The *first* math channel is used to calculate the *displacement*. This is done with the *dynamic wheel radius*.

The equation for *displacement* reads:

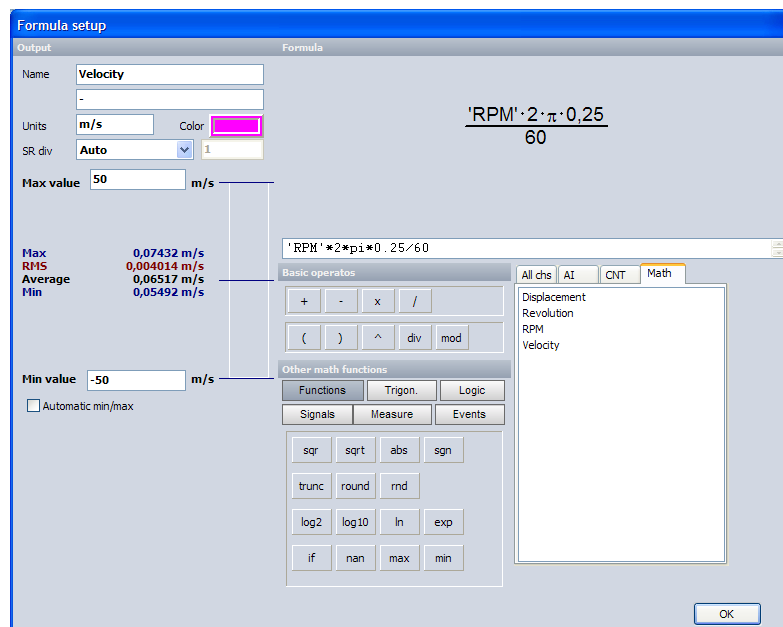
$$"B0_CNT0/Angle" \cdot 2 \cdot \pi \cdot 0.25 \text{ [m]} \quad \dots \quad "B0_CNT0/Angle" \text{ is the angle channel output from the counter. The dynamic wheel radius in our case is } 0.25 \text{ m.}$$



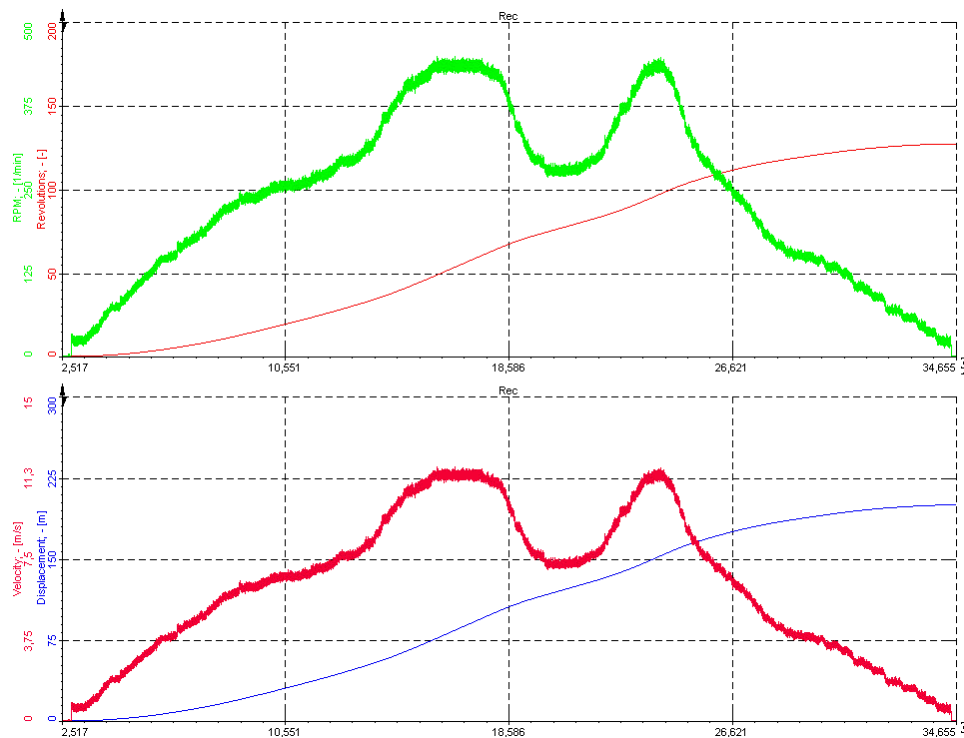
The *second* math channel is used to measure the *velocity* of the car. The same *dynamic wheel radius* is used for this calculation.

The equation for *velocity* reads:

$\text{"B0_CNT0/Frequency"} \cdot 2 \cdot \pi \cdot 0.25 / 60$ [m/s] ... "B0_CNT0/Frequency" is the *frequency channel output* from the counter, if it is set to **RPM**.



All - measured and calculated – channels are available in real time without any hardware or additional wiring. The figure below shows the measurement results from the wheel sensor.



2.7 Video acquisition

We have four basic types of **cameras** which are supported in **DEWESoft**:

- *low* speed (up to **30 FPS**) web cameras or camcorders supported by **DirectX**
- *medium* speed (up to **600 FPS in VGA resolution**) cameras (DS-CAM)
- *high* speed cameras (up to **20.000 frames per second**), where we combine data and video in post processing or directly use the Photron driver inside **DEWESoft** (please consult the user's manual for details)
- *NEC thermovision* cameras

So which camera to choose for a certain application? It is clear that the thermovision cameras have their special applications while the high speed cameras are used to acquire short triggered snapshots where we need extreme video rates to capture crashes, explosions and other fast events.

The decision between a good camcorder and a medium speed camera is not that easy. The main difference between these types of cameras and the high speed ones is that with the medium and low speed cameras we can continuously store video stream to the disk until we run out of disk space. We can also use software triggering on the video to reduce the amount of data or perform the online compression.

However, the system needs to have a good performance to stream video. We will need *high performance hard disks* and a *very well built system*, as we might still run to the limit of performance. We have to know that the typical VGA size image takes 300 kB. If we have 100 frames per second, we need to store 30 MB/s for one camera.

Clearly, if we need high speed video, we need to use either a DS-CAM. DS-CAM 120 has a slightly *higher speed* (120 FPS in VGA). DS-CAM 600 compresses the picture in camera and we can achieve 600 FPS in VGA mode. A big advantage of both camera types is that they can be *triggered* from the *analog card* and therefore the data and video are *perfectly aligned*.

If 25 or 30 pictures per second are enough, we might consider using a camcorder. I would suggest *progressive scan* cameras, so as not to have interlaced pictures.

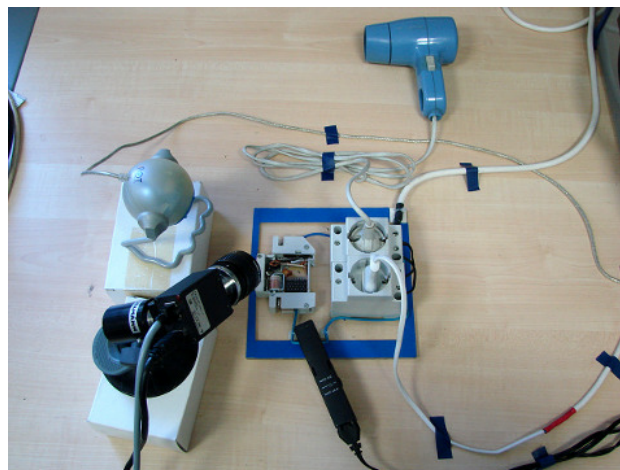
Web cameras are usually low price/low speed/low quality, but are an extremely helpful tool *to document* the experiment. We had lots of feedback from customers telling us that a simple, even poor picture helped them to understand the recorded data much better.

In the following test we will see a basic difference between a web cam and a medium speed camera.

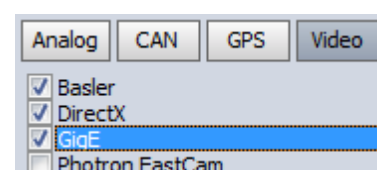
| | |
|-------------------|-----------------------------------|
| Required hardware | Sirius, DEWE-43, Web cam, DS-GIGE |
| Required software | PROF (for DS-GIGE support) |
| Setup sample rate | At least 1 kHz |

2.7.1 Channel setup

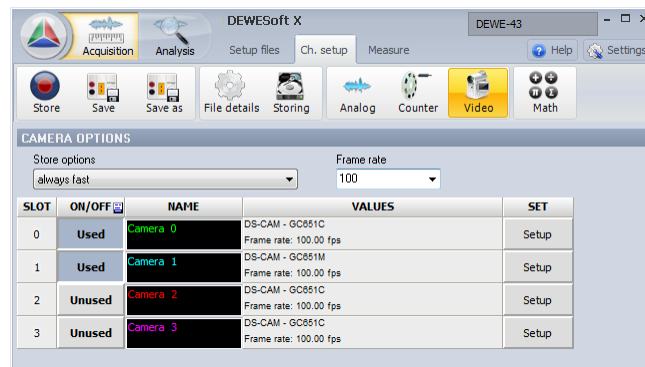
For this test we opened one *0.5 A fuse*, measured the **voltage** and the **current** at the *output* of the fuse and used a *hair dryer* to *switch* the fuse. The Basler and web cam were recording the video.



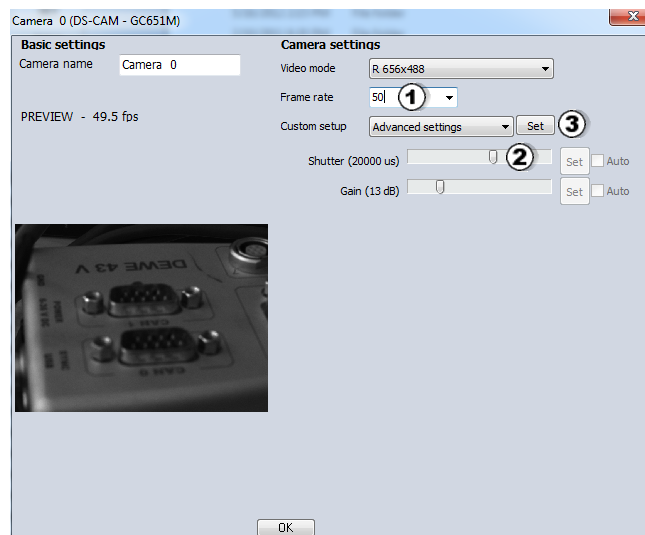
Even though it is against the practice of this tutorial, I would still like to show the **hardware settings** of DEWESoft. There are three camera types selected: Basler, DirectX (which recognizes web cams and camera phones) and GigE. We will measure this experiment with *medium* speed GigE cameras.



In the setup screen of DEWESoft we see all connected cameras. We select needed cameras.



First we **set up** the cam. We set the **Frame rate** to 50 **fps** ① and resolution to **656x488**. The next step is to set the **Shutter** speed ②, which can't be longer than **20 ms** to be *able* to acquire **50 fps**. Lower shutter speeds will *reduce* the smearing of picture with fast movements, but will *also reduce* the brightness of the picture. Therefore we will need either a strong light or we will need to *increase* the **Gain**. On the other hand, this will increase the noise in the picture and will reduce the picture quality.



The **settings** of the cam depend on the capabilities of the camera. There are a huge differences between one web cam and the next in terms of speed, picture quality and available functions. Some cameras have automatic shutter and automatic gain, there are even a few with automatic focus. This changes so often that it doesn't make sense to mention any camera brand in particular in the manual. Dewesoft is testing new cameras on the market all the time, so it is worth asking them what the latest and greatest is.

Some cameras have different *compression* types like **YUV** or **I420**. This means that the each pixel will not have **24 bits** of data (8 bits of data per color), but *less*. In short, using such modes will result in *smaller* picture sizes and will reduce the data file size in the end, but the colors might not be as perfect as with **RGB** (uncompressed). However, the human eye is much more sensitive to scales of gray than to shades of colors. These compression algorithms use exactly this fact, therefore we might not even see any difference.

Usually, the cameras also offer **Custom settings** ③, which will show all the special functions of the specific cameras, such as flipping, rotating picture and more.

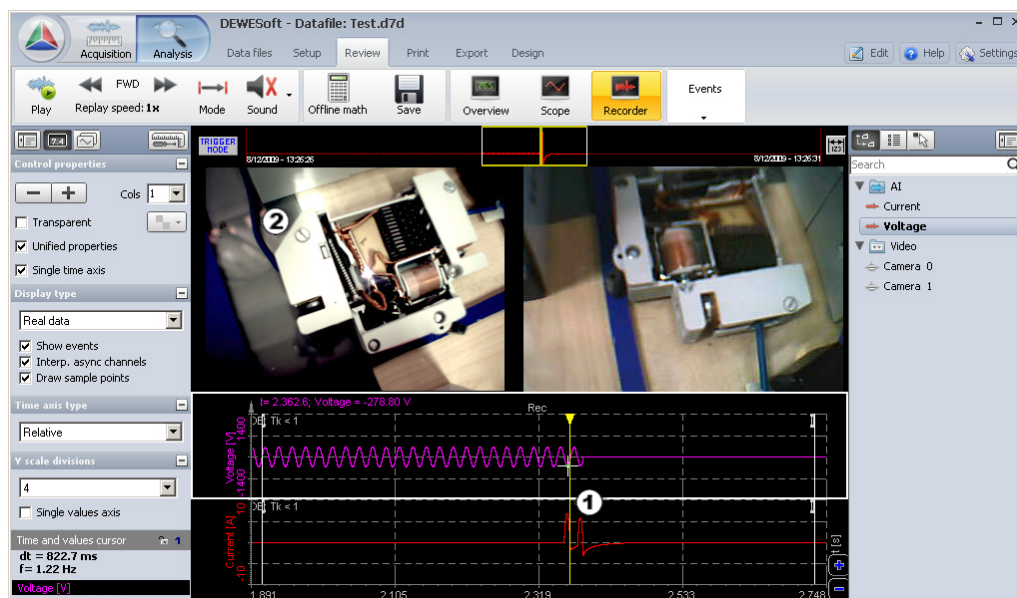
For acquiring *analog data*, we set *one* voltage and *one* current *channel* the same as it is in the **Voltage and current**

tutorial.

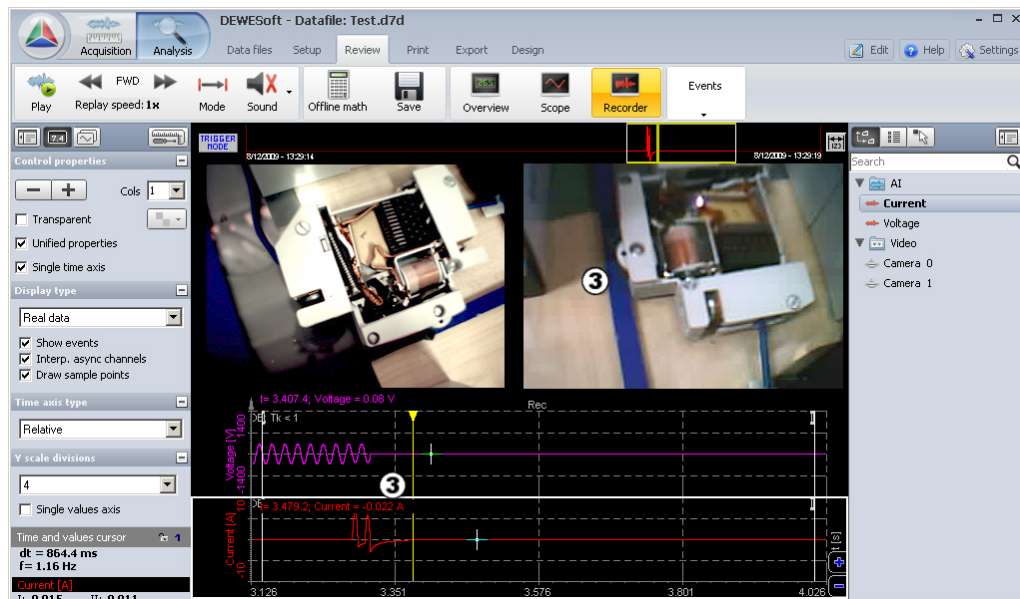
| SLOT | ON/OFF | C | NAME | AMPLIFIER (007) | PHYSICAL VALUES | CAL | SETUP |
|------|--------|-------|------------|--|--|------|----------------|
| 1 | Used | Share | Voltage ① | SIRIUS-HV Voltage: 1000 V; 50 kHz SN: D0C833F4 | U -312.28 / 312.18 V -1000 1000 | Zero | Auto Set ch. 1 |
| 2 | Used | Share | Current1 ② | SIRIUS-HV Voltage: 50 V; 50 kHz SN: D0C81ED0 | I -0.002 A -50 50 | Zero | Auto Set ch. 2 |

2.7.2 Measurement

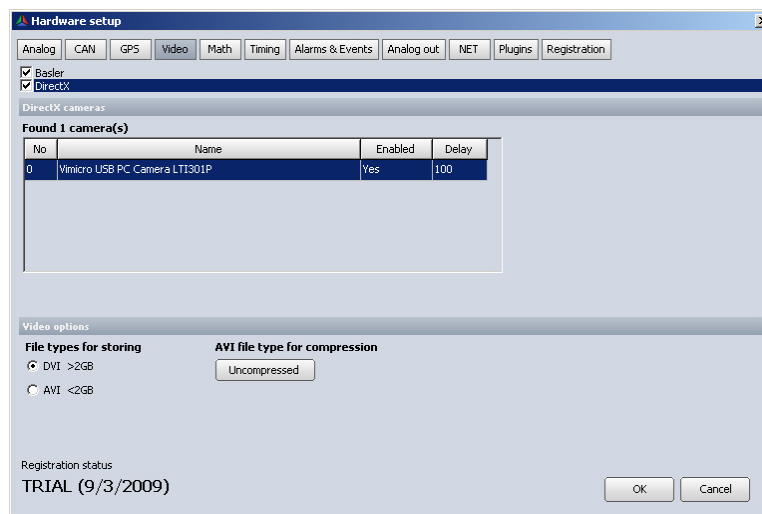
So let's acquire the data and see the results. The *high speed* camera is *synchronized exactly* to the analog data therefore we can compare the *switching times* with analog voltage and current. The *current* is flowing through the fuse for approximately **20 msec** before it really *switches off* ① and we can see nicely the when *spark* switches the fuse off. It is also very nice to see the position of the switch (on the left side of the fuse) as it goes to the off position ②.



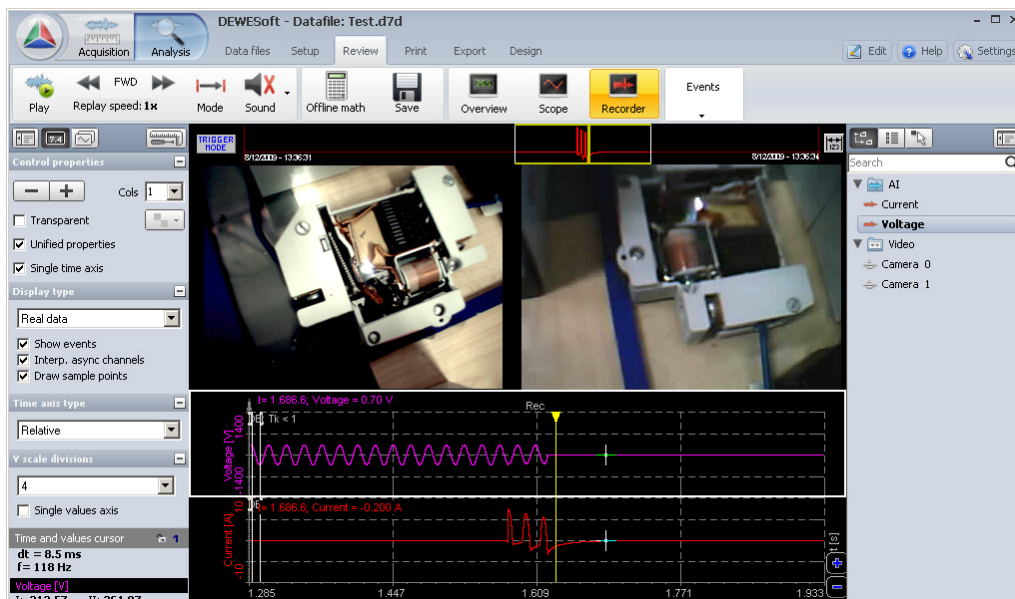
The picture from the web cam is nice, but doesn't show this at all. Because the web cam is neither *clocked* nor *synchronized* with analog data, it is *time stamped* as the picture comes in to the computer. Thus, we can see the switch off with a *delay* of approximately **60 ms** ③. In conclusion, web cams have two limitations: *speed* and *time accuracy*.



However, there is a way to *reduce* the time inaccuracy by entering a *camera delay*. Usually this delay is quite constant and can be compensated. We can do this in the **Hardware setup** by entering this value in the **Delay** field in the table. We will still have a time jitter of each frame, which can be in the range of **30 ms** or even more, when the system is at the limit its performance, but here we can't compensate it. If time accuracy is needed, we should look for *clocked* cameras like DS-CAM.



Now if we look at the repeated measurement, both cameras show *approximately* the *same* time of the event, however, the web cam shows *only one* frame when the fuse switched off while the DS-CAM camera shows quite several frames to be able to also see the spark, the movement of the coil and the switch.

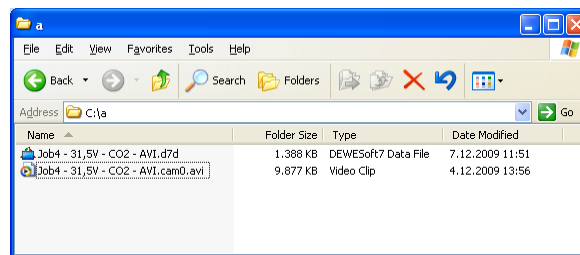


This solution, however, is limited to 600 frames per second. Some events require much faster cameras. These cameras store the picture in internal memory and after that to the video file, so we need to combine the picture and video in post processing.

2.7.3 Video post synchronization

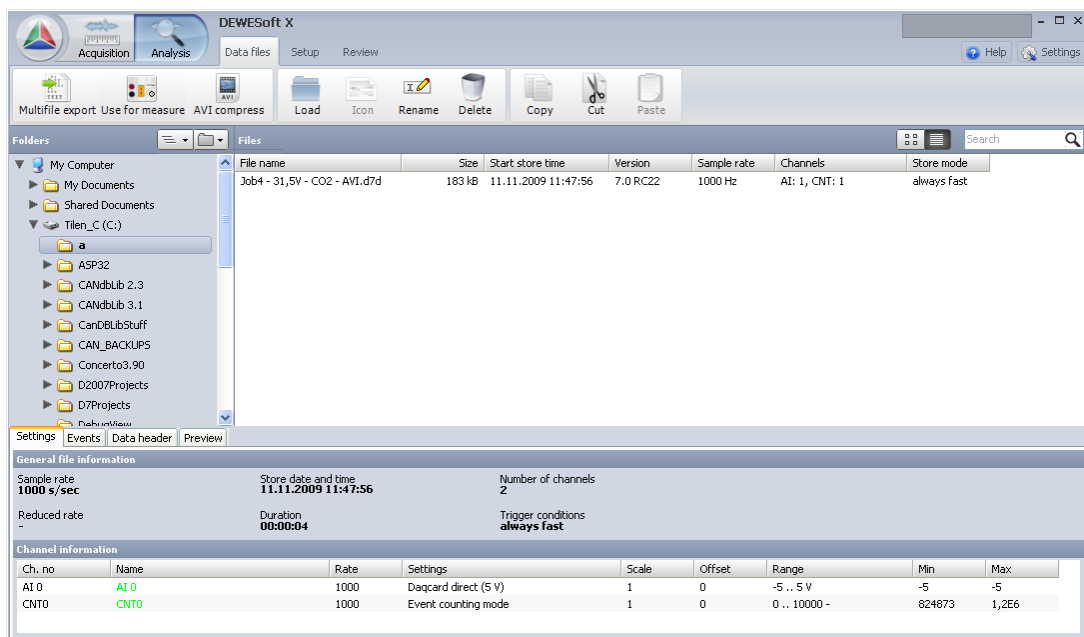
To include the *high speed* video in **DEWESoft**, only few steps are necessary.

1. After acquisition of the video file and the **DEWESoft** data, please **copy** the *.avi camera file to the *directory* where **DEWESoft** data file is located.
2. Please **rename** the video file to this specification: **xxxxx.cam0.avi** where **xxxxx** is the *name* of the **DEWESoft** file to be synchronized. If we have more video files, we can name them **xxxx.cam0.avi**, **xxxx.cam1.avi** and so on.

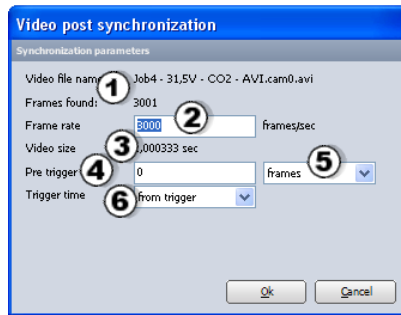


The picture above shows an example of what this looks like.

3. After this, open **DEWESoft** and enter the **Analysis** mode.



Double click on file to open it and **DEWESoft** will *recognize* that this file has no synchronization information included and will ask to synchronize it manually.



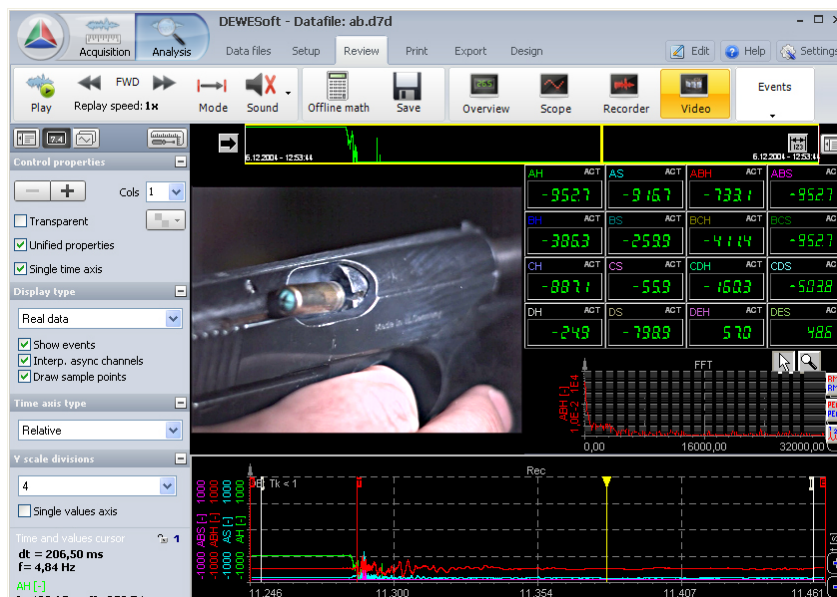
It will automatically *recognize* how many frames are stored ① in the file. Please enter the correct **Frame rate** of the camera ② (sometimes the .avi files hold the correct values, but most of the times not). In the info field **Video size**, you will enter the video length in seconds ③.

Also enter how many pictures were taken before the video trigger - **Pre trigger** - occurred ④. You can enter it in **frames**, **seconds** or **milliseconds** ⑤.

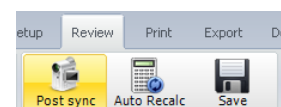
Trigger time can initially be selected **from trigger** (so long as the storing of analog data has been triggered from the *same* trigger source) or in the relative time **from start** of measurement in **seconds** ⑥.

When finished, **DEWESoft** will go to the video screen to see the result of the synchronization and the recorder will show the video frame ticks aligned with the analog data.

You can go *back* and *forth* with the **yellow** cursor to observe the quality of the synchronization.



If you need to **realign** the video, please select menu item **Post sync**. Here you can *change* the parameters. Also, there is the option to select *start* of video from the *current position*. In this case, the **yellow** cursor will be taken as the *origin of synchronization*.

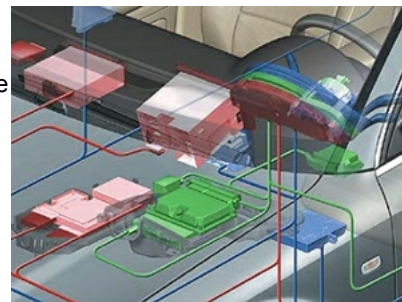


In order to **store** the synchronization info for future post processing once you are done and satisfied don't forget to choose **Save**. Be careful, this can be done *only once* and since it is written into the *original* video file, it is recommended to keep a *copy* of the *video*.

2.8 CAN bus acquisition

Did you know that the average car today has several kilometers of wires inside it? This figure is astounding and therefore car manufacturers have been looking for ways to reduce it. With the emerging technology of today's vehicles there are more and more devices with a need to *communicate with each other* and to *share their data*.

As a result, in 1980s the company Bosch invented the serial bus where data can be sent with up to **1Mbit/second** via a *single twisted pair wire*. This reduced the amount of wiring by a lot (and still there are kilometers of wires). Still, we can easily say that without it, today's cars would not be the same.



The **CAN bus** is not only related to cars. They appear in other forms of transportation like trucks, rail cars, tanks, tractors and other vehicles as well as for **common measurements**. There are *lots of sensors readily available* with CAN bus technology. They are very *robust, fault tolerant* and have a nice *collision detection algorithm*.

So, now we know what the CAN bus is. What do we need to measure the **CAN bus**? First, we need some hardware supported by **DEWESoft**.

| | |
|--------------------------|---------------------------------|
| <i>Required hardware</i> | Sirius, DEWE-43 or DS-CAN2 |
| <i>Required software</i> | LT or higher + CAN option or EE |
| <i>Setup sample rate</i> | At least 1 kHz |

Then, if we measure in a car, we need to know *where to connect* this in the car. For "official users", this task is trivial since the wiring is known. We need be careful that the *not-terminated* wires are *not too long* since the vehicle bus can be interrupted with the connecting of the measurement instrument. If we have a sensor, or an array of sensors, there are connectors with labeled wiring. In this case, we need to *make a bus*, which is not that hard as it sounds, since it is *only a twisted wire* with **120 Ohms** resistors at *both ends* and virtually any number of connections in between.

Only by correctly connecting to the bus can we scan all the messages sent through the CAN bus, but we will *only see a message ID* and *raw data*.

The third thing is that we *have to know* how to *decode* the messages from the CAN. For this we either need a description or a library. These libraries are defined by a **Vector** and are called **DBC files**. It has a full description of the messages with output channels. These libraries are a well hidden secret among car manufacturers and are propriety. For trucks, the messages are standardized and described in the J1939 standard.

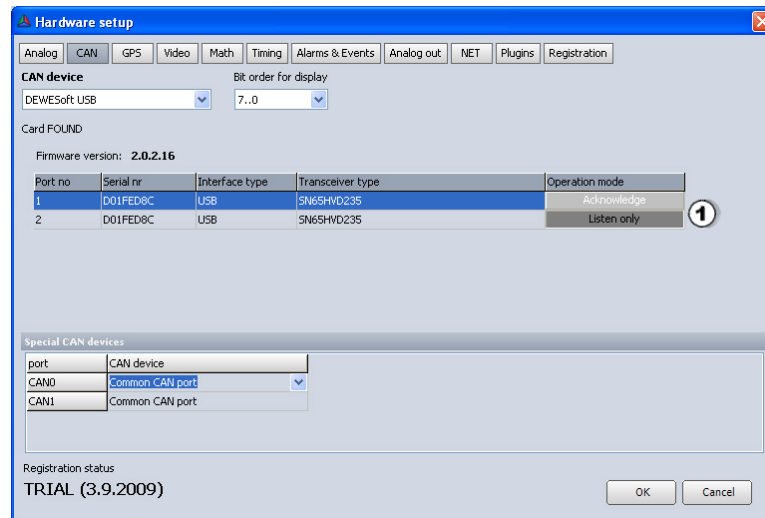
2.8.1 Channel setup

Let's take the same steps as defined in the previous chapter. First of all, let's make *general setup*. There are two types of devices or to put it better *two types* of operation.

- When connected to vehicle bus, for example, we only need to *listen* to whatever *is happening* on the line.

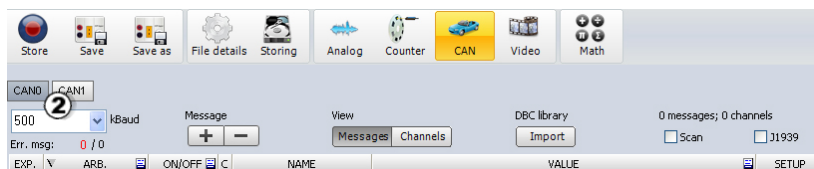
- For dedicated sensors, it is often necessary to *send an acknowledgement* that the data was received

We need to **set up** the *hardware* to meet the application we have. This is done in the **Hardware setup**.

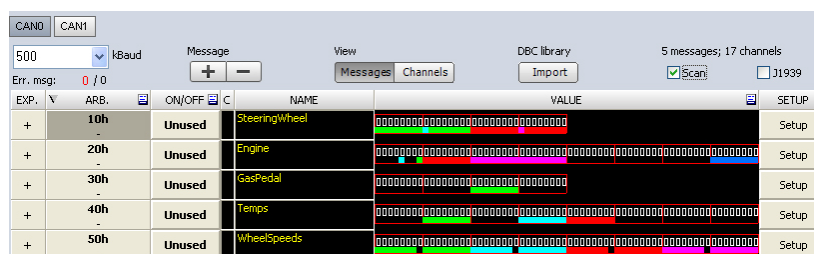


Clicking on **Operation mode** button changes it between **Listen only** mode and **Acknowledge mode** ①. We need to set this up *first, before connecting* to the bus so as not to interfere with the bus operation.

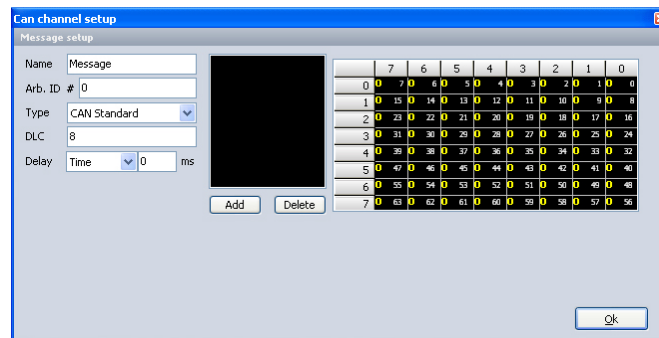
The baud rate setting ② is very important. In fact, some vehicle operations can be *interrupted* if we connect to the bus *with the wrong* baud rate set. Under the baud rate edit box, we also have a notification of *how many* messages came through the bus and how many of them were *corrupted* (red). This information shows if the baud rate is correct and also if the bus has any problems due to bad connections or bus overload.



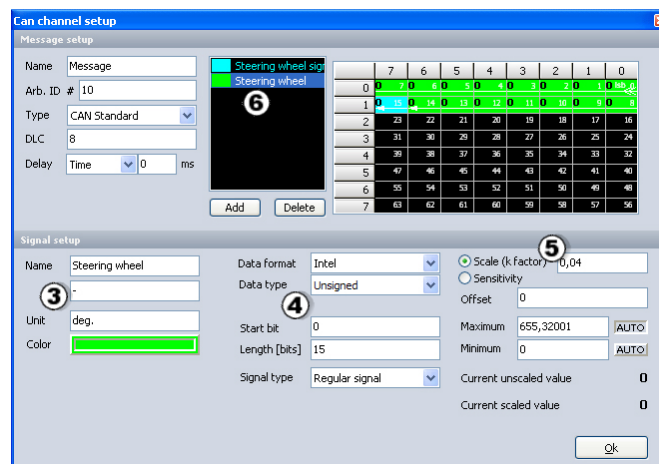
Next we can **scan** for the messages. As soon as we check the **Scan** option, the messages that are coming from the bus will be displayed. So now we can see message *IDs*, *speed* of the messages and the *raw binary values* coming from the bus.



If we know them, we can **define** the channels from the specification. We choose the **Setup** button and the empty message setup screen will appear.

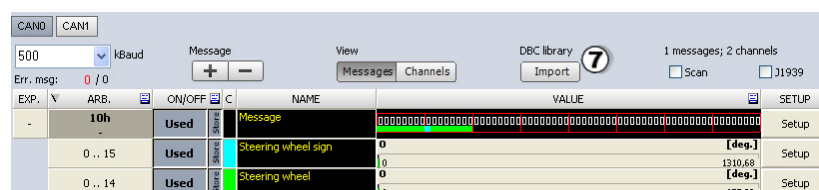


And now... we have to know... or guess. We know that the first 15 bits of this message is the *steering wheel angle*. We need to *add a channel* by clicking the **Add** button. We enter the **Name** of the channel, the **Unit** ③, and the **Data format Intel**. We set the **Data type** as **Unsigned**, change the **Length** of the channel to **15 bits** ④ and enter a **scaling Factor** ⑤



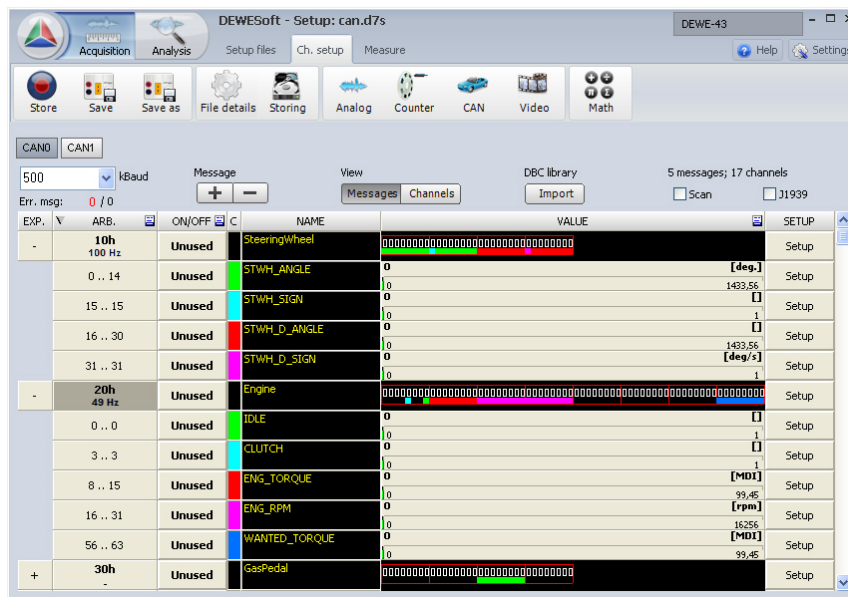
Let's add another channel - *Steering wheel sign* ⑥ which is just **one bit** long and starts at **bit 16**. So far, we have defined two channels that can be used in our measurement.

If we close the message setup, the two added channels will appear *under* the message (if we expand the message to also show the channels).



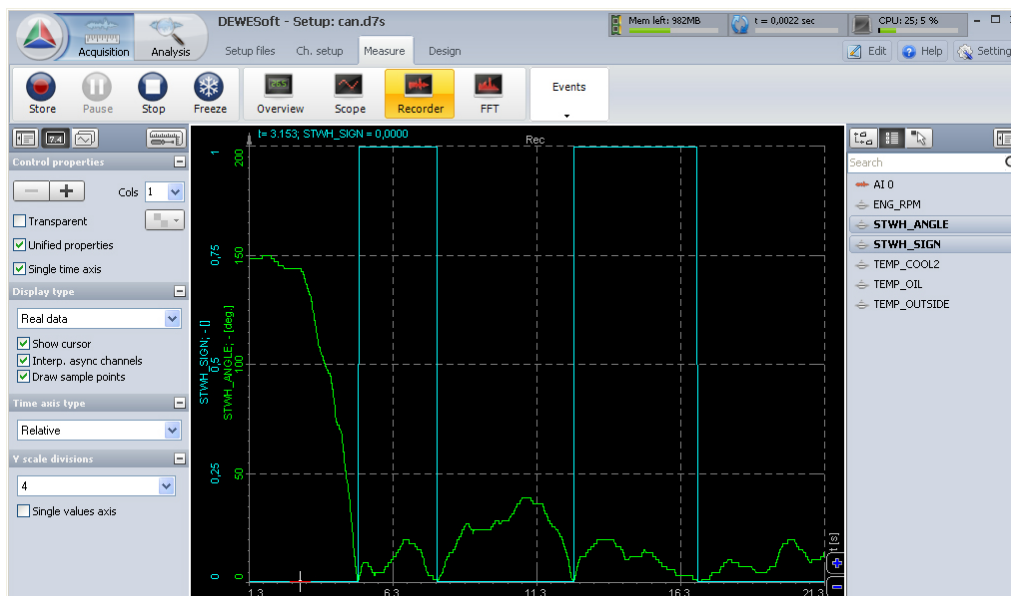
The second way to add messages is to simply *load* the **description file**. The data format of description file is *. **dbc**. We select the file by clicking on the **Import** button ⑦. For our test, the description file loads *all* the channels for the messages.

The data appears on the bus with different rates - we can see that more important information has *higher data rates*. The *steering wheel* in this case has a speed of **100 Hz**, the *engine* info has a speed of **50 Hz** while the *temperature* has a speed of only **5 Hz**. The message ID *priority* of the CAN bus forces the manufacturers to make it so that the most *important* information has the *lowest* arbitration ID and therefore the *highest priority*. Now let's select a few channels and see how they appear in a real measurement.



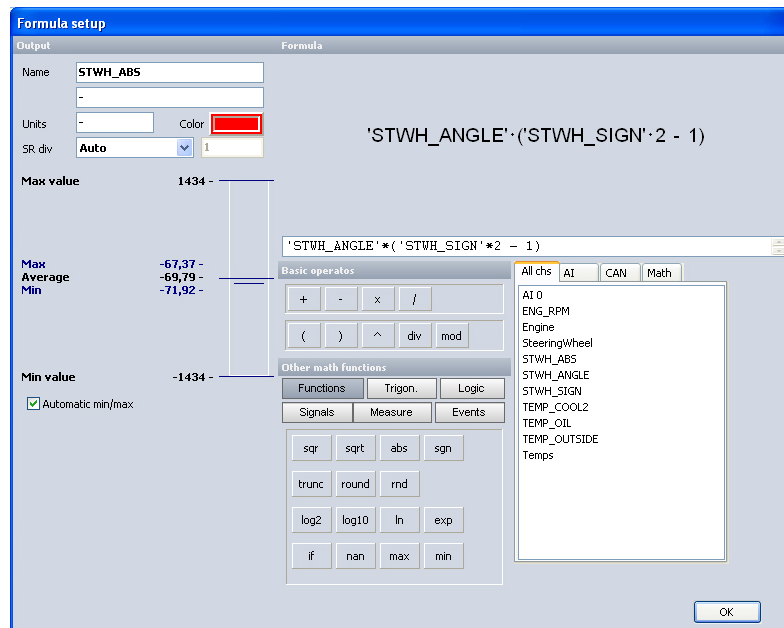
2.8.2 Measurement

Let's just look to the *recorder* to see the data. Look at the *steering wheel angle* and the *steering wheel sign*. The data is coming from the bus at regular intervals. We see that when we move the steering wheel *from left to right*, the sign changes from 0 to 1 and the angle shows the *absolute deflection angle*.



This is a good example to show how the **mathematic channel** could help. Let's create a new *formula channel* and think about how to create an *absolute channel* out of sign and angle. First of all, we need a *real sign*, like *-1* for *negative* and *+1* for *positive* values. Since we have *only 0* and *1*, we need to *scale* it. An algebraic way of doing it is to *multiply* it with 2 (we get 0 and 2) and *subtract 1* (we get *-1* and *1* for the input value of 0 and 1). Next we *multiply* this with the *angle*. Thus, the equation looks as follows:

$$"STWH_ANGLE" \cdot ("STWH_SIGN" \cdot 2 - 1)$$



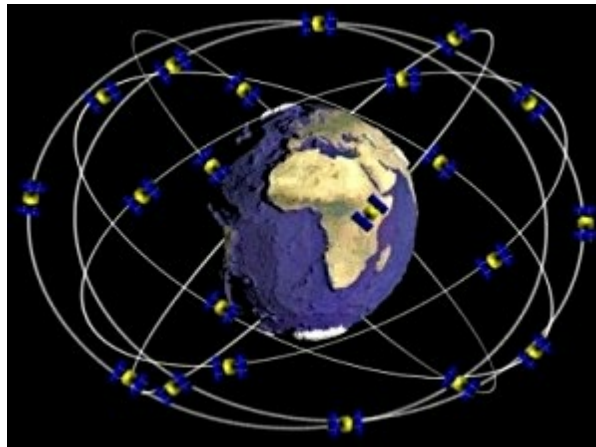
Now let's add the *resulting channel* in the *recorder*. We see that the angle is *negative* when the *blue* curve (sign) is *low* and *positive* when the *blue* curve is *high*. The formula in DEWESoft has a useful feature in that it checks the *input rate* of the signals. If a signal is *asynchronous* (that means if it gets samples at irregular intervals), the *output channel* is *also* asynchronous. Since the two combined channels are coming from the *same* CAN message, they will have the *same* time stamp and we can *combine* them in a formula. This way, we can *be sure* that the *output* will have *exactly the same speed and time stamps* as the *input channels*.



2.9 GPS acquisition

GPS stands for global positioning system. Actually, the abbreviation comes from the US system, also known as Navstar, which was the first positioning system ever. There is another system from Russia called GLONASS. In Europe, the system is called Galileo.

The system consists of *several* (24 for GPS systems) satellites in an earth orbit transmitting the time information and satellite location. From this information the user can **determine the position on earth**. This is calculated based on the triangulation method, so we need at least three satellites to determine the position.



This is very well known and the most commonly used application, but there is more to it. Better receivers can use the Doppler effect method to *precisely calculate the speed*. The accuracy of speed measurement can be better than **0.1 km/h**. Therefore we can also use receivers like VGPS as *speed sensors*. A low cost market GPS doesn't have this option. The output update rate for speed can vary from **10-100 Hz**.

The third possible application comes from the way how the GPS system is functions. Since the satellites are transmitting *exact absolute time* and better receivers usually output this pulse with a high precision (below one microsecond), we can use this technology to **synchronize remote systems**. Actually there is *no distance limit* - we can accurately synchronize two systems with one placed in California and the second one in India. With that, we can clearly see if there is a correlation from butterflies flapping their wings in San Francisco to the earthquakes in India (just kidding, but this actually very useful to recognize the source of faults on power lines, for example). This technology is available with GPS-HS, GPS-HSC, Minitaur or S-BOX hardware.

The quality of the signals coming from the GPS depends a lot on the *number* of satellites recognized by the receiver. The signal can't really go around a corner or through walls, so a clear line of sight is very important. High buildings and trees will obstruct or block the signal. High buildings can also produce reflections, which are a major cause of measurement errors, even though the number of available satellites is high.

In general, we can get a *good* signal if we have **six or more** satellites; otherwise we get jitters in speed and position.

We can use different *antennas* for the GPS. Usually there very tiny ones all the way to quite huge ones. The difference between them is that the *small* ones need a *ground plate* to deflect from the earth. The ground plate should be at least 30x30 cm big. The car roof, if it is metal, is a perfect ground plate for the antenna. For other applications, we need to be sure

to *make* a ground plate or we should use an antenna with an *integrated* ground plate.

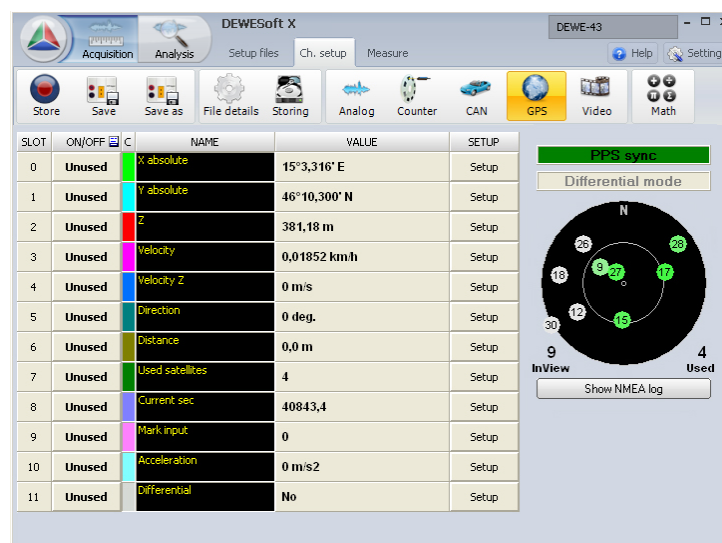
2.9.1 Channel setup

| | |
|-------------------|--|
| Required hardware | DS-VGPS-HSC, DS-VGPS-HS, Minitaur, S-BOX, Novatel, Javad, or any NMEA compatible GPS |
| Required software | Any version |
| Setup sample rate | At least 1 kHz |

When **GPS** is selected in the **Hardware setup**, we get another tab in the **setup** screen. We have several channels to choose from - position in x, y and z axis, velocity, vertical velocity and direction, used satellites, mark input (the external event) and the current second of the day. The acceleration and distance are *calculated* from the velocity channel.

On the bottom, we have a sky map that, shows the current *satellite constellations* in the sky. The satellites that which are being currently used at that moment are drawn in **green** (if the receiver supports GLONASS, then the satellites are shown in **red**) and the color shows the strength of the signal. **Pale green** signifies a *weak* signal and **dark green** is a *strong* signal. Then we have information about whether **PPS sync** is available. This is information about the receipt of the *pulse per second signal* over the GPS interface (RS232 or USB), which can, if available, *enhance* a synchronization to other data source a lot. If the PPS sync is not available, we need to switch it off in the **DEWESoft Tuner** utility under the **GPS** section in order to *receive* the data from such GPS. Usually this signal is *available*, but some recent low cost receivers don't have this.

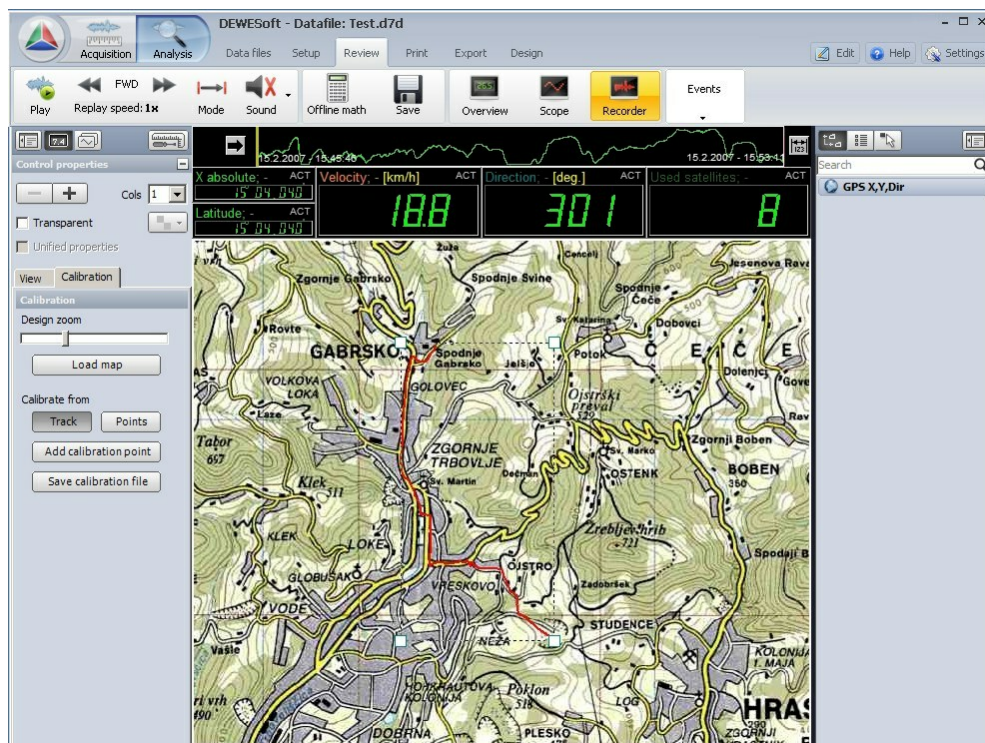
Further, if the receiver *supports* differential mode from SBAS or WAAS and these signals are *used*, this will be shown in the **Differential mode** indicator.



When we *choose* the *channels*, we can take some **measurements**. A part of setting up the GPS is to assign a **map** to the **GPS display**. Now let's take a look how to *add a map behind* traveled path of the GPS. If we have a map of the area in either **bmp** or **jpg** format, we can adjust it to the GPS.

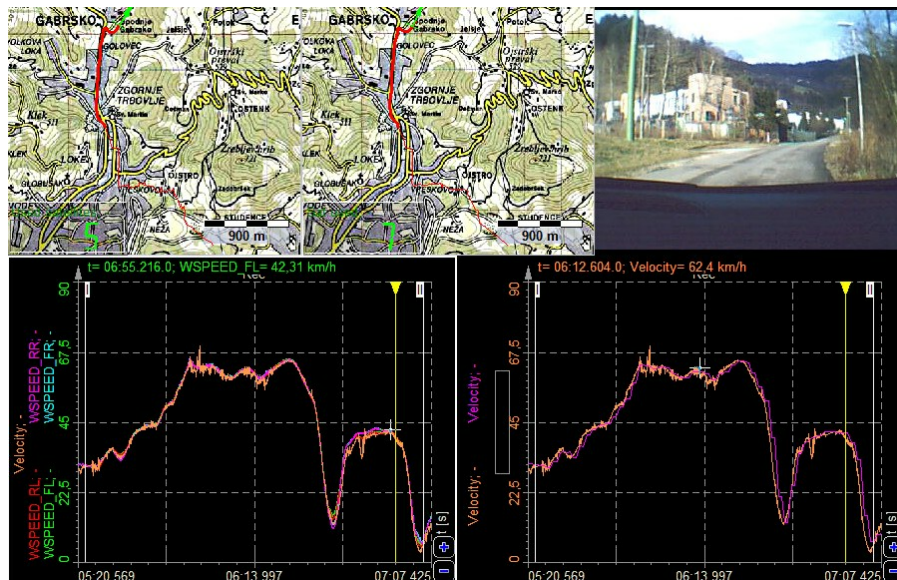
If we *don't know* the coordinates of the map, we can take a *short tour* in the area *storing* the measurement. Then we go to the GPS screen, go to **Calibration** and **Load map**. The map is shown in the background, and the traveled path gets four handles around it. Now we can *drag* and *adjust* those handles to match the position and the map.

If we *know* the *exact* coordinates on the map, we can also *enter* them by switching to the **Calibrate from points** mode. Once this is adjusted, we can **Save calibration file**. From that point on, this map will *always be shown behind each* measurement. We can also have multiple files for *different zoom levels*. We can have an overview file like this, and when we zoom in, it will *switch to the picture*. Now is a good time to go for a drive and see what this looks like.



2.9.2 Measurement

We used *two* **GPS** receivers in our measurement. One is a *low cost* 1 Hz receiver, and the second one is a *high speed* 100 Hz GPS. I used two GPS visual controls - the *left* one has a high speed VGPS and the *right* one has a low speed GPS. I have also *connected* the **CAN bus** to show the *wheel speeds* (bottom left recorder). The recorder on the right has an *orange* line, displaying the *high speed* GPS velocity and the *pink* line shows the *low speed* GPS velocity. The web cam is just for *reference*.

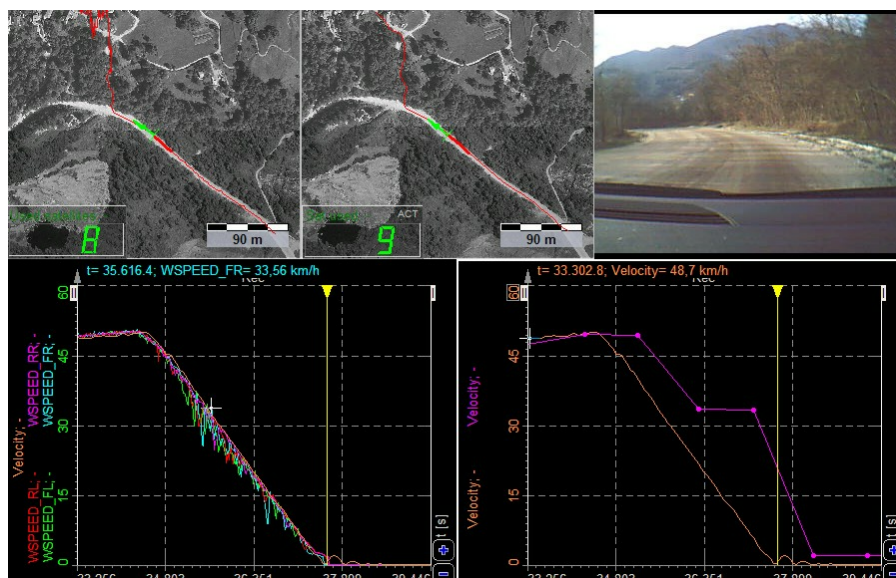


Let's zoom in on a certain part of the trip. We tried braking on a dirt road to observe the blocking of the wheels. Notice on the GPS maps that when we zoomed in, the map changed to the satellite picture. When we zoom in on the recorder, part of track is shown with a **thick** line while the other part is shown in **thin red**.

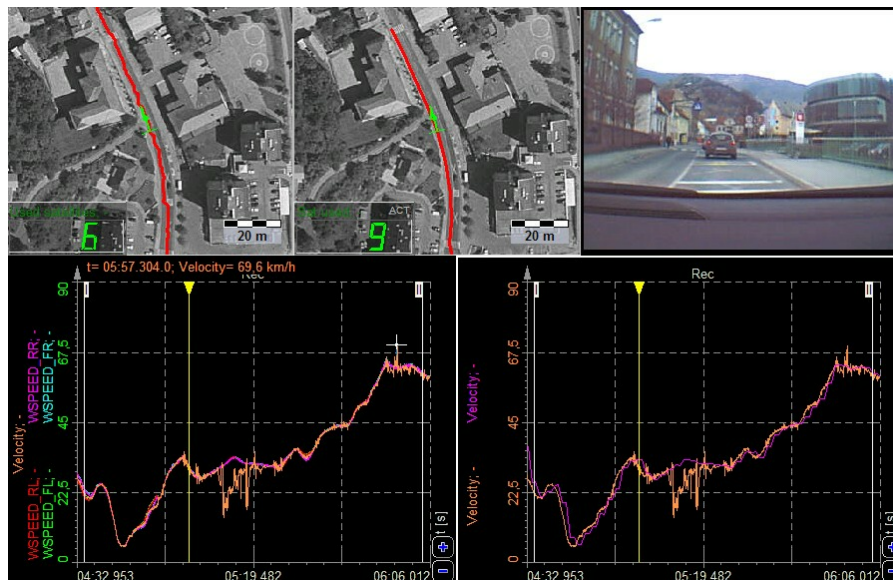
The braking reveals the difference between the high and low speed GPS. The lower right side *recorder* clearly shows half second *delay* from the real speed. This renders it useless for any evaluation purposes.

The **orange** curve is also shown on the left recorder, displaying that there is virtually *no delay* between the measured *wheel speed* and *GPS velocity*. However, since the wheels are locking up, the speed is dropping. But here we *see the action* of the *ABS system*. Without the ABS, the wheels would totally lock up leaving us without the control of the vehicle. At 50 km/h and on a wide road this wouldn't be a problem, but we can think of many different situations where this is not desirable. However, that graph also shows that we *can't trust* the measurements from the *wheels*, especially under such conditions.

The third thing worth noticing here is the *end of braking* when the vehicle stops. Since the car body moves on its suspension, we see a small jump at the end. This is where the vehicle moves slightly back and then settles down. If we measure brake distance, this is not wanted and *needs to be removed* from the calculation.



The next screenshot clearly shows the *effect of high surrounding buildings* where the high speed GPS velocity has dropouts which are surely wrong. The *low speed* GPS shows a *better* behavior in this case since it has much *more smoothing* in the calculations. The path also shows that the signal was disturbed.



So what would be the conclusion?

The *low speed* GPS is very useful when we have *slow changes*, like with trains, or if we just want a reference of *where* the vehicle was driving.

The *high speed* GPS is needed when we require *high precision* speed measurements, but we *need a clear sky view* to have *enough* satellites to be able to trust the measurements.

3 Power module tutorials

The **power module** is one of the most complex **mathematic** modules in **DEWESoft**. It allows measurements of different *frequency power grids* in different *configurations* and even *variable frequency* sources. This section will demonstrate how to use it.



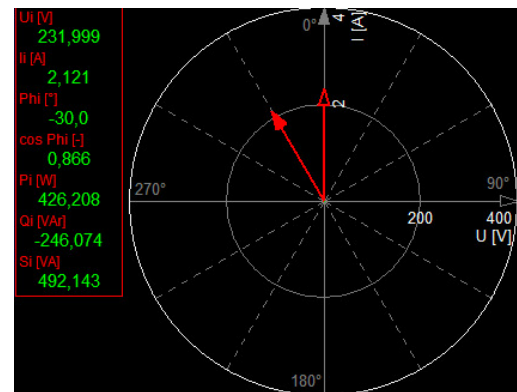
Single phase power

measures **voltage**, **current** and **phase relations** between them; calculates **harmonic** components, apparent, active and reactive **power**

3.1 Single phase power measurement

We have seen already example how to measure, store and trigger on voltage and current inputs in the "**Voltage and current**" examples. When measuring those two parameters, it is always interesting to also *measure* the **power consumed** or **produced** by the device being tested.

If we look to the **DEWESoft power** module, there are lots of parameters which can be **calculated**. Actually it is amazing *how many channels* can be produced from of just *two input* channels. First of all, let's take a look at the *common picture* from the power measurement. This view is called a **vector scope**. The name comes from the fact that *not only absolute* values of the voltage and current are important, it is the also very *important to know* the phase relations between them. Thus, the vector scope shows the *amplitude* of **voltage** (red arrow with **black** pointer) and **current** (red arrow with red pointer) as well as the *phase* in between them.



Why is the phase information so important? This is essential because the machine can *use only* the *part* of the current that is *in phase* with the **voltage** for producing the work. Therefore we measure the phase angle between the voltage and current as angle **phi** (in our case -30 deg), and from that also the **cos phi** (which is just a cosine of that angle, but it is very nice because it is *directly* the ratio of work against the *total consumed* current).

Thus we *define* the power:

$S=U \cdot I$ this is so called *apparent* or *consumed* power; in short, this is a power for which we pay the power distributor

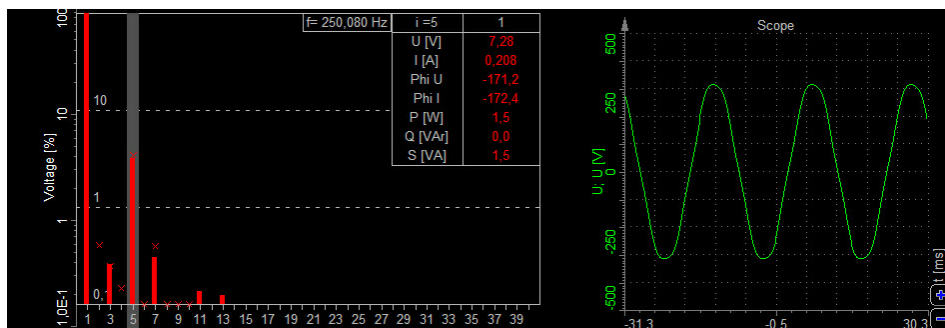
$P=U \cdot I \cdot \cos \phi$ *active* power, which is the power used for doing the active energy or work

$Q_i=U \cdot I \cdot \sin \phi$ so called *reactive* power; this is *wasted energy*, so we need to keep it low

Next, we can calculate the *harmonic components*. In theory, a *line voltage* is a perfect 50 (or 60) Hz sine wave. Since nothing in the world is perfect, the line voltage can have *distortions*, which are nicely shown as *harmonics* in the *frequency spectrum*.

The next picture shows the typical line voltage signal in the *scope*. We can already see in the scope that the voltage is not a pure sine wave. The *special* display called harmonic FFT nicely shows that the fifth harmonic is quite high - it has 7 volts amplitude on 220 volts of grid voltage. What is the effect of the harmonics? Imagine we have an AC *electro motor*. The *first* harmonic (*line frequency*) is *driving* the motor. The rest of the harmonics are producing *vibrations* and *noise*, but the ironic truth is that there are bad and even worse harmonics.

The 2nd, 5th, 8th... harmonics are really *bad* ones, since they are breaking the motor. The 3rd, 6th, 9th... harmonics are *either driving or braking*, while 4th, 7th, 10th... harmonic are *driving* the motor, but they are *still producing* higher *noise* and *vibrations*. Harmonics on the grid are produced either on the *generator* side or on the *consumer* side from for example switching power supplies or nonlinear devices like a transformer on or off.



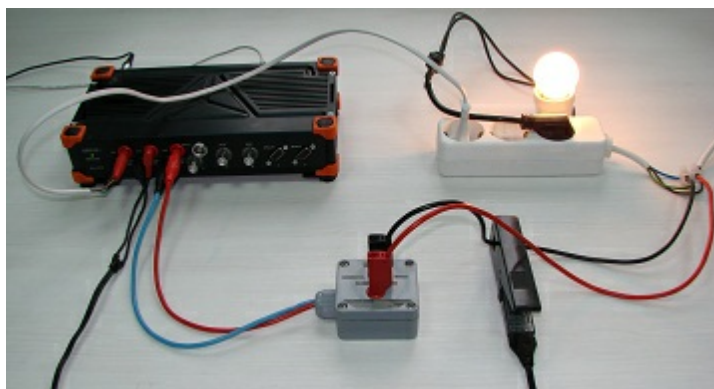
Actually we can calculate the apparent, active and reactive power for *each individual* harmonic. We can also calculate the interharmonics. Those are sine waves or other signals appearing in between the harmonic lines, which are *not covered* by the harmonic calculations.

There are also a few other parameters which that can be calculated. Further, we can calculate THD (total harmonic distortion or a *sum of all harmonic values*), period values (values for *voltage*, *current* and *power* for *each period* - this is very helpful for *triggering*) and the flicker (this is actually a *power quality parameter* measuring low frequency distortions of the voltage which tells us how much the lights are blinking).

3.1.1 Channel setup

| | |
|-------------------|-----------------------------------|
| Required hardware | Sirius HV isolated |
| Required software | SE or higher + POWER option or EE |
| Setup sample rate | At least 5 kHz |

So let's *measure* the line voltage and the current with a *current clamp*, *calculate the power* and *connect some loads* like our old *hair dryer*, a *classic light bulb* and an *energy saving light bulb*.



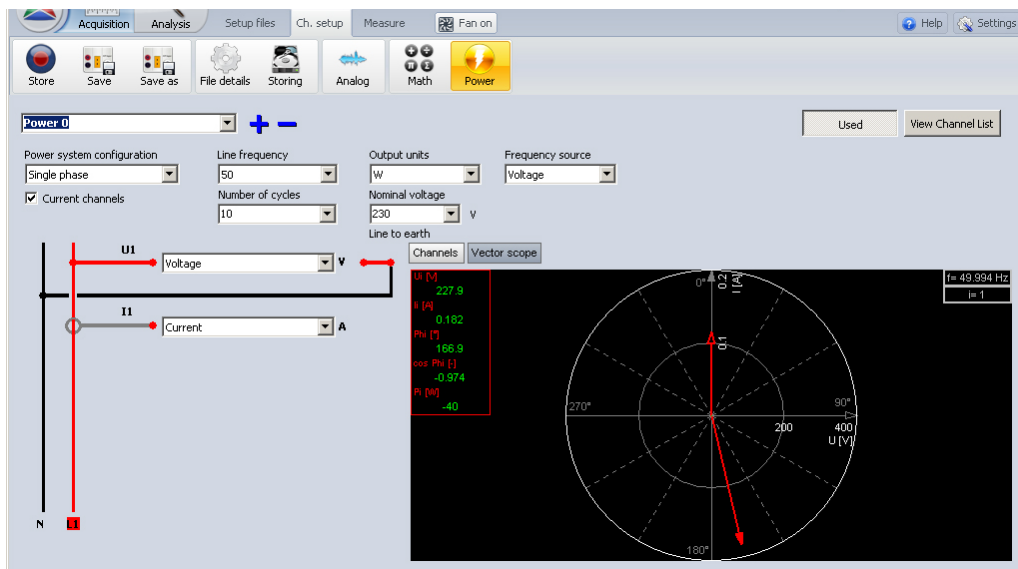
We will use the same **setup** for *analog channels* like in the "*Voltage and current*" example. So the *first* channel is the voltage while the *second* channel is the current.

| SLOT | ON/OFF | C | NAME | AMPLIFIER (002) | PHYSICAL VALUES | CAL | SETUP |
|------|--------|------|-------------------------|-----------------|-----------------|------|-----------|
| 1 | Used | AI 1 | SIRIUS-HV | SN: D0C81E14 | 0,00 V | Zero | Set ch. 1 |
| | | | Voltage, 1000 V; 50 kHz | | -1000 1000 | | |
| 2 | Unused | AI 2 | SIRIUS-HV | SN: D0C81D4C | 0,02 V | Zero | Set ch. 2 |
| | | | Voltage, 1000 V; 50 kHz | | -1000 1000 | | |

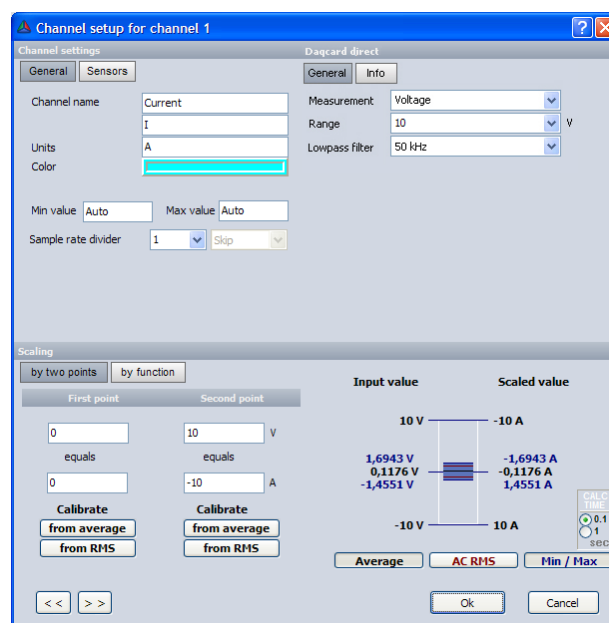
Note that we need to have the *correct unit* for voltage and current. Voltage units could be **V**, **kV** and **MV** while the current units could be either **A** or **kA**.

Now we go to the **Power** module tab and create one power module by clicking on the **+** button. First, we click on the **Power system configuration** drop down list, selecting the **Single phase** configuration and *defining* the voltage and current channels. We select **U** for voltage and **I** channel for current. Then we select the **Line frequency**. In Europe, the common line frequency is **50 Hz**, in the US **60 Hz** and in *large vehicles* like ships, it is common to have **400** or **800 Hz**. We can also measure the frequency inverters when selecting variable **Frequency source**.

Next, we need to check the *vector scope* to see if everything is connected correctly. It is right there on the setup screen. First thing to notice is that the voltage and current are *opposite*. Since we don't have a generator, but a hair dryer connected, we can assume that we are measuring the current in the *wrong direction*.



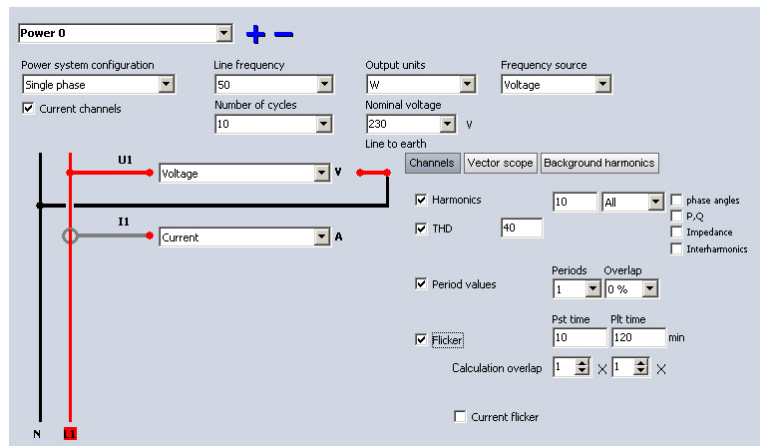
We can solve this by *reversing* the voltage plugs or *turning around* the current clamps, but there are cases where this is not as easy. If we have used current transformers which can't be opened, it is the best to *change the polarity* in the **software**. Let's go back to the setup for current. To *reverse the polarity* is easy - just enter a **negative "scaling factor"** instead of a positive one. So 10 V measured actually means -10 A on the *input* (and -10 V means +10 A).



Let's go back to make sure everything is ok. Now the phase angle is approximately 10 degrees, which seems reasonable.

Now let's see which channels are *available*. Basic parameters like *frequency* or *power channels* (P, Q, S, phi, cos phi) will be added automatically. So if we only want to measure basic power parameters, we don't have to select anything.

If we want to have *additional* parameters, we can choose to calculate the **Harmonics**, but even if we don't choose them, the values will be still *available* in the *harmonic FFT* and the *vector scope*. The only difference is that we can't, for example, show the 3rd harmonic of voltage in the recorder display. I have also chosen to select the **THD**, **Period values** and the **Flicker**.



Now let's make a measurement.

3.1.2 Measurement

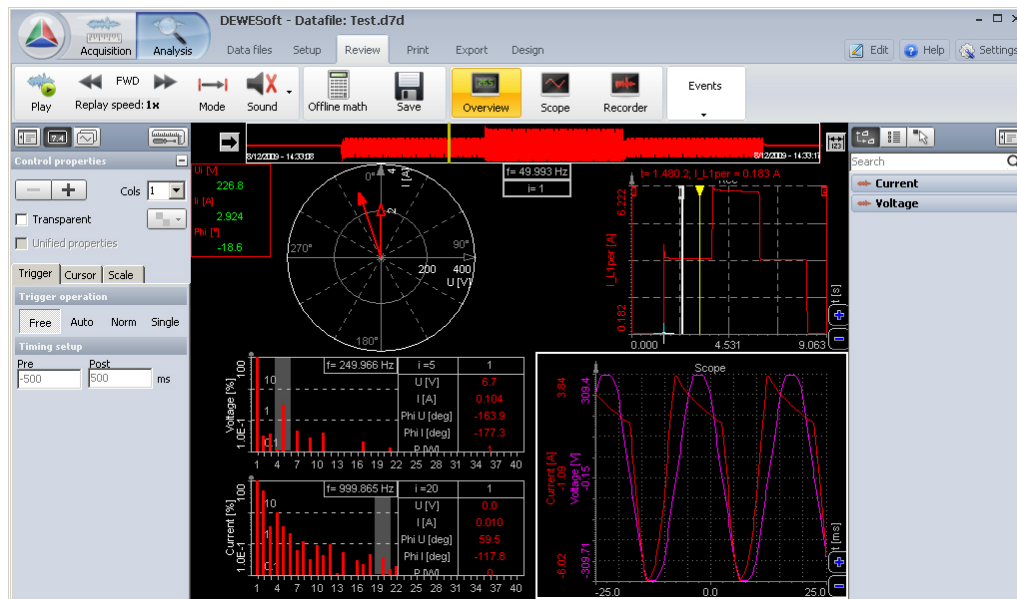
Let's go to the *overview* screen and **create a display** for power parameters. Additionally, there are two visual controls available in the *control tool* box, if we have at least one power module in the setup which includes *vector scope* and *harmonic FFT*.



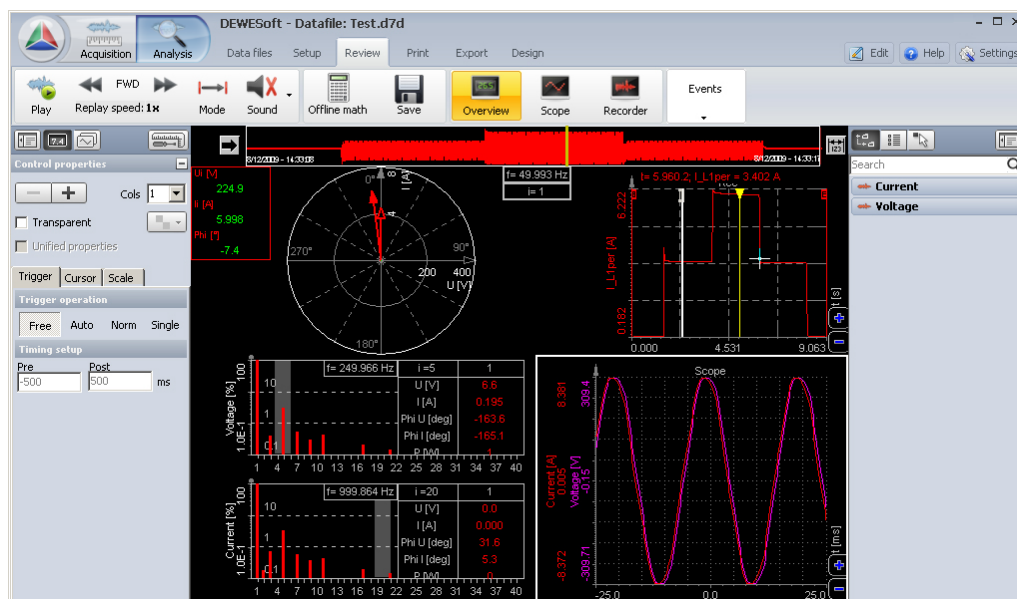
So let's place a *vector scope* at the top, which shows the *phase relations* between *voltage* and *current*. We can see there is a phase angle between voltage and current, since there is an electro motor inside the hair dryer. The *recorder* on the top right shows the *period value* of the *current*, showing how the hair dryer was *switched on* or *off*.

The bottom right *scope* shows the voltage and current and we can see nicely how the *hair dryer* is *only operating at half power*. The current on the lower part is cut because of regulations. We can see this behaviour even better with speed variable tools like a drill.

On the bottom left there are two *harmonic FFTs*, where the upper one shows the *voltage* and the lower one shows the *current harmonic*. We can observe nicely that the *5th harmonic* of the *voltage* has approximately 7% of the line voltage value, while the *current* has *many* harmonics, due to the regulation circuit.

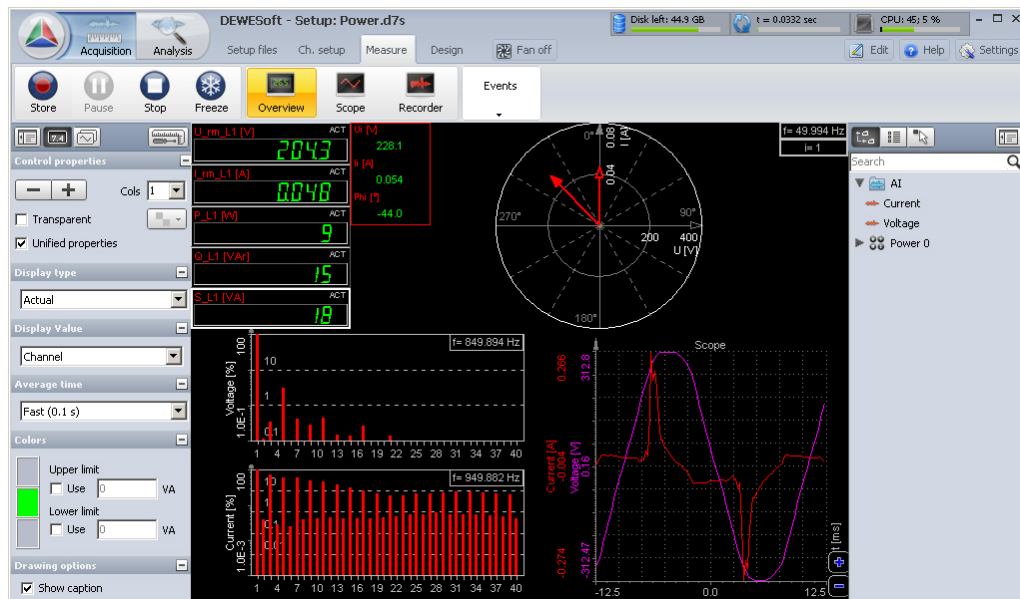


Now let's observe these characteristics *at full power*. The *current* is taken from the *full cycle* and the *phase angle* between voltage and current is much *lower*. Also, the *current harmonics* have *dropped* significantly and are *following the voltage* harmonics.

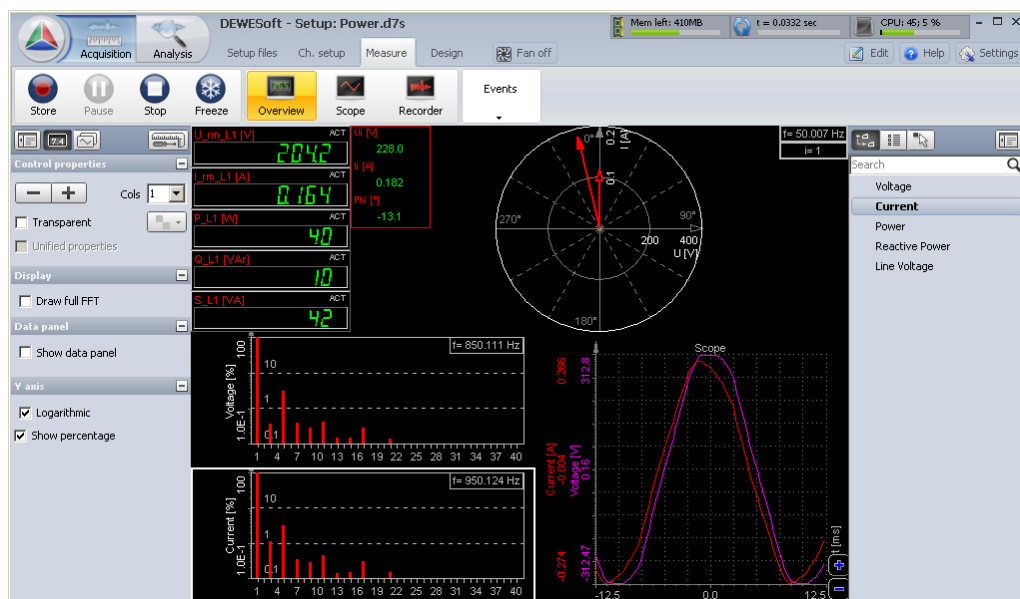


Now let's do make another test - observing the difference between a *normal* and an *energy saving light bulb*.

First let's connect the *energy saving* light bulb. I have used a 11 W light bulb, which should be similar to a 60 W normal light bulb. The total power is indeed very low, but the shape of the *current* is *not really* a sine wave. Therefore we have *lots of* harmonics of the current, which "pollutes" the *power grid*.



Now let's take a look at the *classic 40 W light bulb*. The first thing to notice is that the load on the grid is *linear to the voltage*. The measured power is exactly 40 W, but the *vector scope* looks strange. In fact, since the light bulb is a *purely ohmic load*, the voltage and current should be perfectly aligned, but as we can see, they are not. What is the reason for this? Remember the voltage and current tutorial where we have seen the difference between the current clamps and the shunt resistor? Since we are using the *current clamps*, we have *amplitude* and *phase errors*. As a result, the current clamp is main source of the calculation error in this case.

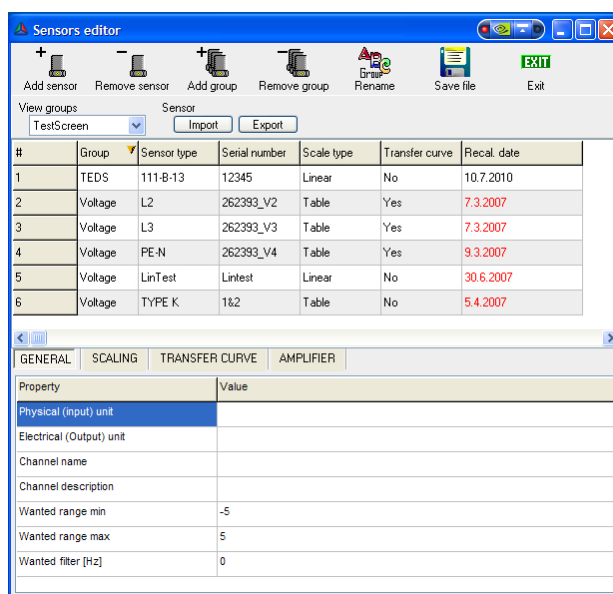


In **DEWESoft** we have a chance to *compensate* for these errors. Let's take a look how.

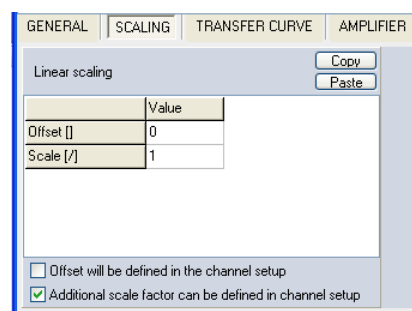
3.1.3 Sensor correction

Obviously we need to tell **DEWESoft** that we have a **sensor** which is *not perfect* and somehow enter the transfer curve of our sensor. A transfer curve gives information about *amplitude* and *phase* for sensors at certain frequencies, and from this information **DEWESoft** can **compensate** for these **errors**.

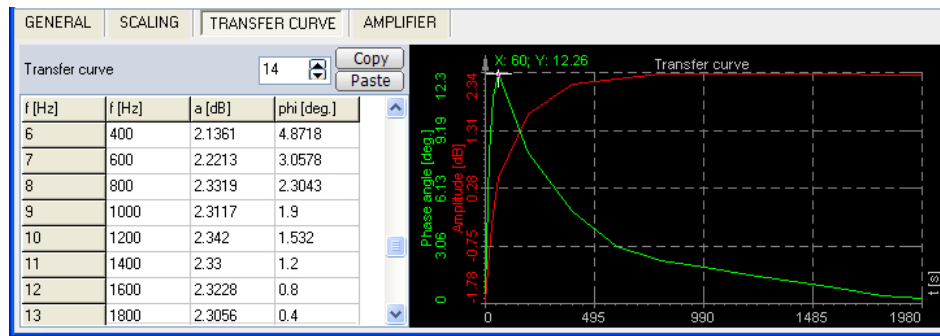
Let's choose the **Settings** → **Sensor editor...** menu item. We get the list of all possible sensors. Now let's **Add** one sensor and enter the **Sensor type** and **Serial number**. Choose the **GENERAL** tab, enter the **Physical (input) unit**, which is **A** (amperes) in our case and the **Electrical (Output) unit**, which is **V** (volts).



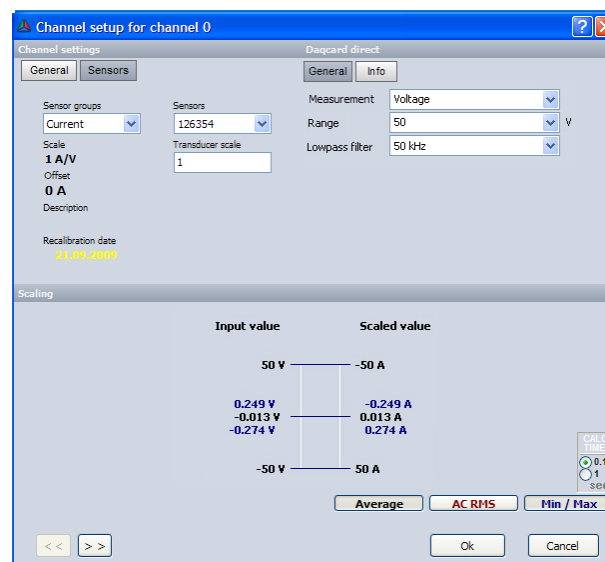
Next let's enter the **SCALING** factor. Since the sensor is linear with amplitude, we only need to enter the scaling factor, which is **1** in our case ($1A=1V$). We also select that we can define **Additional scale factor can be defined in channel setup** to be able to reverse sensor polarity.



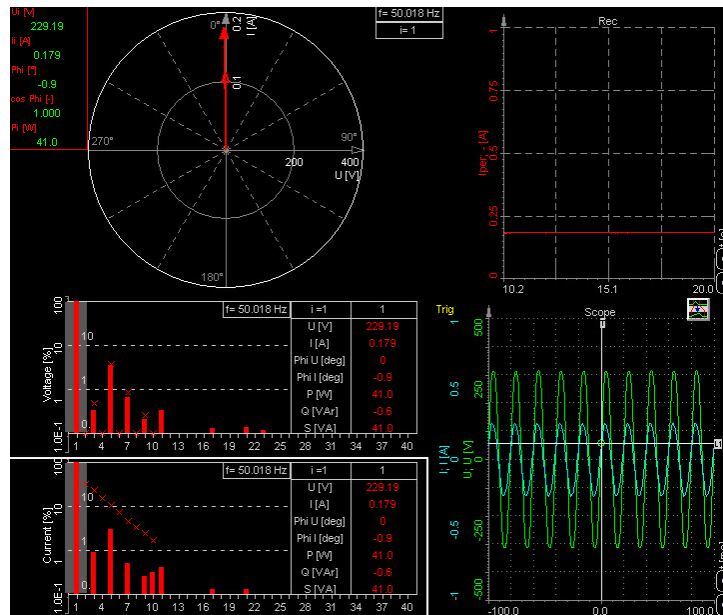
Now we came to the most important part - the definition of the **transfer curve**. In the table under the **TRANSFER CURVE** column we select **Yes** to signify that the transfer curve *will be defined*. Now we need to enter the points of the curve. We need to enter the **a[dB]** - amplitude deviation in **dB** and the **fi[deg]** - phase angle in **degrees**. The next question is: *where* do we to *get* this transfer curve? There are lots of transfer curves for the most common sensors, *already measured*, so it's worth asking if it already exists. A second option is to *copy it from the calibration sheet* of the sensor, if the cal sheet includes the transfer curve. The third option is to *measure it* with **FRF option**, but this requires some equipment. When we get this transfer, we need to enter it in the table. We see that at 50 Hz, the angle was around 10 deg, which could explain phase shift we saw in the measurement.



Save the sensors with the **Save file** button and close the sensor editor with **Exit**. Now let's go back to the **analog setup** and choose the *sensor* for the *current channel*. Open the **Sensors** tab and select the *serial number* of the sensor previously entered in the **Sensor** field of the editor. Nothing much happens, but note that we *can't enter the normal scaling or sensitivity* anymore. We only have a choice to enter a **Transducer scale**, which we can use for *reversing the polarity* of the sensor by entering a value of **-1**.



That's it. For the next setup we don't have to define a sensor anymore, instead we can *just select it from the sensors list*. Now let's see what the effect on our measurement is. The results are *much* better. The phase angle is virtually eliminated and the power is calculated correctly.







From this example we can clearly see how *big the errors could be* if the sensor characteristics are not taken into account for power measurements.

4 Dynamic signal analysis tutorials

Dynamic signal analysis covers a wide range of measurements in the field of **structural dynamics**, **industrial acoustics** and **machine diagnostics**.

The frequency response function measurement is a part of a different manual - the [FRF user's guide](#).

| | | |
|---|---------------------|--|
|  | Sound level | explains procedures for working with Sound level module, which is used to calculate levels of sound with time and frequency weighting |
|  | Torsional vibration | explains procedures for working with Torsional vibration module used to measure dynamic and static bending and vibration of the <i>shafts</i> |
|  | Human vibration | explains procedures for working with Human vibration module, used to evaluate effects of vibration on the <i>human body</i> |
|  | Order tracking | explains procedures for working with Order tracking module used to extract the harmonics during <i>machine run ups</i> and <i>run downs</i> |

4.1 Sound level

The sound level **Math** section allows **calculating** the typical parameters for sound level measurements from a *single microphone*. It allows **DEWESoft** to be used as the typical *sound level meter*. With appropriate hardware (Sirius ACC) it can easily fulfill all the requirements for a **Class I** sound level meter.

| | |
|--------------------------|---|
| <i>Required hardware</i> | Sirius ACC, MULTI, STG, DEWE-43 with MSI-BR-ACC |
| <i>Required software</i> | SE or higher + SNDLVL option, DSA or EE |
| <i>Setup sample rate</i> | At least 10 kHz |


Sound level measurements are available *selecting* the **Sound level meter** checkbox (Settings → **Hardware setup...** → **Math** tab). After selecting this option, a tab labeled **Sound levels** appears in the **DEWESoft Setup screen**.

Basic procedures of **Sound level** measurement are:

- **Channel setup** for applied hardware
- **Microphone calibration**
- **Measurement**

4.1.1 Channel setup

To use sound level module, please first *select* one or more *analog channels* in the **Analog** tab to measure the sound.

Then *switch* to the **Sound levels** tab and click the  button to add a *new Sound levels module*. Several modules can be used within a session.

A screen like the one shown below will appear. First of all, *select* the *input channels* ① that should be measured at the upper left side of the display. In our case, only **AI 0** is selected. We can select *multiple* channels and then have several *output channels* with the same settings.

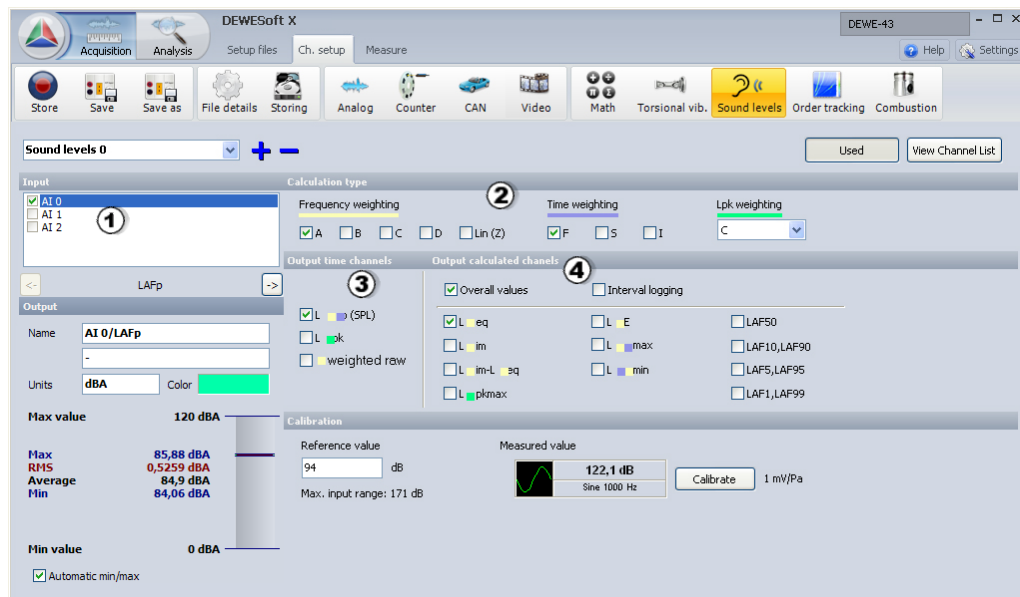
We have several options to select from in **Calculation type** section ②. Any combination of **Frequency** and **Time weighting** can be selected. We can also select the weighting specifically for **Lpk weighting** from **linear**, **A** and **C**.

We have three *types* of **Output time channels** ③. First is the **L_{FTP}** - *time* and *frequency* weighted sound pressure level already scaled to **dB**. Then we have the **L_{pk}** value, which shows the *current maximum value* of the sound levels, the **L_F weighted raw** value shows the *frequency weighted time curve* of sound in **Pascal**.

Then we need to select **Output calculated channels** (parameters) ④. These parameters can be **Overall values**, which means that we have only *one* value at the end of the measurement and/or interval logged. If we have an *interval logged* value, the *time interval* for logging is defined. For example, if we select to have **Interval logging** for **Leq** with **5 seconds** intervals, we will get a new value of **Leq** *after each* 5 second period. After that, the value is *reset* and the calculation is *restarted*.

The values that can be calculated are:

- *frequency-weighted* **L_{Feq}** value, which is equivalent to sound level
- **L_{Fim}** tells the *impulsivity* of sound and is the *impulse-weighted* equivalent; the difference between those two values is *calculated* as **L_{Fim}-L_{Feq}**
- **L_{pkmax}** is either **C** or a *linear* weighted maximum *peak* value of sound
- **L_{FE}** is the *frequency-weighted* sound *energy*
- **L_{FTmax}** and **L_{FTmin}** are the *time-* and *frequency-weighted* *minimum* and *maximum levels* of sound *pressure*
- the next section of sound is *classified sound levels*. Each *calculated* value is put in these *classes* and then we can choose to see **LAF1**, **5**, **10**, **50**, **90**, **95** and **99** % classes of values.



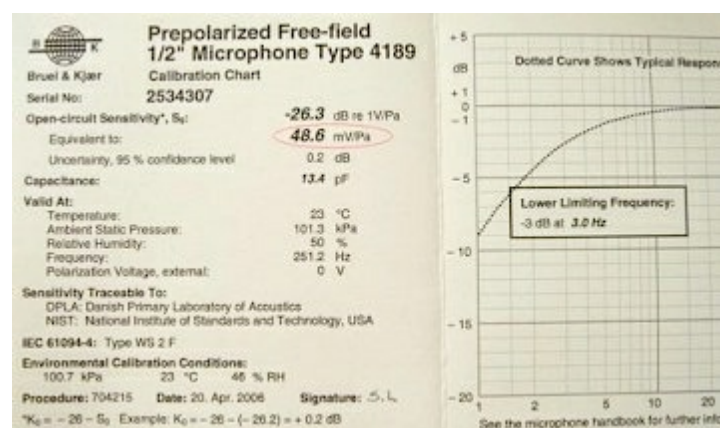
Now let's see how to **calibrate a microphone**.

4.1.2 Microphone calibration

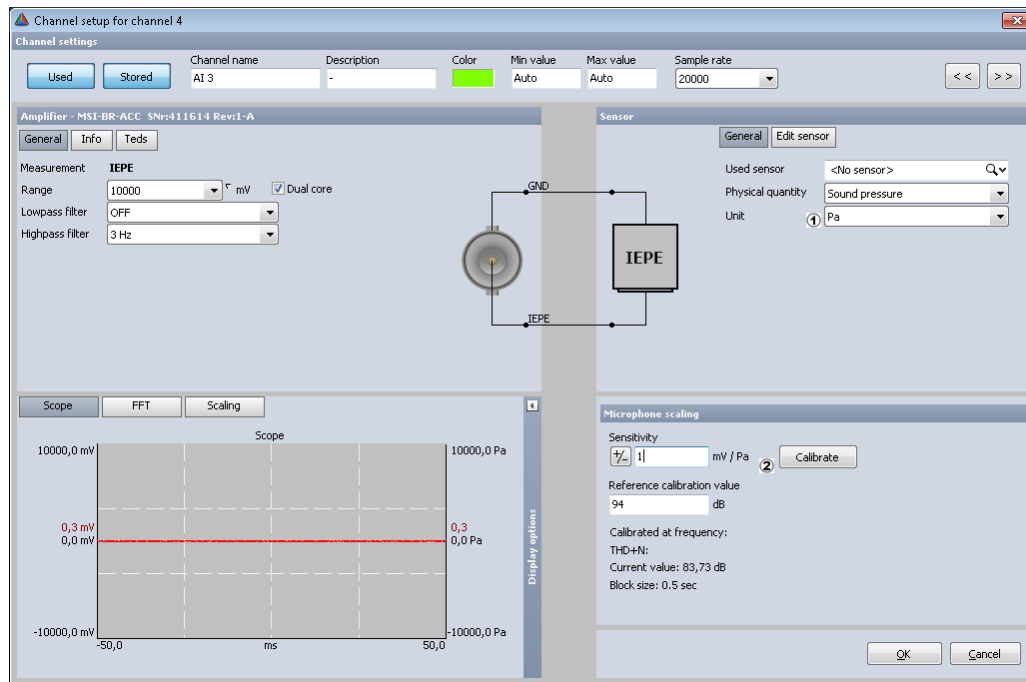
Microphones can be **calibrated** in two ways. First, we have to know that the *direct value* of measurement from the microphone is the *sound level* in Pa. Therefore, we need to *scale* it to the physical quantity.

Scaling with calibration certificate

If we don't use the calibrator, but have the *sensitivity* of microphones, we can define it *directly* in the channel setup.



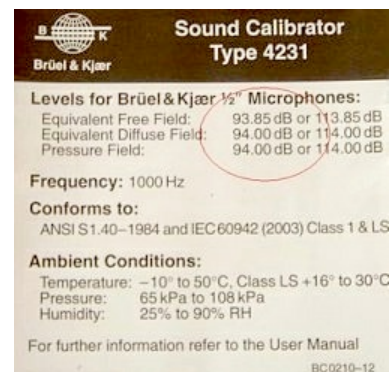
First, Pa is *defined* as the *physical Unit* ① of measurement. Next, we go to **Scaling by function**, check the **Sensitivity** and enter the value in V/Pa ②, which can be found on the *calibration certificate* of the microphone.



Calibrating the microphone with calibrator

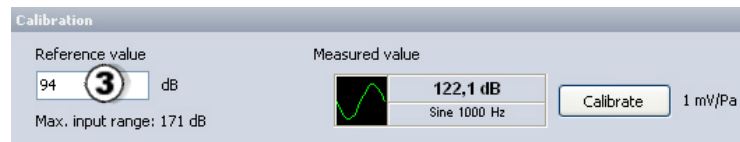
Another way is to calibrate the microphone with the **calibrator**. In this case, the *known* parameter is the *sound level* emitted by the calibrator.

In our case it is 94 dB.



This value is entered *directly* in the **Reference value** ③ field of the **sound level** module channel setup. Then we *connect* the calibrator to the microphone, and turn it on. We can see the signal directly in the small overview. In our case, it should be a sine wave with a *frequency* of **1000 Hz**. Since all the frequency weighted curves are referenced to 1000 Hz, this is a *very usual* frequency for calibrating microphones.

After we see that the sound is correctly *recognized* as the sine wave at 1000 Hz, we can click the **Calibrate** button to perform a calibration. The **sound module** will *calculate* the *sensitivity* of microphone *from* the *highest FFT amplitude* and *reference value*.

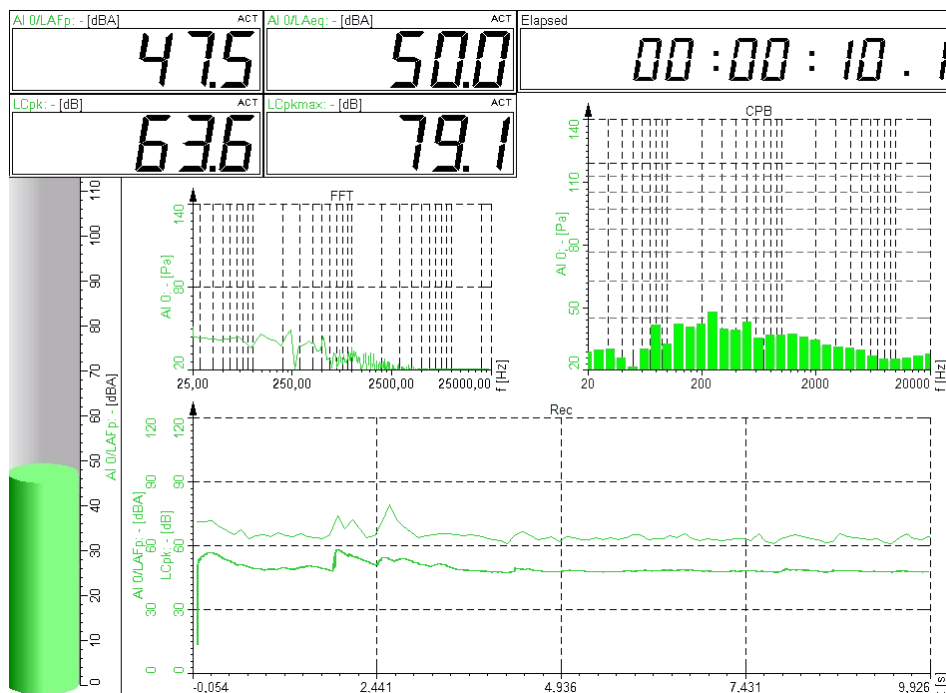


The *sensitivity* will already be directly *corrected* in the *source channel* and therefore no additional *analog scaling* is necessary. We can directly check the *calibrated sensitivity* with the information found on the *calibration certificate*.

4.1.3 Measurement

The **sound module** does not automatically create a *display*. The users has to *create* the **setups** to show the *sound level channels*. The display below shows typical screen for sound measurement.

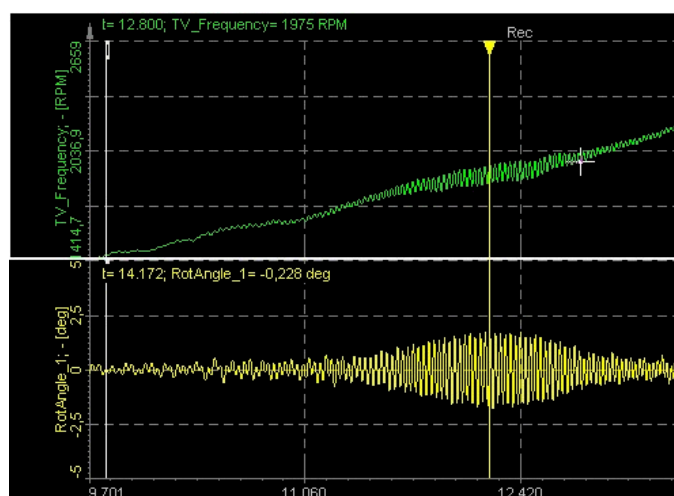
We use the typical **A** weighting to calculate **Leq** and **Lp**, which are shown in the *multimeter*, *bar display* and *recorder*. Additionally, the *octave* or *narrow band* FFT is calculated. This can be achieved by *adding* an *FFT screen* and in this screen where we can define *frequency weighting*, such as **A** weighting and **dB** scaling.



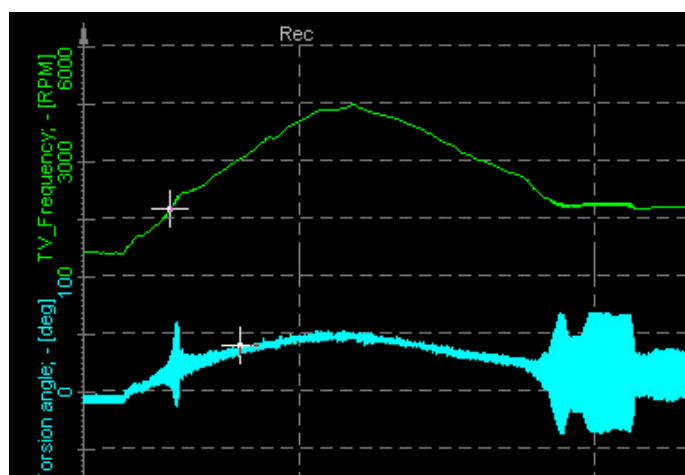
4.2 Torsional vibration

It is important to know that we actually have two different *parameters* that can be measured with the **torsional vibration** module: rotational vibration and torsional vibration. What is the difference between them?

Rotational vibration is simply the dynamic deviation of the rotation speed. If we measure the rotation speed of the shaft with high precision, we will notice that we get a high deviation of rotation speed in some regions of the run up. This is caused by the angular vibration crossing the angular natural frequency of the shaft. It is calculated by cutting off the DC component of the rotation speed or rotation angle. We can see an enlarged section of the run up in the graphs below. The high deviation of this frequency can be seen, while the yellow curve shows the deviation in degrees, which is actually the rotation angle vibration.

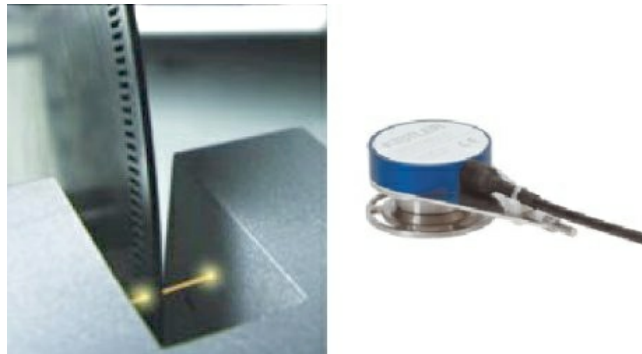


Torsional vibration is an oscillation of angular motions (twist) which occur along rotating parts, such as gear trains, crank shafts or clutches. We need two encoders to measure the torsional vibration, so the torsional vibration is actually a difference between angles of the two encoders. The torsional vibration also measures the static twist of the shaft with higher RPMs. The graph below shows the run up and coast down where we can nicely see the static twist of the shaft, and when passing natural frequency, two high angular vibration of the shaft.



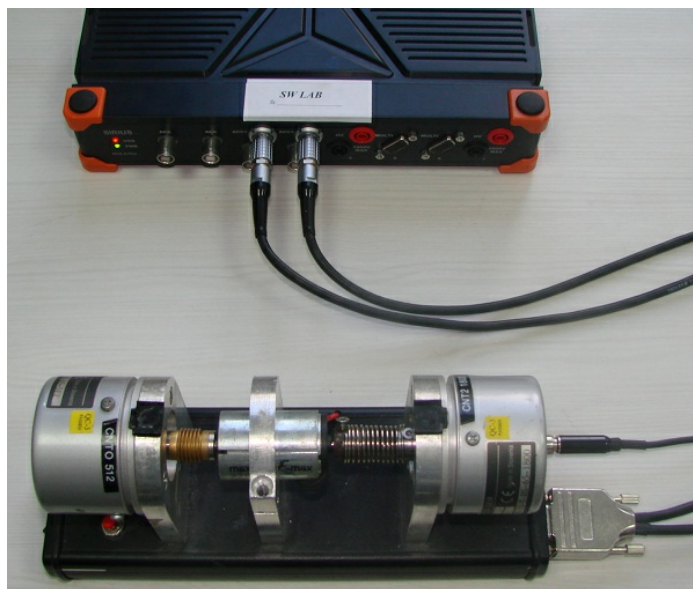
| | |
|--------------------------|--|
| <i>Required hardware</i> | Sirius ACC+, MULTI or DEWE-43 |
| <i>Required software</i> | SE or higher + TORVIB option (if order extraction is required, also ORDTR option is needed), DSA or EE |
| <i>Setup sample rate</i> | At least 10 kHz |

Torsional and rotational vibration can be measured with either an **encoder** (up to 3600 pulses per revolution) or a *special sensor* that has less resolution (up to 720 pulses per revolution) but is much less sensitive to vibrations that could damage standard encoders.



4.2.1 Rotational vibration setup

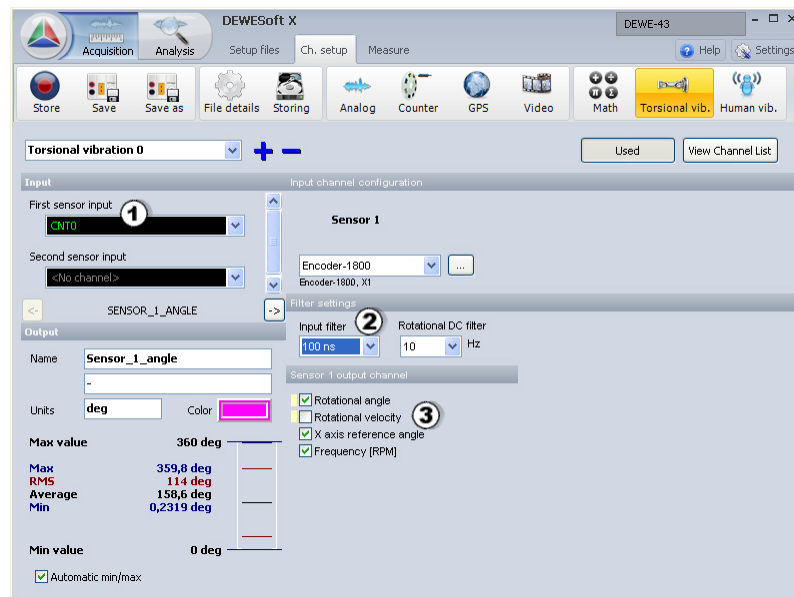
For our test, we will take our test electro motor where two encoders are mounted on each side with a spring to produce high torsional vibration. Both **encoders** are connected to the counter input. First let's do the rotational vibration.



We don't need to set anything in the *analog* or *counter channels*. Let's just go to the **Torsional vibration** tab where we can **add** the new module by clicking the **+** button. Next we select the **First sensor input** ①. Since we have connected the first sensor to **CNT0**, we need to select it from the list.

Then we define the **Sensor**. In our case we have **512** pulses per revolution, so we choose **Encoder-512**. If the sensor used is not defined so far, we need to create it in the **Counter sensor editor** by clicking the three ellipsis button the right to

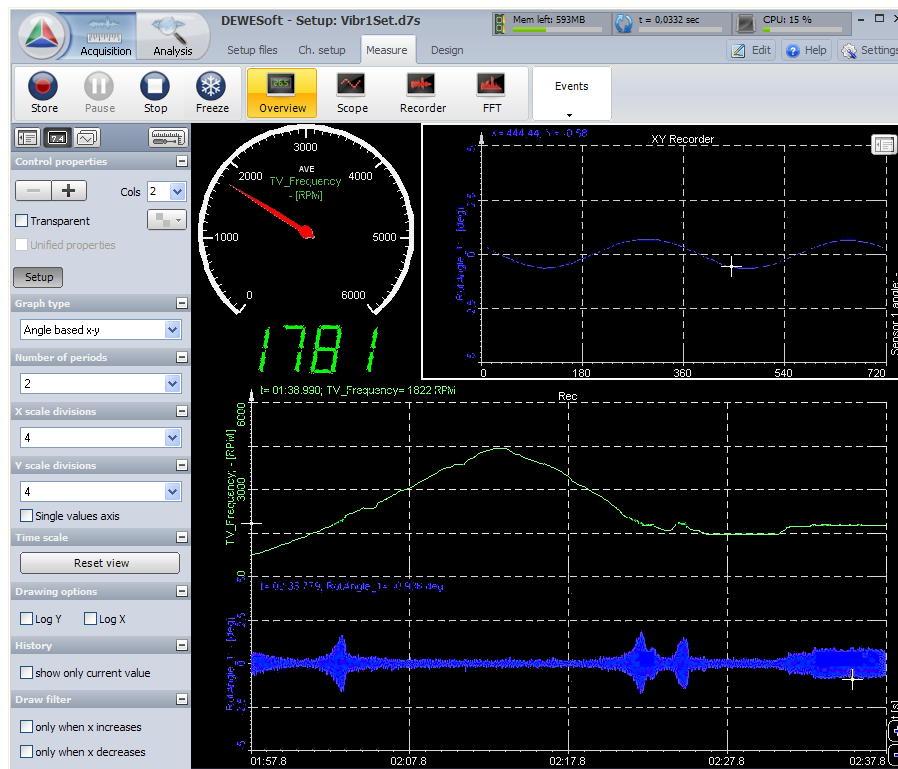
the sensor drop down menu. Next we need to set the **Input filter** for the counters. The input filter is needed *to prevent* glitches and spikes in the signal. The **Rotational DC filter** needs to be set to cut the DC component out of the RPMs. We need to set the filter to *include all wanted* frequencies, but not too low, else we will have static DC deviations on the output signal ②.



The output channels are **Rotational angle** (filtered angle value of vibration), **Rotational velocity** (filtered velocity vibration value), **X axis reference angle** (the reference angle, which is always from 0 to 360 and can be used as a reference in angle based xy diagrams) and **Frequency [RPM]** in unit of RPM ③.

4.2.2 Rotational vibration measurement

The torsional vibration module doesn't have any pre-set displays, so we need to *add* them on our own. I have added *analog* and *digital meter* for frequency as well as *recorder* for frequency and rotational angle. On the upper right side, we can see the *xy recorder* running in a special mode called **Angle based x-y**. For the **x axis**, we need to select the **Sensor 1 Angle** (a reference angle from 0 to 360 degree indicating the current shaft angle) and the *rotational angle* for the **y axis**. This *xy recorder* now displays the rotational angle of the *current* revolution. It is like a scope, but with an angle reference instead of a time reference.



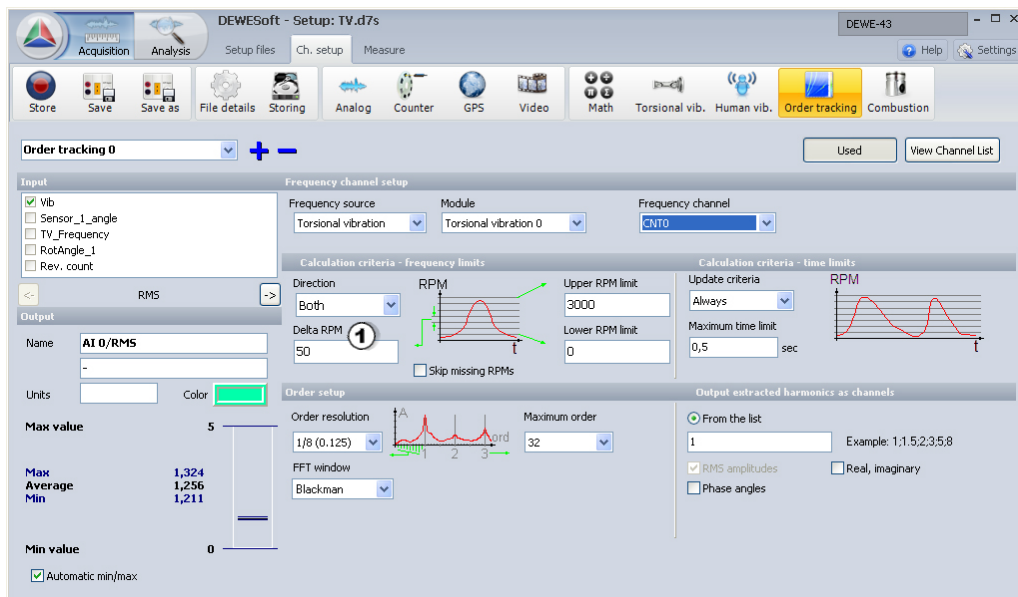
On the lower recorder we can see the *natural frequency* region around 1800 RPMs. It would be nice to also see the extracted orders of this data. To do this, we need to use the order extraction module.

4.2.3 Rotational order extraction

To **extract orders** from a rotational vibration, we need to add the **order tracking** module. For the frequency source, we need to define the Torsional vibration module. For this, you select the module and *Frequency channel* (within the module if torsional vibration is used).

For this, you select the module and *Frequency channel* (within the module if torsional vibration is used). Next we need to define the **Upper RPM limit** and **Lower RPM limit**. This is used to reserve the memory for waterfall FFT. The waterfall will be drawn from the lower to the upper limit in the **Delta RPM** step ①. In this case, we will have $(3000-0)/50=60$ steps of waterfall.

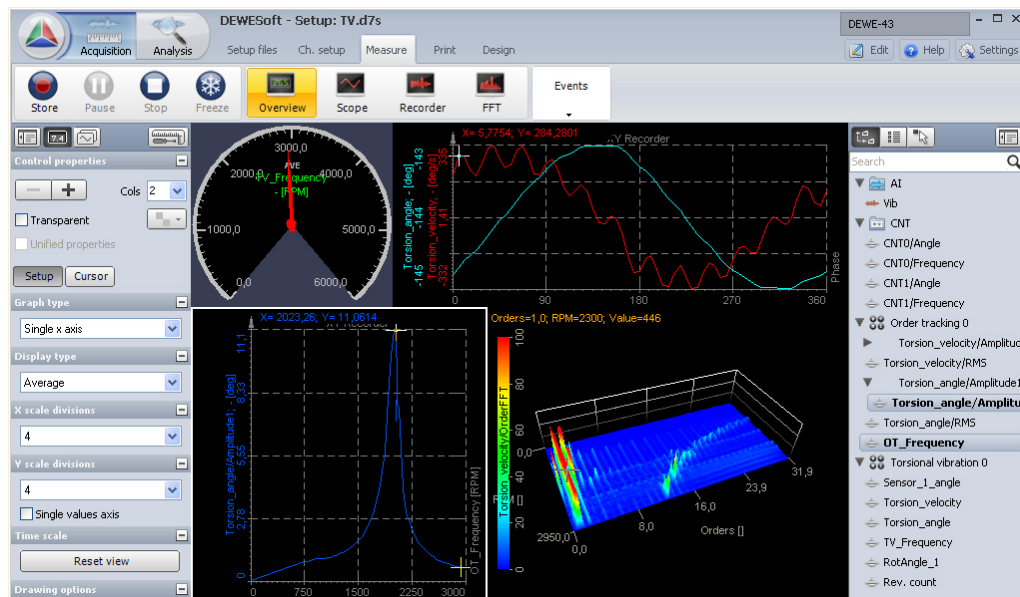
We choose to extract *first three* orders.



Now let's do some measurements. We have frequency and harmonic channels, therefore we can add the *x-y display* for displaying the orders. The *OT_Frequency* should be used as *x* reference (first chosen channel of x-y), and the *H_1*, *H_2* and *H_3* orders should be used as the *y* axis. Then we make a run down to observe the orders, where we can clearly see the *peak* at 1800 RPMs that we previously observed in the recorder being the first order of vibrations.

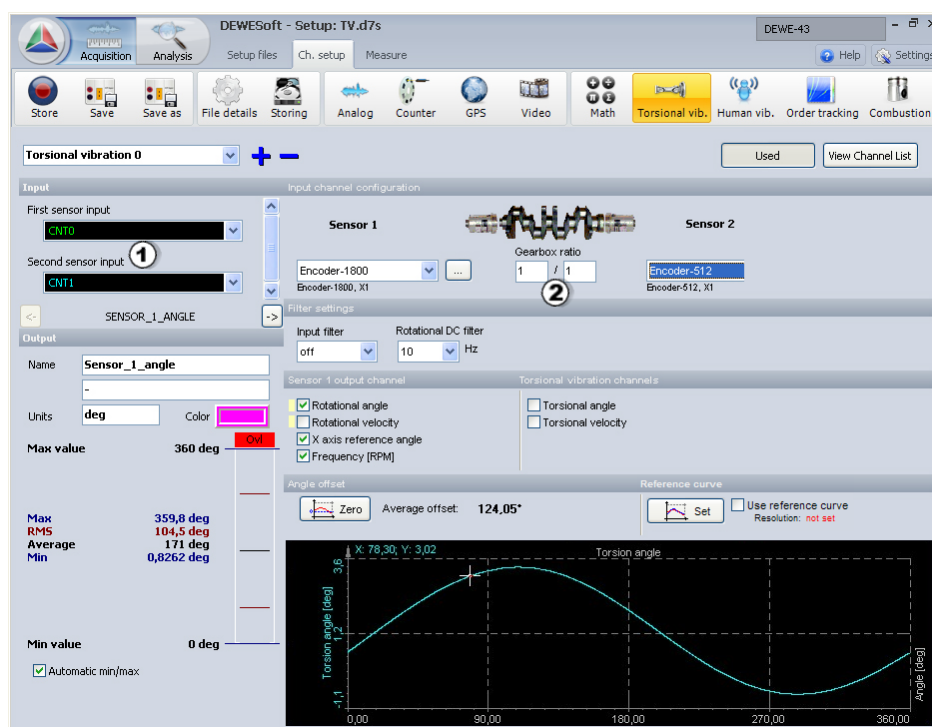


To see it even better, we can add the 3D graph. This graph can only show matrix channels. Order tracking has two of them - *OrderFFT* and *TimeFFT*, which can be used as the data sources. Now we can start our run up or run down. Since the color shows the amplitude at a certain frequency and RPMs, we need to carefully choose the *amplitude scale* to show the graph in a nice way. To see also the small amplitude values, the *logarithmic* scale is recommended.



4.2.4 Torsional vibration setup

The next step is to **measure** the torsional vibration. To do this, we need to **select** both input channels in the **Torsional vibration** setup. We select **CNT0** as the first channel and **CNT2** ① as the second channel. Then we need to define **Sensor 1** and **Sensor 2**. If we have a gearbox in between, we need to enter the **Gearbox ratio** ②.



The **Input filter** can be set in a range between **100ns** and **5μs**. The optimal settings are derived from the following equation:

$$InputFilter\ r[s] \leq \frac{10}{RPM_{MAX} \cdot PPR}$$

RPM_{MAX} max. revolutions per minute [s^{-1}]

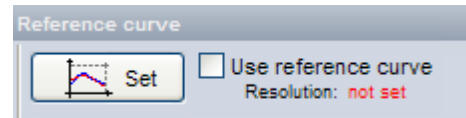
PPR pulses per revolution [-]

Angle offset / Reference curve configuration

A click on the **Zero** button **removes** the *angular difference* (offset) between the two **sensors** (set angle offset to 0).



A click on the **Set** button *records* the *current torsion angle* over one revolution as reference. When **Use reference curve** is checked, the recorded reference is *subtracted* in the angle domain from the *current* torsion angle! Thus, torsion errors caused by the sensors or their fixing can be overcome!



There are two additional channels available:

- **Torsional angle**, which is the dynamic torsional angle that is the *angle difference* from sensor 1 to sensor 2
- **Torsional velocity** is the *difference in angular velocity* from sensor 1 to sensor 2.

4.2.5 Torsional vibration measurement

Now let's do the measurement. I have added the *analog* and *digital meter* for *frequency*, *recorder* for *frequency*, *torsional angle* and *torsional velocity* and have also added *angle* based *xy* for those parameters.

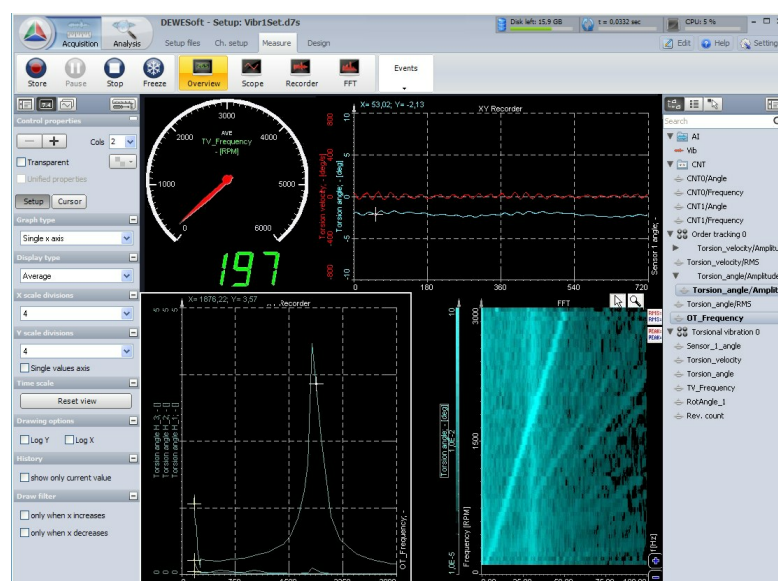
The *run up* and *run down* clearly shows the static torsion angle and region of natural frequency with high torsional vibration values.



Here we can get better results by using order tracking.

4.2.6 Torsional order extraction

Basically we do the same setup as for rotational vibration, except this time we choose the **Torsion angle** ① or **Torsion velocity** (or both) for the *input channel* of the order tracking **math** module. The *waterfall FFT* clearly shows the natural frequency which is confirmed by the orders shown in the *XY recorder*.



4.3 Human vibration

Human vibration is a measurement of **effect of vibrations** to human body. Especially at work places exposed to vibrations there is a big likelihood of permanent damage to some parts of the human body. One effect is known as *Raynaud's disease* or the effect of *white fingers* where the fingers change color to white and become painful. Another typical effect of working with *heavy machinery* or vehicles (a typical example is the *helicopter*) is the problems with the lumbar region.

The **human vibration module** provides measurements to be able to **judge the risk** of such damage. It is based on an [ISO 2631-1](#) (dated in 1997) standard which defines *basic* procedures, [ISO 8041](#) (dated 2005), which defines *exact* procedures for measurements and [ISO 2631-5](#) (dated 2005) which defines *calculations* of lumbar spine response to the vibrations.



There are two main types of measurements: *whole body* and *hand arm*. Both measurements are performed with *triaxial accelerometers* (it is very common to use [50 g sensors](#)) and using special adapters. For work places with *high vibrations* (for example impact hammers) it is necessary to use [high g sensors \(500 g or more\)](#). This sensor should also survive high shock.

For the measurement we need several *ICP channels* with a *24 bit sigma-delta AD card* (Sirius or Dewe-43, for example).

Whole body measurements

Whole body vibrations are measured with the help of the so called *seat sensor*, where we need to install the *triaxial sensor* in the *rubber adapter* on which we sit. It is important that the **z** axis is in a *vertical* direction, since it is weighted differently than **x** and **y**.



Hand arm measurements


The second application is the measurement of hand-arm where the *sensors* are installed on *special adapters* for holding them on a handle or between fingers. The orientation of the sensor is not important in this case since all three axes have the same weighting.

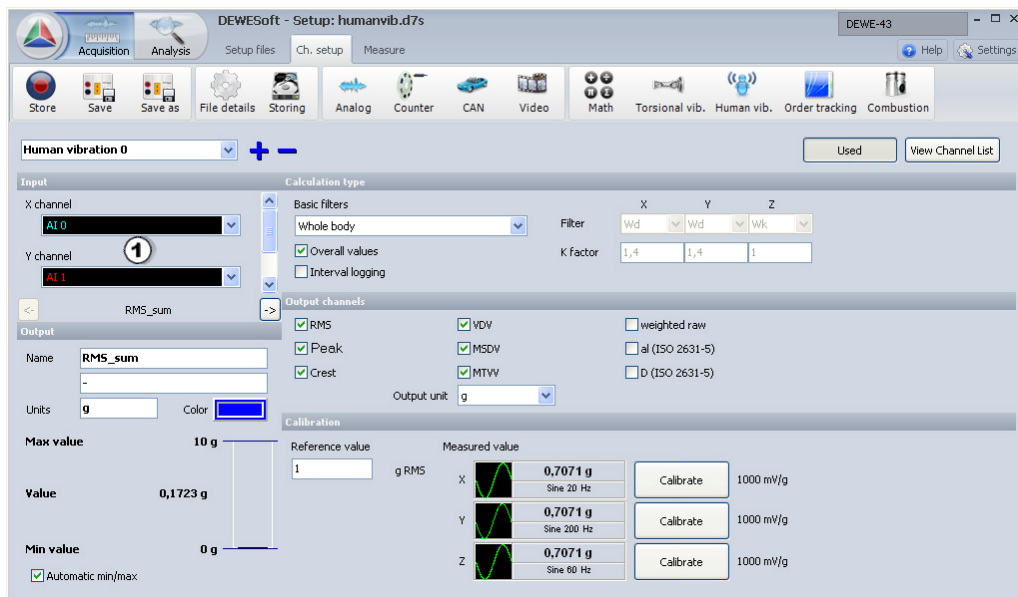


In theory, we would need to measure a *full working day* with all the *significant loads*. Often the measurement interval is shorter, but we need to ensure that all the significant vibration patterns are *covered correctly* in the obtained measurements.

4.3.1 Input channel setup

| | |
|--------------------------|--------------------------------------|
| <i>Required hardware</i> | Sirius ACC, MULTI, DEWE-43 |
| <i>Required software</i> | SE or higher + HBV option, DSA or EE |
| <i>Setup sample rate</i> | At least 5 kHz |

To use the human vibration module, please first select at least *three vibration analog channels* in the **Analog** tab. Then *switch* to the **Human vibration** tab and click the  button to add a *new Human vibration module*. *Several* modules can be used within a setup, and we will need three channels for *each* module.



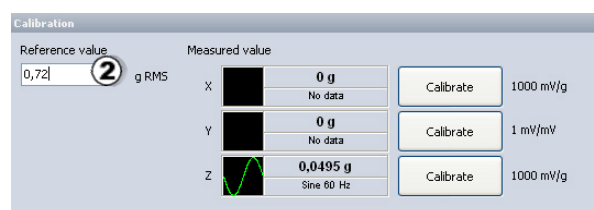
If we have a calibration sheet for the sensors, enter the **Sensitivity** in the **analog input setup screen** (see the **Vibration measurement** tutorial for details on how to do sensor calibration in the **analog Channel setup**).

The next step is to **assign them in the Input section ①** of the **Human vibration module**. We should then already see our live values in the calibration part of the screen.

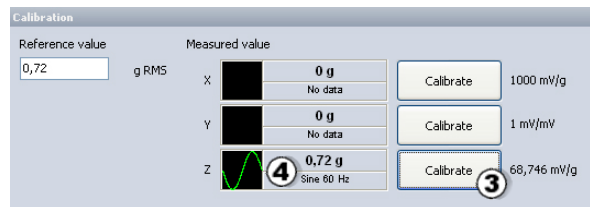
Calibration

If we want to perform a calibration with the *calibrator*, we can perform it here - in the **Human vibration** module itself. First enter the **Reference value ②** of the vibration. This can be read from the calibrator. Usually (let's take this as an example), the calibration *levels* are 10 m/s² peak, which is 7,07 m/s² RMS or **0,72 g**. We need to enter this value in the **Reference value** field.

As soon as we mount the *sensor* on the calibrator, we should see the **amplitude** and the **frequency** of the *signals*. In this case we see that we applied the sensor in the **Z** direction. The frequency of the calibration is below 200 Hz (**80** or **160** is typical, in our case it is **80 Hz**). This is a simple to make sure that everything is working correctly.



The next simple step is to click the **Calibrate ③** button near the axis that is *currently* calibrated. As soon as we do this, we will see *sensor sensitivity* in **mV/g**, which can be checked against the sensor calibration certificate. A small percentage of difference is acceptable.



Now we can see the calibrated live RMS value of the vibration (0,72 g) ④.

4.3.2 Output channel setup

Measurement modes

The next step is to define the measurement. There are two basic modes of operation. First is the **Whole body** mode and the second one is **Hand arm** mode.

Different modes define different **Basic filters** ① used to *simulate human response* to the vibrations. Those filters are defined from numerous measurements of natural frequencies of certain parts of the human body.

We can also use the **Linear** filter to check the measurement chain or the **Custom** filters.

If we use *custom* filters, the individual **Filter** ② can be chosen from the list on the *right* side to do *special* measurements (for example *building vibrations* or *sea sickness*).

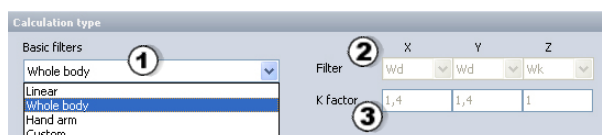
I would like to point out that we need to keep in mind the *high pass frequency limit* of the *sensor* and the *amplifier* used. For hand arm mode, the *high pass frequency* is **6.4 Hz**, which is easy for any sensor. For the whole body the *frequency limit* it is **0,4 Hz**, where we need to choose the sensor carefully. We can also use *higher* filters (like 3 Hz), if we know there is no frequency content below this limit. This will help to perform a measurement faster and with less error (lower frequency filters means longer settling times).

The recommended **sampling rate** of the measurement also depends on the application. For *hand arm*, the minimum sampling rate is **5 kHz**, while for all the others **1 kHz** is enough.

Special attention must be paid to the **Wf** filter for *motion sickness* (for example on ships) where the *frequency limit* is only

0,08 Hz. We need a *very special* sensor to measure this.

With a custom filter, we also need to define a **weighting K factor** ③. This is a *multiplication factor* for each axis when calculating the vibration sum.



Calculated parameters

Next we need to select the calculated parameters on **Calculate type** section. These parameters can be either **Overall**

values, which mean that we have only one value at the end of the a measurement, and/or **interval logged values**. If we have **interval logged values**, the **time interval** for logging in a contiguous field is defined in **sec**. For example, if we select to have **Interval logging** for RMS with 5 second intervals, we will **get a new** value of RMS after each 5 second interval. After that, the value is **reset** and the calculation is **started again** over.

We have several parameters to calculate. In short, the "**root means square**" (abbreviated **RMS**) value is a statistical measure of the magnitude of a weighted signal, **Peak** is the maximum deviation of the signal from the zero line, **Crest** is the ratio between the peak and rms, **VDV** is the fourth power vibration dose value, **MSDV** is the motion sickness dose value while the **MTVV** is the maximum transient vibration value, calculated in one second intervals.

| Output channels | | |
|---|--|--|
| <input checked="" type="checkbox"/> RMS | <input checked="" type="checkbox"/> VDV | <input type="checkbox"/> weighted raw |
| <input checked="" type="checkbox"/> Peak | <input checked="" type="checkbox"/> MSDV | <input type="checkbox"/> al (ISO 2631-5) |
| <input checked="" type="checkbox"/> Crest | <input checked="" type="checkbox"/> MTVV | <input type="checkbox"/> D (ISO 2631-5) |

Each value is **calculated for each axis** individually while the RMS, MSDV, VDV and MTVV are also calculated for the **sum of all three axes**. These values are enough to evaluate the human vibration exposure according to ISO 2631 and ISO 8041.

The next output is the **weighted raw** channel. This is the full speed time signal weighted with a chosen filter. We can use those channels for the calculation of the FFT or CPB spectrums.

al and **D** are the values based on ISO 2631-5 which describe the calculations and the limits for lumbar spine response to vibrations. The base for this standard is that the professional drivers of buses or trucks are exposed to vibrations when driving on rough roads or over bumps. Multiple shocks cause transient pressure changes at the lumbar vertebral endplates which can causes damage after years of driving.

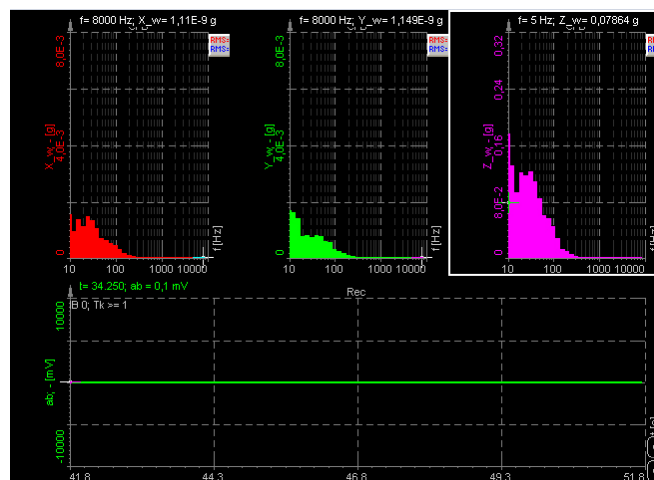
The **al** is the lumbar spine response from excitation measured in all three directions. The **D** value is the acceleration dose, measured from the lumbar spine response. These values are enough to evaluate the human vibration exposure according to ISO 2631-5.

4.3.3 Measurement

The human vibration module does not automatically create any **display**. Basically we only need to display the **measured statistical values** of interest. For long term measurement, it might be a good idea to **turn off** the storing of **input channels** and to **just store** the **output statistical quantities**. This will reduce the file size in the end.



Some applications require the user to measure the CPB or narrow band FFT. For this we need to *enable* the *weighted raw channels* and select them in the FFT or CPB display. This will give us the *weighted frequency spectrum* of the signal. The CPB spectrum will be *slow* since the bands with low frequencies needs a longer time to recalculate.



4.4 Order tracking

The order tracking method is used to **extract the harmonic components** related to rotational frequency of the machine. The machine vibration pattern is a mixture of excitation frequencies, usually related to rotational speed, such as unbalance, eccentricity, bearing faults and others and machine response function, which *relates* to machine natural frequencies based on the structure and mounting of that machine.

With order extraction, we can see a *specific* harmonic component which relates to a certain machine fault. That is - the *first order* (harmonic) usually relates to *unbalance* of the machine, the *second* harmonic often relates to *eccentricity*, such as if we have for example 9 rotor blades, the *9th* harmonic relates to errors on the blades. Or, if we have for example 31 teeth on a gear, then the *31st* harmonic will show the gear mesh frequency.

These are *excitations* forces which produce vibration accelerations. The ratio between excitation and system response is defined by the system transfer curve. Thus, the *final* measured vibration of the system is a product of the excitation force and the system transfer curve. Since the transfer curve is fixed, we get different responses for excitations at different rotation speeds.

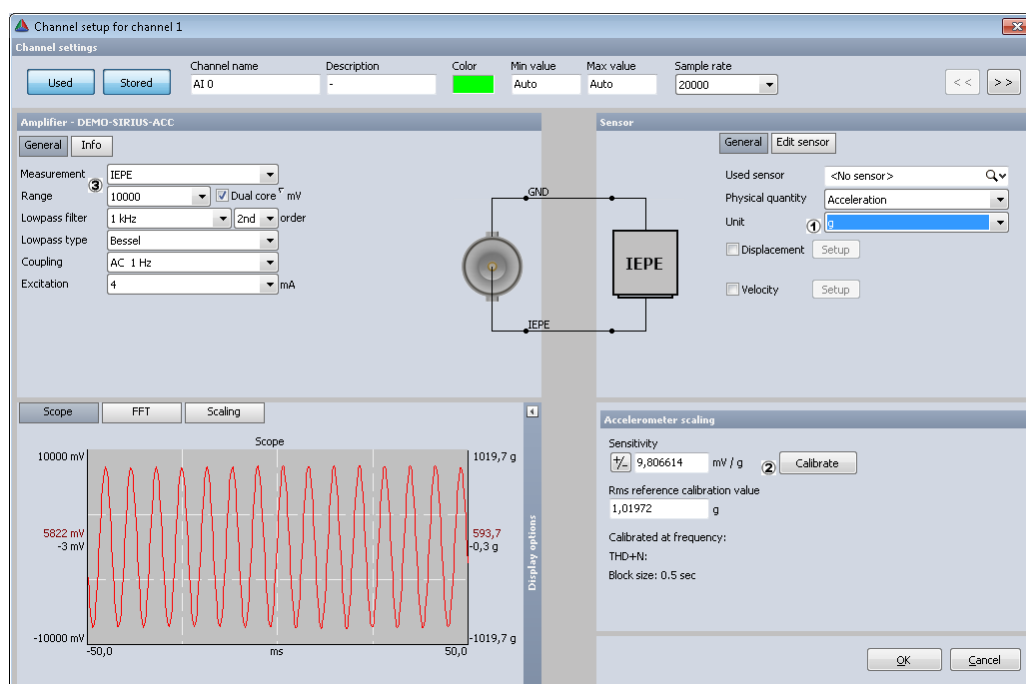
When the excitation *passes natural frequency*, we get so-called resonance with increased vibration amplitudes, which could be fatal to the machine.

| | |
|-------------------|--|
| Required hardware | Sirius ACC, MULTI, DEWE-43 |
| Required software | SE or higher + ORDTR option, DSA or EE |
| Setup sample rate | At least 10 kHz |

4.4.1 Channel setup

For order tracking, we need *analog channels* to measure *vibration*, *noise*, *pressure* or other parameters from which we would like to extract the orders. Additionally, we need the sensor for measuring *rotation speed* of the machine, either an *encoder* or a *tacho probe*. It is recommended that we use Orion counters for calculating the RPM or, for tacho, to simply acquire an analog signal. Then we need to make sure that the sample rate is high enough to see the triggers, even at the maximum speed. Please note that we also need to use internal sampling for the correct calculation of the order tracking.

In this case we have one *accelerometer* for measuring the vibration and one *encoder* for measuring RPM. First we **set up** the *vibration channel*. We enter the **Unit** of measurement in **g** ① and the **Sensitivity** ② of the sensor as **96 mV/g** from the calibration paper of the sensor. Next, we select the measurement **Range** ③.



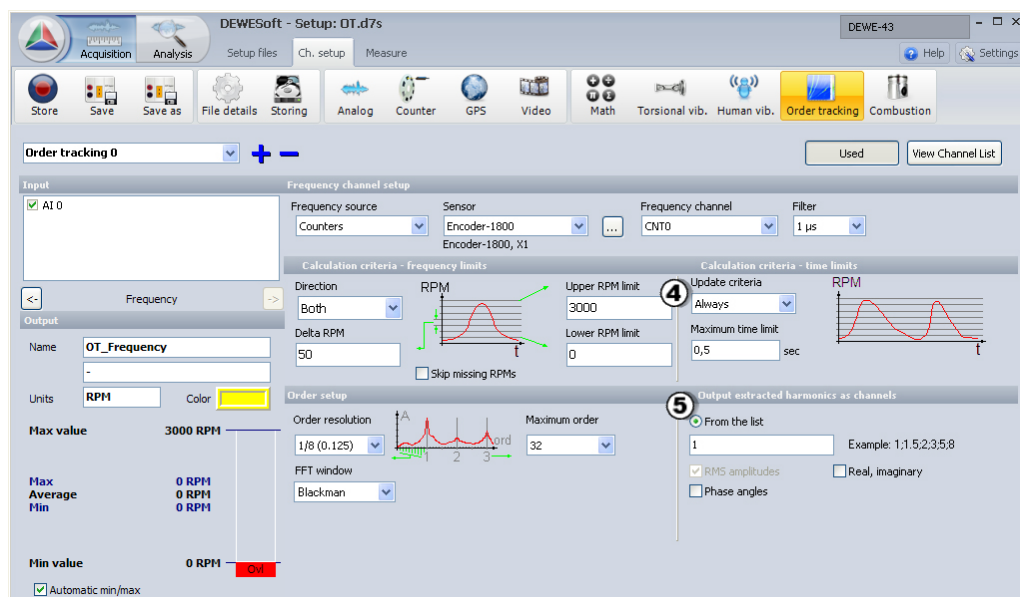
The following step is adding one **Order tracking math** module and selecting the source of the frequency measurement. We can select any counter sensor. If we choose the encoder, we need to connect the encoder to the counter

input. If we have a tacho sensor (or any other analog sensor for that matter), then we need to *connect* the *sensor* to the *analog input* and define the *trigger levels* by clicking the **Setup** button, which appears to the right of the **Frequency channel** settings.

Next we define the **Lower RPM limit**, **Upper RPM limit** and **Delta RPM**. These three parameters are important for *reserving memory* of *waterfall FFT* based on the order tracking. In this example we expect the RPM to be in the limits of a 0 to 3000 RPM range and we want to have an FFT each 50 RPM. This will give us the $(3000-0) / 50 = 60$ frequency spectrums over defined RPM range. We can also define the **Direction of triggering** either to capture only **run-up**, **coast-down** or **both**. Additionally we can define **Delta time** interval. If the RPMs are not changing, we can define the **Maximum time limit** for calculating the harmonic point. It will calculate a new point in the run up diagram after this time interval to be able to see the deviation of amplitudes at the same speed. In this section we also define the time **Update criteria** for calculation. **Always** means that the data will be acquired any time the frequency range is passed, while **Only first time** means that the data will be acquired *only the first time* that the frequency range is *entered* and not subsequently ④.

Then we define the Order **setup**. For the calculation of the frequency history spectrum we define the *order resolution*, *maximum order* and *FFT window*. FFT window is the window used for the calculation of the FFT spectrum. Since the data captured is angle based, we can also use "soft" windows like **Hanning** if only a harmonic component is seen in the signal. If we have many non-harmonic components, then it is recommended to use a window like **Blackman**. The maximum order defines the order span of the frequency spectrums. When the data is recalculated, it is *filtered* to this maximum order. Also the result of the frequency spectrum will be limited to this order in Order OT mode. This number depends on the maximum excitation frequency we expect. The order resolution defines what the steps will be in the frequency spectrum. If we want to see the natural frequencies nicely, it is recommended to use a **0.125** order.

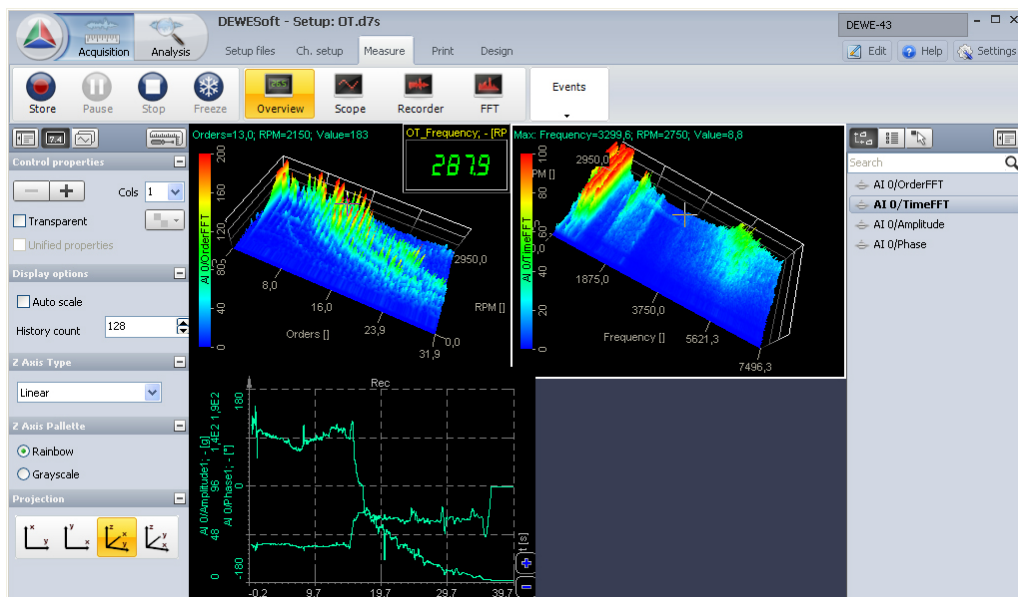
Next we need to define the Extracted harmonics ⑤. We can enter a list (like **1;3;8;12**) or define to extract **All** orders. This will *create* the channel for *each* order defined in the order setup, so with our setup we would have 512 channels ($8 \times 64 = 512$), which is hard to handle. Thus, it is recommended to extract only the orders that are actually interesting. We can extract the *amplitudes*, the *real* and *imaginary* parts and the *phase* of each **harmonic**.



4.4.2 Measurement

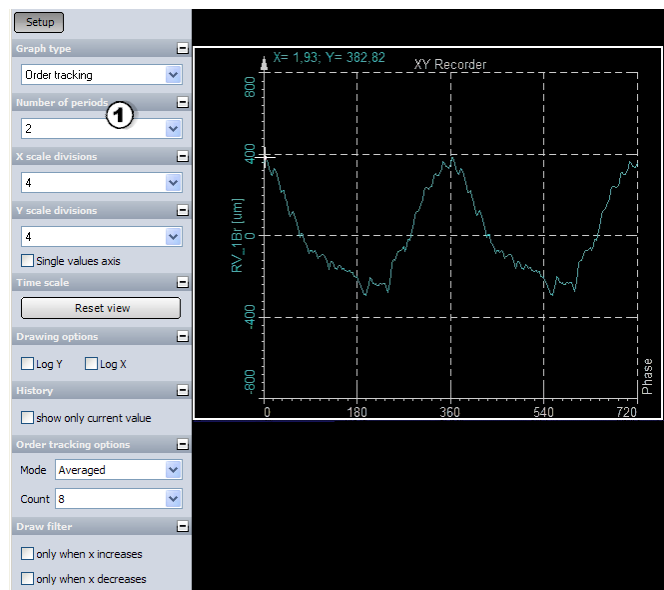
Now we can make some measurements. The order tracking math doesn't create any display on its own, so we need to define the **display manually**. I have added the *analog* and *digital meter for rotation speed* and *recorder for rotation speed* and *vibration value* just to be able to see the run up. Next we can use the *xy recorder*, where the *x* axis should be *OT_Frequency* and *y* axis can be used for the harmonic components (in our case *Vib H_1*, *Vib H_2* and so on). We need to select the "Single X axis" mode of the recorder and it is recommended to use the "Real data" display type to show every point of the run up on the xy graph.

Next, I have added the frequency plot for Vib signal to be able to see the current frequency speed, and below, I have added the 3D graph. Then select either *TimeFFT*, if we want to display the *x* axis in frequency or *OrderFFT* channel, if we want to display the *x* axis in order. It is recommended to select *logarithmic* scaling and *two to three decades of amplitude resolution* (if we have 1g maximum, we choose 0.001g as minimum) to show the amplitude in nice colors. Less dynamic (two decades on the picture below) will emphasize the peaks more, while more dynamic will reveal smaller components of the run up.

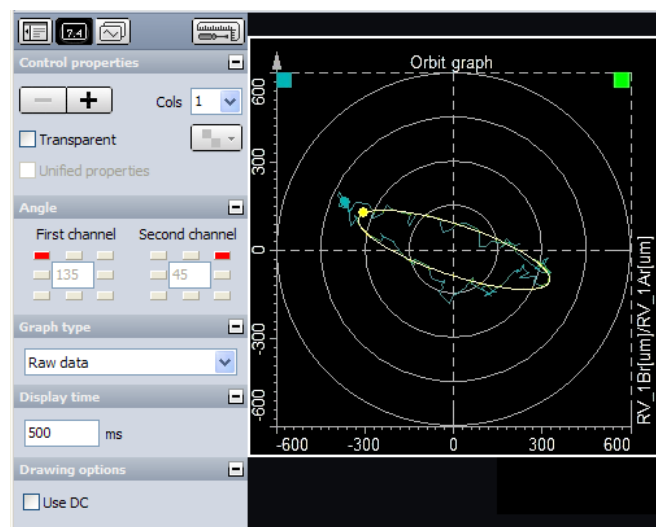


We can use the xy plot to display the amplitude of certain *orders compared to the frequency* (rotation speed) of the machine. In the same manner we can also display the phase to create the Bode plots. If we plot *real* vs. *imaginary*, which are used in the x-y graph, we can create a Nyquist plot as seen on the picture below.

There is also a nice way to see the current time data in the x-y plot. We can select the *Order tracking Graph type* in the x-y section, then we *select any channel* that is defined in the order tracking and we can see the periods aligned with the zero pulse of the encoder or the tacho signal in time. We can select any number of periods and choose to display *current period*, *averaged* number of periods or show the data with *persistence* ①.



If we have two *displacement* measurements on the shaft axis, we can display the *shaft movement* in the orbit plot. We can select the *Order tracking* mode and **Display Current** or **Averaged** data, or show *orbits* with *persistence*. We can also display the harmonics, like shown on the picture below. The **yellow** dot shows the position of the *zero pulse* of the *angle sensor*.



5 Combustion analysis

The **DEWESoft Combustion analysis** math module enables the **Analysis** of internal **combustion engines**. If we measure the *pressure* inside the cylinder and the *angle* of the shaft, we can **calculate** the main indication *values* for *engine development* and *testing*, like maximum pressure, position of maximum pressure, heat release, knocking and other important parameters.



The **combustion analyzer** is fully integrated inside the **DEWESoft** software, which means that we can use any functionality of **DEWESoft** including *CAN bus*, *video*, other *analog signal acquisition* and more...

For **combustion analysis** measurements we need:

| | |
|--------------------------|-----------------------------------|
| <i>Required hardware</i> | Sirius MULTI, STG, ACC or DEWE-43 |
| <i>Required software</i> | DEWESoft PROF with CA option |

5.1 Channel setup

Analog setup

First please *select all* necessary *analog channels* and *scale* the parameters according to the sensor calibration data. The *pressure sensors* are usually *charge pressure transducers* and we need external *amplifier* which can correctly acquire these signals, since it can also acquire very low frequencies (or even DC pressure).

There are two basic choices - either a **DC** or **AC** input coupling. A DC coupling will also measure *static* pressure, which is useful when a calibration is performed. A DC coupling also features a **Reset** button which resets the charge to **zero** ③. Now we can apply a static pressure and calibrate the sensor.

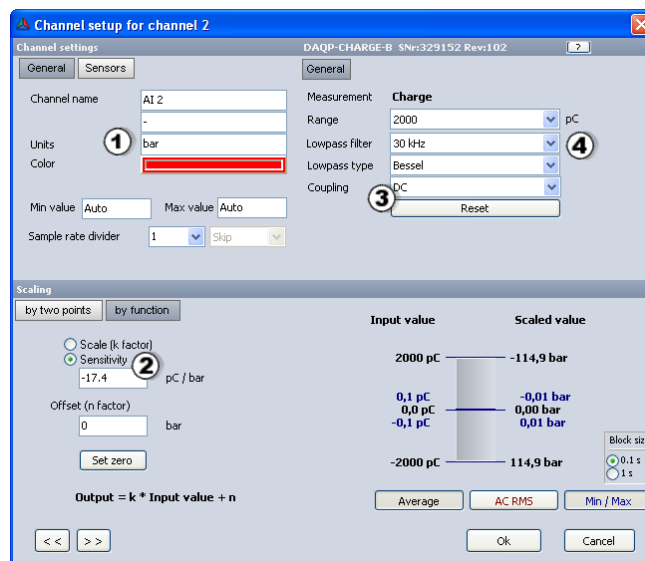
Please note that for *normal* (especially long term) measurements, the AC coupling should be used. The AC coupling has a time constant from **2** to **1000 second**, so the pressure signal from the engine will also be measured correctly.

Another possibility is using the *calibration certificate*. When we have *factory* calibrated sensor, we will get the certificate with

stated sensitivity. Pictured below is an example of such certificate.



We enter the **Unit** of measurement as **bar** ① and then go to the **Scaling by function** tab and select the **Sensitivity** ②. We *enter* the sensitivity from the certificate (-17,4 in our case). Once we have entered the correct scaling, we can immediately see the scaled range and current pressures. The range can be adjusted by selecting the **amplifier measurement Range**. Sometimes these sensors produce high frequencies *spikes* which can be *removed* with the **Lowpass filter** (10 or 30 kHz) ④. If the signal is *clean*, it is recommended to set the *highest* filter (100 kHz) in order to see the *entire dynamic range*, even at *high* RPMs.



Angle sensor setup

There are two main *sensors* that are used with a **combustion analyzer**: *angle sensors* with 360 or more *teeth per revolution* and with a *zero pulse* for *each* revolution, or the *in-car* angle sensors which are basically a *geartooth* with some *missing teeth* or some *double teeth* (a typical example is 60-2 which would have 60 teeth except two of them are physically missing to mark the beginning of the revolution).

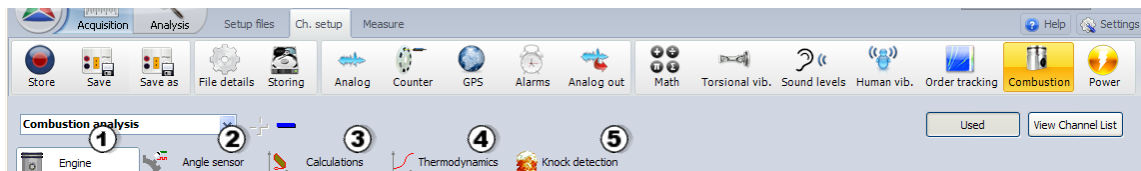


Sensors are usually *connected* to CA-CPU, which is the device that *creates any number of pulses* and the *zero mark* from the *input signals*. In some cases sensors are connected to the *analog input*, but let's continue with the instructions to see how to wire the inputs for each case.

You *shouldn't* set anything on the *first two counter inputs*, because the combustion analyzer will in most cases *need* those counters for *internal* operations. If you have any *additional* counters (CNT – Expansion), you can set them however you like.

5.2 Combustion setup

Select the **Combustion** button and click a **blue plus** button to add a combustion analysis **module**. We can set the combustion analyzer in any way we want, but usually the easiest way would be to follow the five steps that are shown in the following image:

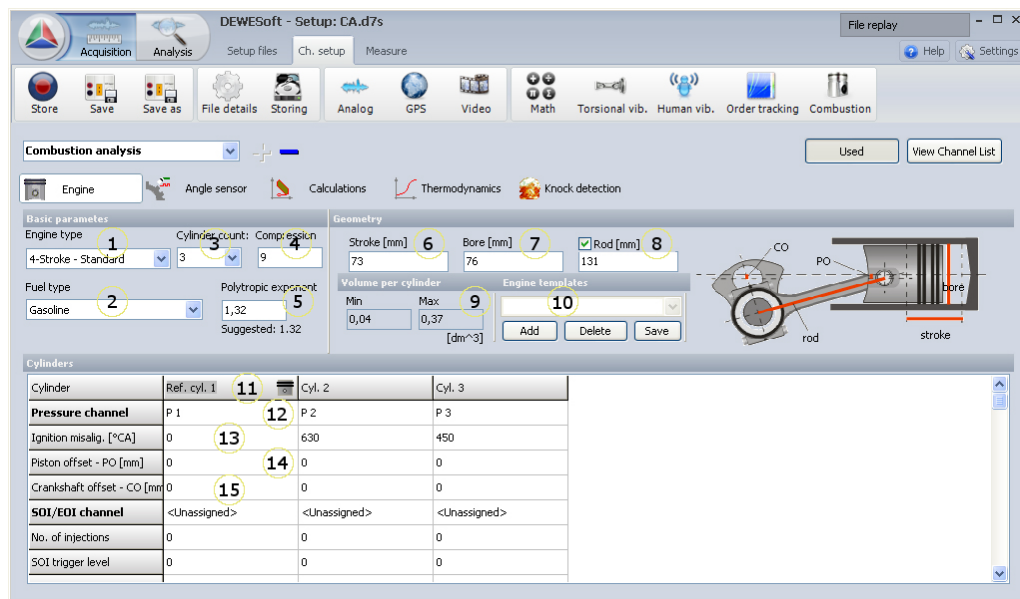


1. Engine setup

Here we **set** all engine parameters (*geometry, engine type,...*). We define the *type of engine* (gasoline or diesel [2], four stroke, two stroke [1]), cylinder count [3] and engine geometry [6÷10]), which has to be identified for *volume* calculations. For each *cylinder* we can also define the piston and crankshaft offset [14, 15], but for most engines these values are zero. We need to define the polytropic coefficient [5], which is a factor for compression (without ignition). This is important for the calculation of thermodynamic zero and for heat release. If you don't know the *polytropic exponent* for your engine, take the suggested value for each engine type. We will see more details about how to find this value below.

Next, we define the channels that correspond to the cylinder pressure [12]. Note that we can leave some cylinders *without* assigned channels (like *Cyl. 4* in picture below) and those channels will *not be* calculated. We also need to define the ignition misalignment. This defines the firing order of the engine. For example, if we take a 6 cylinder engine, the firing order is 1-4-3-6-2-5, so if the whole cycle is 720 deg, the ignition misalignment will be: 0-360-240-600-120-480.

We can also define the *start/end of injection* channel, trigger levels and *number of injections* here. We will get the angles of injection as *additional* channels during the measurement.



2. Angle sensor setup

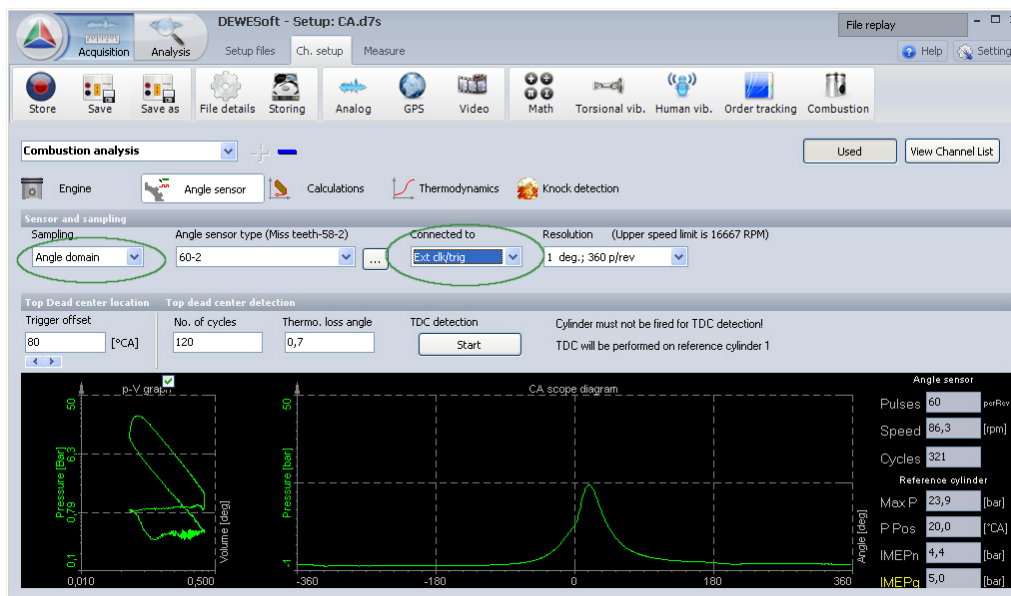
The angle sensor setup is a very important part of setting up the combustion analyzer. It defines the *performance* and *abilities* of the combustion analyzer module. We have two basic ways of acquiring this data: the *internal* and the *external* clock. These two modes totally redefine the acquisition process. Let's take a more detailed look at these two modes.

External clock

External clock means that the data will be acquired in the **Angle domain**. In other words, if we choose *360 pulses per revolution*, it will *always acquire 360 points* regardless of the current speed of the engine. This procedure is useful for *high speed* and/or *high number* of cylinders since it acquires the data directly, which is suitable for the calculation of combustion parameters. However we can't use any additional time domain calculations, for example combustion noise. We also can't acquire a cold start from 60-2 or similar sensors.

In the case of an external clock we need the CA CPU hardware to *create* the needed number of *pulses* from *any sensor*. In the next step we choose the *angle sensor* from the list (in our case a 60-2). If the sensor doesn't exist, we can *add* it by clicking the three ellipsis button on the right side of the sensor. CA-CPU supports *encoder*, *CDM sensor* as well as the *geartooth with missing* and *double teeth*. We define *where* the *input connection* of the sensor is on the CA-CPU and the *output resolution*. This resolution defines the *maximum speed* (limited by the AD card *maximum rate*) and the *calculation load*. If we choose 0.1 degree resolution and we have 500 ks/s AD card, our maximum RPM will be $500000/3600 \cdot 60 = 8300$ RPM. The output resolution *directly* defines the *number of points per revolutions* for all calculations as well as for CA displays.

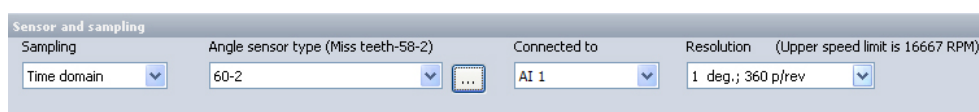
As soon as we define the engine setup and the angle sensor correctly (and the engine is running), we should see the CA-CPU *tracking message* and already see the *number of pulses*, *speed* and *cycle count* in the right side of the display.



Internal clock

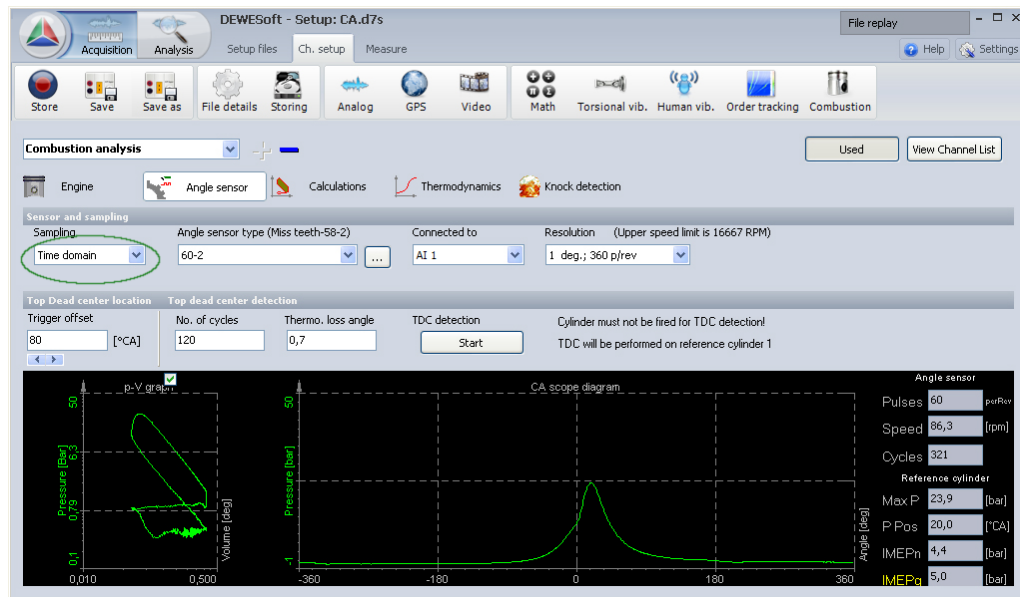
Internal clock uses a totally different method. It *acquires* the data in the **Time domain** and *recalculates* by software the angle domain for **CA** calculations. Since the data is *low-pass filtered* according to the speed of the engine, the calculation is more demanding than with external clocking, but on the other side it is *easily possible to correlate time domain* data (like *CAN bus*, *video* and so on) and *calculate* time domain parameters like *combustion noise*.


We *don't* have to use the CA-CPU in this case, but we connect the CDM or encoder sensors *directly* to *counters* of the Sirius or DEWE-43.

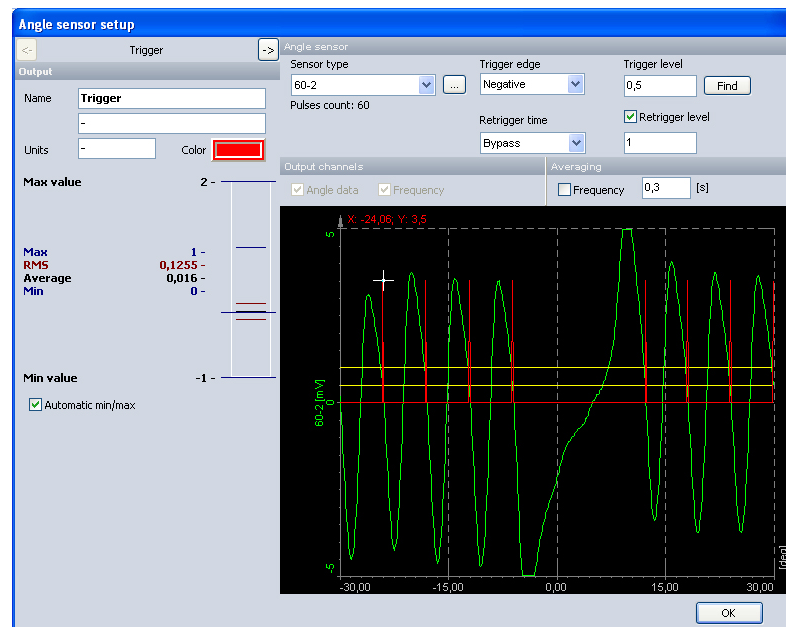


If we have a *geartooth sensor* with *missing* or *double teeth* we connect it *directly* to *analog channels* and select it (like in the picture below). Additional benefit of the internal clock is that we *can* measure a **cold start**. If we use an external clock, the CA-CPU will *miss* the *first few cycles*, which are the *most important* for cold start.

Since the **angle sensor math** is used the back, it *waits* until first gap is detected and also calculates the cycle before. To use this, we need to set the *trigger levels* and *direction* by clicking the **Setup** button.



The easiest way is to click  - the **Find** button which should set the *trigger edge*, *trigger* and *retrigger level*. On the graph below, we can already see the live data.

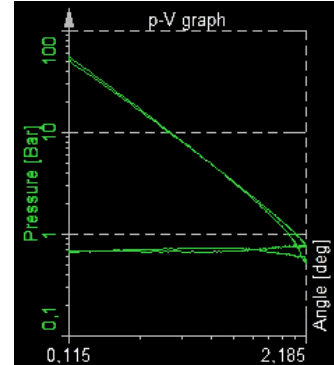


The output resolution and the sample rate define the *maximum speed* of the engine where the CA still works correctly. We can easily see this from the message shown to the right side of the resolution.

Top dead center detection

Regardless if using internal or external clock, we will see some data already in the *CA cycle graph* when we finish with the settings. To see the *p-v* diagram correctly, we need to define the *trigger offset*. Trigger offset is the *angle* between the trigger and the *top dead center* of the *reference cylinder* (noted as *ref. cyl.* in the engine setup).

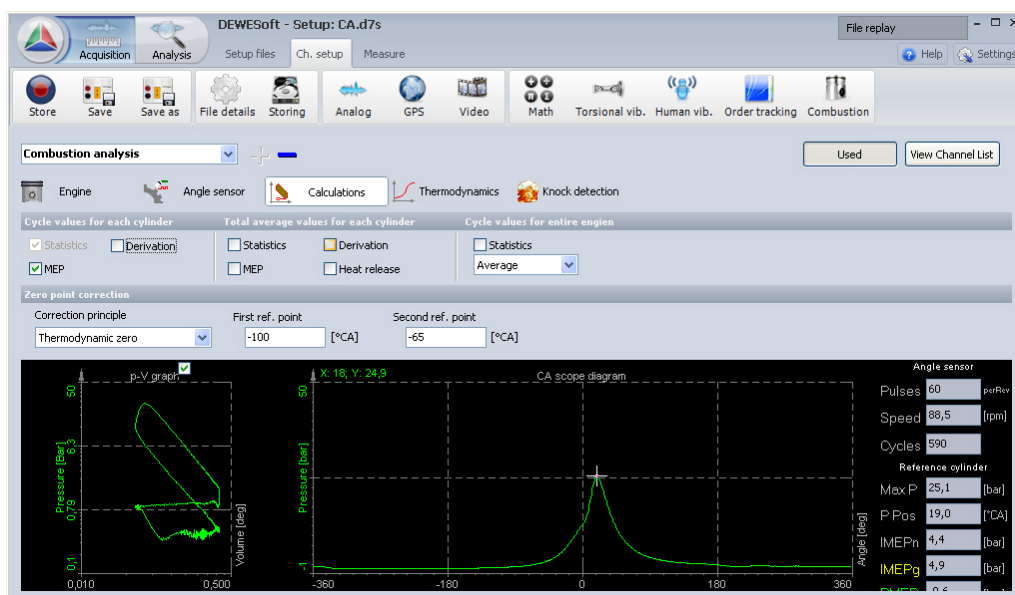
This angle can be *physically measured* or it can be also defined with **TDC** (top dead center) detection. The engine must not be fired for this procedure. In the test bed this is easily achieved since we have an option to switch off the ignition. We can use the combustion analyzer inside the vehicle by driving to a *certain speed*, leaving the car in gear and *releasing the gas pedal* so that the engine *brakes* the vehicle. Then the engine will also run in the so-called *motored cycle*, meaning only compression and expansion with no work. The right picture shows a typical motored cycle, where we clearly see that the compression fits the expansion exactly, therefore no work is done inside the cylinder.



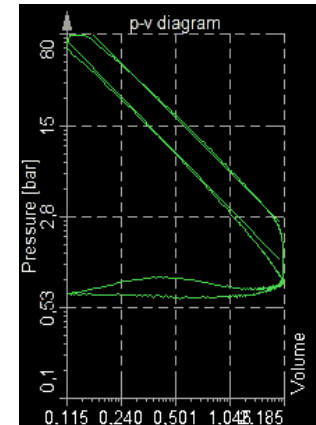
We enter the *number of cycles* for averaging and the *thermodynamic loss angle*. This is the *delay* between the top dead center and the *pressure peak*. The pressure peak happens a bit after top dead center and we can compensate for this by entering the loss angle. Then we select the **Start** TDC detection procedure. The pre-defined number of cycles will be acquired and the trigger offset will automatically be calculated and set.

3. Calculation setup

Here we define the *output channels* and the pressure correction method. The last thing to set is the **Zero point correction**. As was already mentioned, the charge sensors will drift over a longer time, so we need to calculate the *absolute* pressure. This can be done in three ways: with **Thermodynamic zero**, which looks at the compression and assumes that if we were to *expand the volume to infinite*, the *absolute pressure* should be **zero**.



At this point we need to take a closer look at the p - v diagram. In a logarithmic p - v diagram the *polytropic compression and expansion* is a **straight** line and the *steepness* of that line is defined by the *polytropic coefficient*. We can easily see this during the measurement, as shown in the picture below. If the polytropic coefficient doesn't fit, the zero correction and heat release will be calculated in the wrong way.



To come back to the **zero point correction**, we need to define *two* angles in the *compression* part of the cycle where the compression fits with the polytropic exponent. If we define the points where the ignition already starts or where the intake valves are not closed, then we will have a wrong calculation.

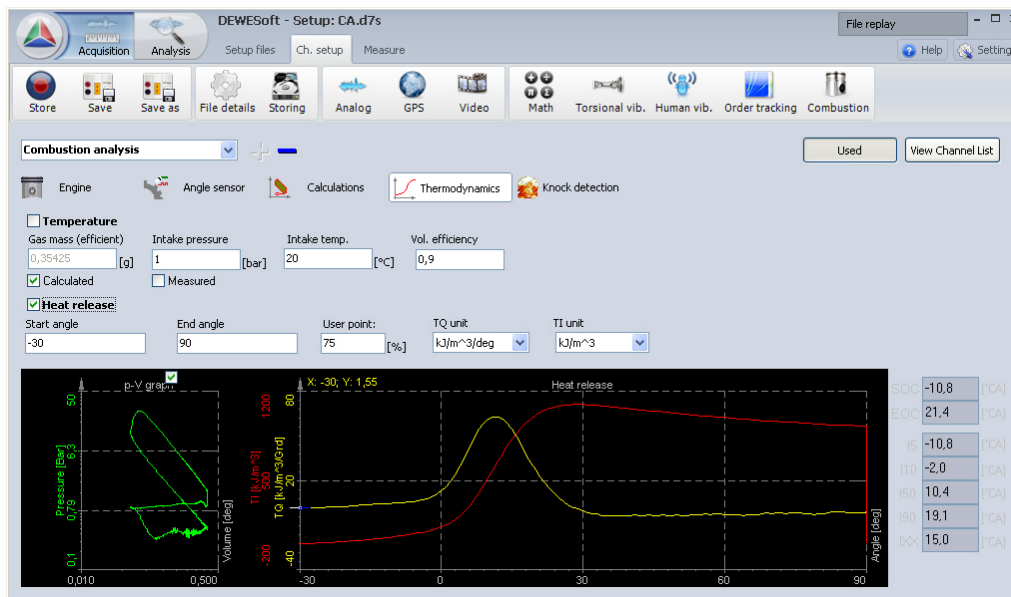
Other options are to *enter* the *known* or *measured pressure* at a certain angle (usually at intake).

Now we are finished with setting up the system and we only need to choose which **parameters** we would like to see during measurement. The **statistic values** like *maximum pressure* and *position* of maximum pressure will *always* be calculated. We can choose the *derivation* of the pressure signal as an option and we *get* the *value* and the *position* of maximum pressure *rise*. The **MEP** option will calculate IMEP_n (for the *entire* cycle), **IMEPg** (for the *working* part of the cycle) and **PMEP** (for the *pumping* part of the cycle). The MEP value is the mean effective pressure and is actually the *area* inside the p - v diagram (energy of the engine), *normalized* to the *displaced volume* of the cylinder.

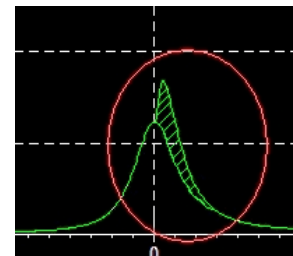
Additionally we can get the total average values (for the entire run) for each cylinder by selecting Statistics, MEP, Derivation and Heat release. This will allow us to see averaged data in CA-scope, as well as adding the total statistical values.

4. Thermodynamics setup

In this section we turn on the calculation of the **heat release**. Heat release is quite intensive mathematically, so we need to set it up with care. We *set* the *Starting point* and the *Ending point* of the calculation, because heat release only makes sense *around* the top dead center of the working part of the cycle. The result from this calculation is the *start of combustion*, *end of combustion* and the *degrees* at a certain percentage of the curve.



First let's take a look what heat release is. Heat release is indicated as the *pressure rise above the polytropic compression and expansion part of the cycle*. We have already shown a difference from the motored cycle to the working cycle in p-v diagram, but now let's repeat this in a CA graph. In the picture below the motored cycle is drawn below the working cycle and the area between those is the work produced by the engine. The part of the cycle where those two curves don't fit is the part where the heat is released from the fuel.



In this section we can also choose a temperature calculation. By selecting gas mass or defining intake pressure, intake temperature and volumetric efficiency, the temperature in the cylinder can be calculated.

5. Knock detection setup

Knock detection is the procedure that *filters* the data with a *high* and *low pass filter* and *compares* the *expansion* to the *compression stroke*. The result is the *knocking factor*, which shows the *amount* of knocking inside the engine.

☐ **Knock detection (Method Mannesmann VDO AG)**

Lowpass filter: [taps]

Highpass filter: [taps]

Noise threshold: [bar]

Reference signal window width: [°CA]

Knock signal window width: [°CA]

Shift reference window:

Speed: [RPM]

Correction: [°CA]

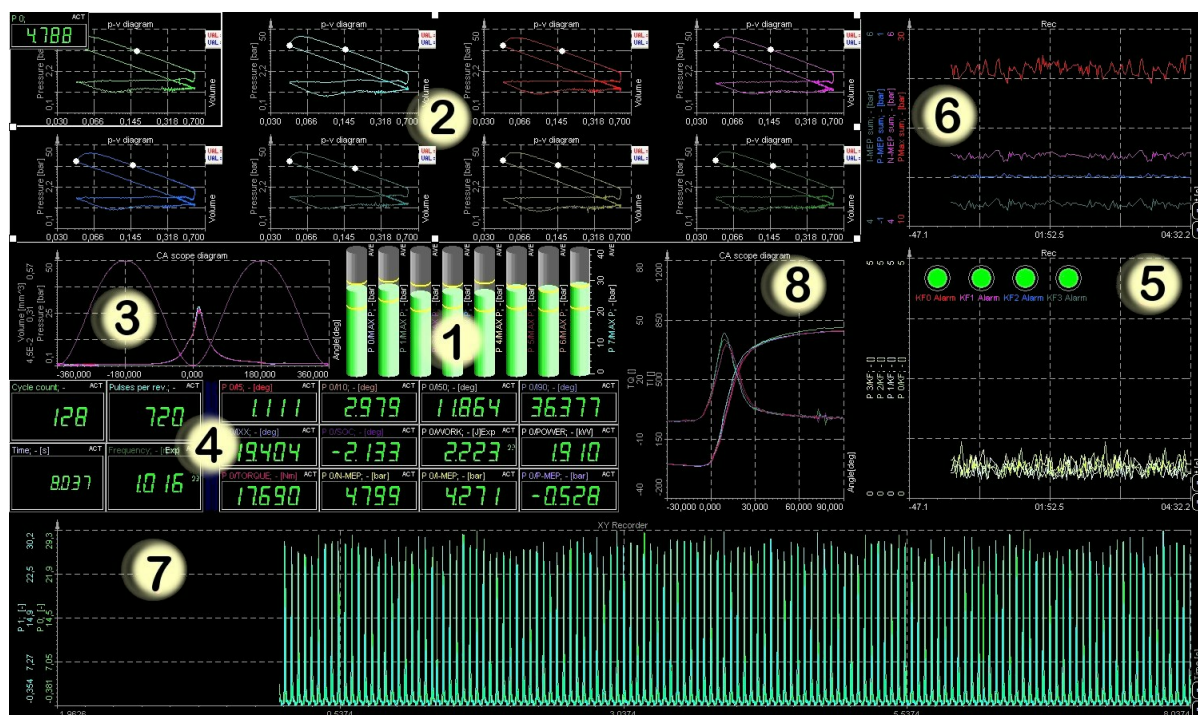
[RPM] [°CA]

5.3 Measurement

Now that we have set up all the parameters, we can acquire some measurements. The combustion analyzer doesn't create any *display*, so we need to build our own.

We can see typical a CA setup in the picture below which shows:

- Maximum pressure values for *all cylinders* ①
- Pressure versus volume (pV) – white dots present start and end of combustion ②
- Cylinder volumes and pressures (CA Scope) for the *last cycle* ③
- MEP values (IMEP_n, IMEP_g, PMEP), frequency of rotation, captured cycle count... ④ ⑥
- Knocking factors ⑤
- *Heat release* (TI, TQ, burn angles) ⑧
- Time, as well as angle based data in the recorder ⑦

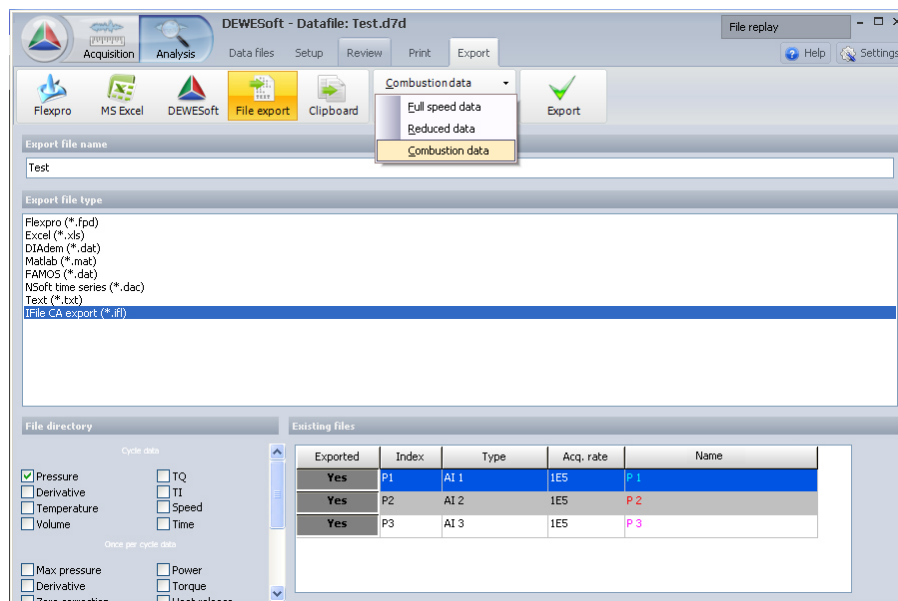


When we use **CA math**, we have *two more displays*: *p-v* and *CA scope*. For more information how to use them, please consult the user's manual. If we use *external clock*, we can use the *x-y recorder* to show the *time domain* data. CA math *creates one additional channel* called *Time*, which can be used for *x-reference* of the *x-y*.

5.4 Analysis

To **review** the CA data, we can use all standard procedures for the **analysis**. When *moving* through data, the current cycle will be shown in either *p-v* or *CA scope*.

One important option is the ability to **export** the *angle* based data in *different formats*. We can select this by choosing the **Combustion data** option in the **Export** section. We can choose *standard* formats supporting the real time data, like Flexpro or Diadem, but can also export the data in a *standard combustion IFile*, which can be *imported* by most *combustion analysis post processing* applications.



We can choose several export options:

Items listed under **Cycle data** are the *angle based* values or pressure and other parameters. It is actually data displayed in the *p-v* scope.

Items under **Once per cycle data** are the statistical quantities (**MEP**, **Max pressure** ...) calculated *once per each* valid cycle. They are the result of combustion calculation.

Items under **Cycle average** will export *only one* single cycle which contains an average pressure, heat release, temperature curve. Data is calculated as angle based average of all acquired cycles.

6 Networked data acquisition tutorial

| | |
|--------------------------|-------------------------------|
| <i>Required hardware</i> | Any system |
| <i>Required software</i> | SE or higher + NET option, EE |
| <i>Setup sample rate</i> | Depends on math module used |

The **DEWESoft NET** application module allows one or more **measurement units** (named **MU**'s) to be under the **control** of other computers, named **CLIENTS**. The **MU(s)** and **CLIENT(S)** must be *connected* via TCP/IP. The best choice is a *high-speed* and *direct Ethernet* network topology, although wireless Ethernet can also work well. *Mixed* hard wired and wireless systems are feasible.

Working with **DEWESoft-NET** is comprised by **three basic steps**:

NET Setup

Network *configurations*, appropriate hardware and **DEWESoft-NET** *setup*:

- **Setting up Client And Measuring Units**
- **Remotely Controlling a Slave MU**

Measurement

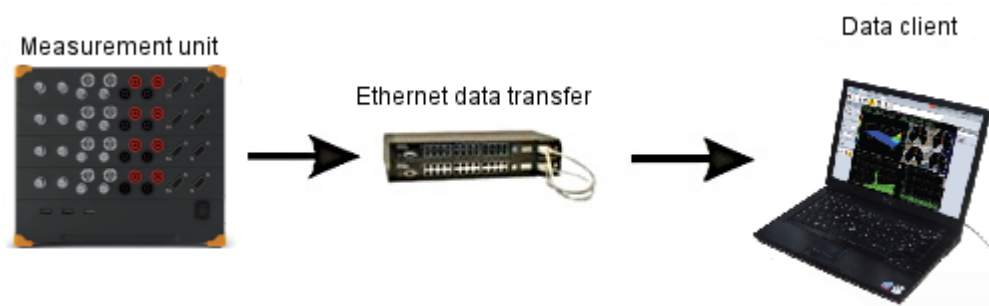
Creating a display, *measuring* and *acquiring* data and storing this data on a network

Analysis

Analyze acquired and stored data on network, *export* measured data

Principle of DEWESoft NET application module

It is important to note which channels can be **viewed** on the **CLIENT(s)**, and the actual data which is **stored** on the **MU**'s. This is critical *to safeguard* against data loss which might occur by the network going down or transmission being interrupted. Even if this happens, the data is *safely stored* on the **MU(s)**. When the network connection is reestablished, it is possible to *reconnect automatically*.



It is often *not possible* to **transfer all** possible channels in real time to the client for storage, *even* using a gigabit Ethernet interface.

Imagine even one Dewesoft system with **32** channels, being sampled at **200 kB/s** each at 24-bit mode. This is already 25.6 MB/sec, which is more than **200 Mb per second** (where each Byte = 8 bits). It does not take long for the network to be completely *overloaded* with data and be overwhelmed with packet loss.

Immediately after the **acquisition** is *stopped*, a button appears on the controlling **client** allowing the *data file(s)* to be

uploaded *immediately* from the **MU**'s for *viewing* on the **client** computer.

6.1 Setup Measurement Units and Client

Client and Measuring Unit Configurations

There are *several* possible configurations and thus **Setups**, including:

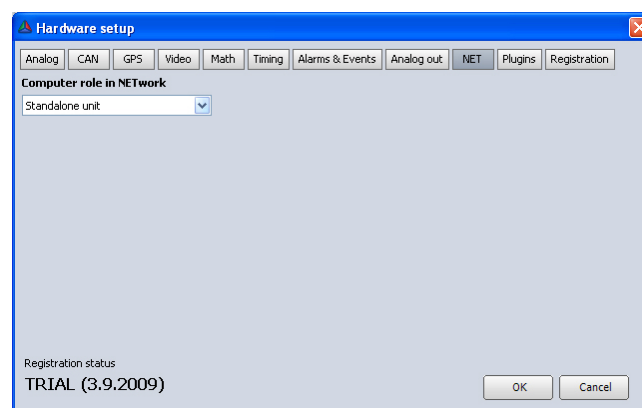
- **Standalone unit** – *not* networked to anything (the *default* setting of **DEWESoft** upon installation)
- **Slave measuring unit** – can *measure* data under either local *control* or under the *control* of a master **client**
- **Master measuring unit** – can both *measure* data *and* *control* other **measurement units** (optional)
- **View client** – can *view* data being *recorded* on the **measurement units**, but cannot *control* them
- **Master client** – can *control* the **measurement unit(s)** and *view* their data

Each **MU** must be **configured** as a **slave measuring unit** in order to *utilize* the **DEWESoft NET** software – however if you are going to use *one* of your **MU**'s as the controller for the others, then you should *configure* that one unit as a **master measuring unit**. It is *not possible* to have *more* than one "**master**" within a single **DEWESoft NET** system, in order to avoid confusion and conflicts.

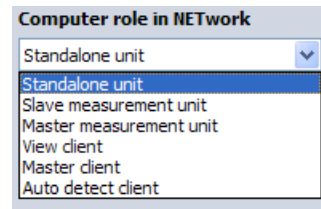
Hardware setup

To activate the appropriate mode for each system within the **DEWESoft NET**, first run **DEWESoft**. Now click on the **Settings** menu and select **Hardware setup....**

When the dialog box appears, click on the **NET** button on the top, and you will see this screen:

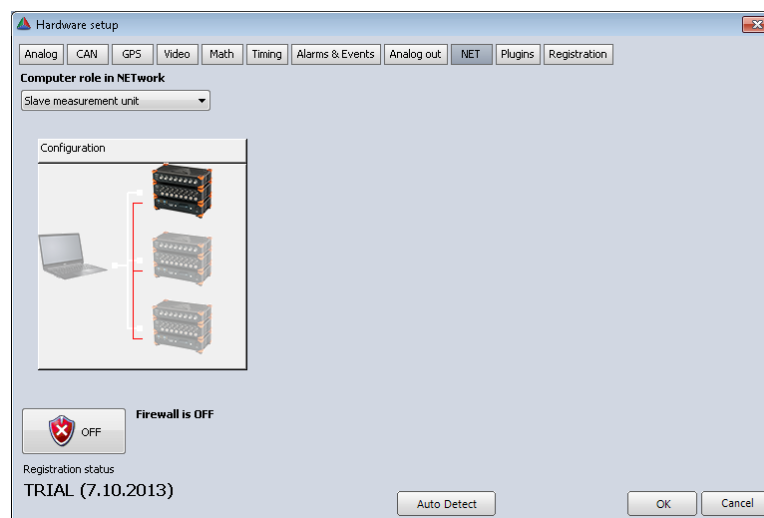


In selector list you can select appropriate type for your unit.



Setting up a Slave Measurement Unit

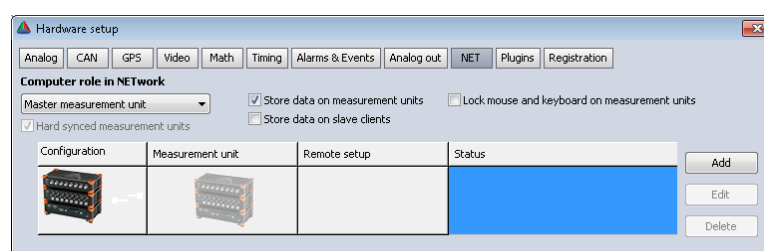
Assign the *role* of this system using the selector list (which in the picture above calls **Standalone unit**). Assuming that this is our *first* measuring unit, set it to **Slave Measurement Unit** as shown below:



Note that a *graphic* named **Configuration** appears which *indicates* the topology type, where this system is the *darker* picture ... some kind of acquisition system (the picture is only a *reference* – it is not meant to look exactly like your particular model). The *white* lines represent the TCP/IP network, which must be connected among all **MU's** and the *client* (represented here as a notebook computer, again only as a reference). The *red* lines represent the **SYNCR** which must be present if there is more than *one* **MU** and you want to *show channels* from more than *one* **MU** on the *client* at the *same time*. It also *ensures* that the *data files* recorded on *each* **MU** are, in fact, really **synchronized**. Unless you have a hard sync, as mentioned in the paragraph [DEWESoft Manual](#) → **Data Synchronization**, it is not possible to achieve accurately synchronized data files.

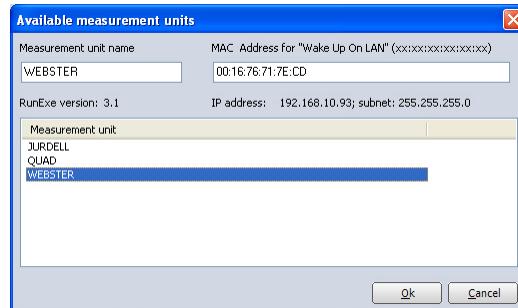
Setting up a Master Measurement Unit (optional)

As mentioned, if you are going to use **one** **MU** to both record data and control *one* or *more* other Dewsoft **MU's**, then you must *assign* this *one* to be a **Master Measurement Unit**:

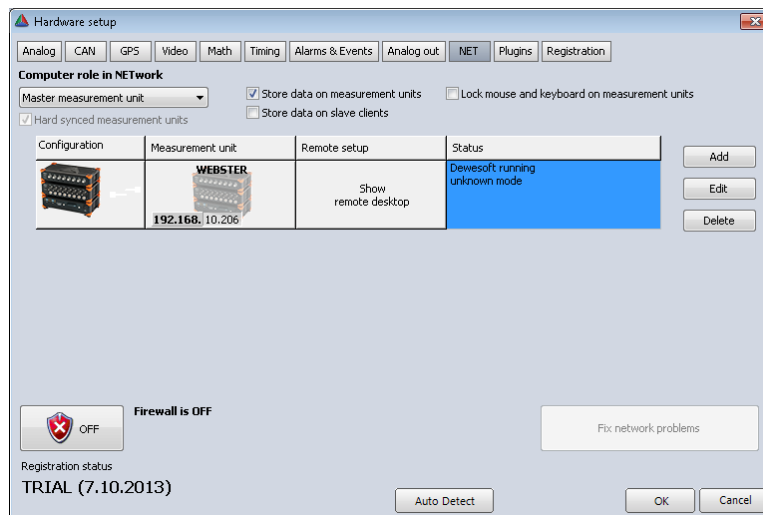


When you assign this system to be a **Master MU**, the screen will change to show a graphic where a **MU** is a *darker* image. By default it will show *one MU*, as shown above.

Assuming that you have already configured one or more other systems to be **slave MU's**, you can click **Add** here and **detect** any systems already available on the network:



In the example above, the system sees *one MU* on the network, called **WEBSTER**. Select it and click **OK** at the bottom of the dialog box:



Now this unit is shown as your *first MU*. If there are other ones, click **Add** to add them, and they will appear here on the list.

Note – this information is **saved automatically** by **DEWESoft**, so you won't need to enter it next time unless it has changed.

Note the check boxes above:

- **Store data on measurement units** (checked by default, and highly *recommended*!)
- **Lock mouse and keyboard on measurement units**

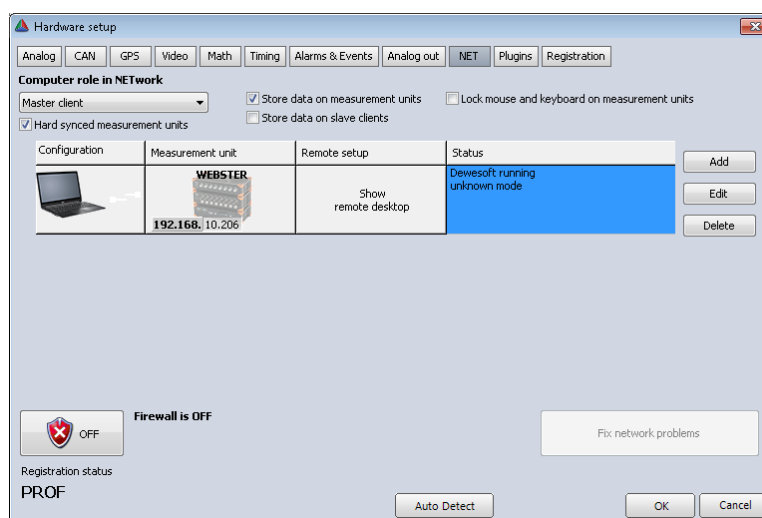
Use the "lock mouse and keyboard" checkbox if you want to *prevent* a local operator from changing any *settings* on the **MU**, or interfering with the test. With this checked, the **MU** cannot be operated *locally* – you will have *complete* control over the **client**.

IMPORTANT Do not does assign this computer to be a MASTER MEASURING UNIT if you will use a separate computer as the MASTER CLIENT. In that case, all MU's should be set to SLAVE MEASUREMENT

UNITS.**Setting up a Master client**

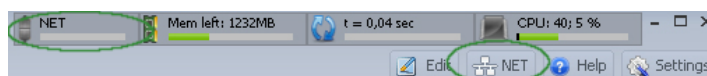
In the case where your Dewesoft systems are *all* slave measurement units, you will need one Master client to control them. Let's assume that we are now sitting at this computer and have DEWESoft properly installed and ready. Click on the **Settings** menu and select **Hardware setup...**, then click on the **NET** button.

Now use the selector to *assign* this computer to be the Master client, as shown below:

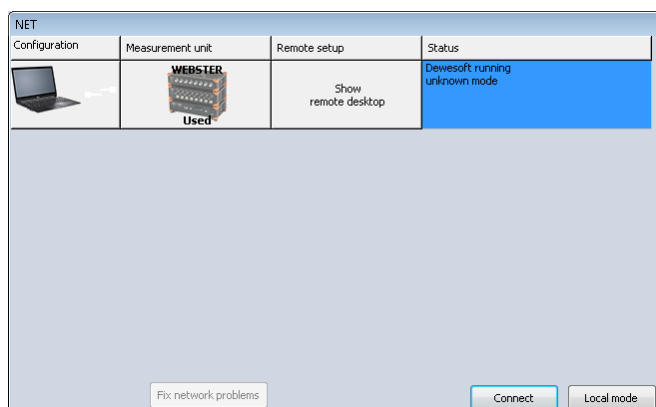


Note that the slave MU called WEBSTER was already found in our example. But if you have not already *configured* and *connected* to an MU, just click the **Add** button and *select* one or more MU(s) in order to add them to the system.

Now click **OK** to close this dialog and you will notice that the basic DEWESoft screen has a new addition to the top bar – a **NET** icon:

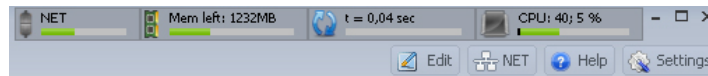


You must click this button and then **connect** to the MU(s):

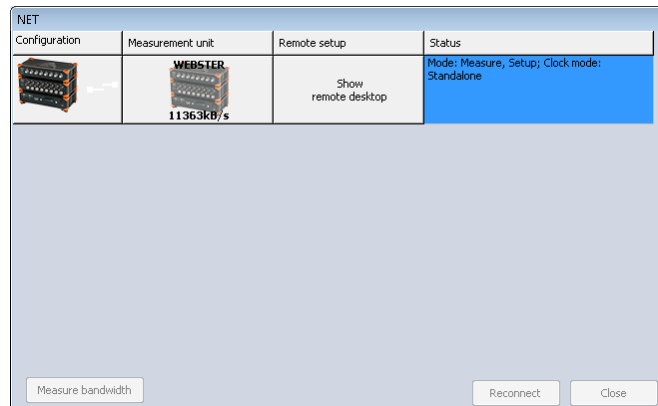


Click the **Connect** button and **DEWESoft** will *connect* to the **MU**. This box will close automatically.

The **NET** icon will *change* slightly. The green *bar* indicate the transmission speed:



Click on **NET** icon to show the connection screen. Clicking the **Measure bandwidth** button in the system will check network *performance*. With a 100 MBit network the transfer speed is about 10 MB/second while Gigabit LAN offers speeds close to 100 MB/second. Please note that this real bandwidth is also limited by system performance.



Notice from the menu - **NET** option that you have several other useful capabilities → see **NET Measurement**.

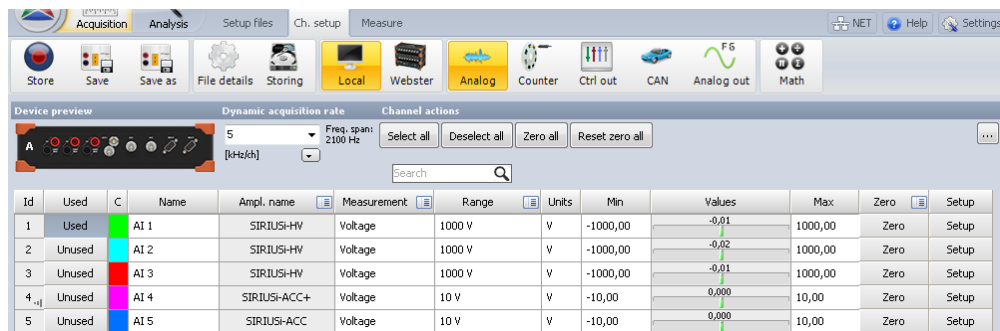
6.2 Remotely Controlling a Slave MU

In this section we are using *only* the **master client** computer to **remotely configure** and **control** a **slave MU** from this master client. The measurement unit is set to **Slave MU mode**, and we have already *connected* to it using the steps from the preceding section. All steps are done on the CLIENT!

We are *not touching* the **MU** at all. It could be a few feet away, on the other side of the building, or miles away. As long as it has a *reliable* network connection to the **client**, we can **control** it from this **client**!

"Local" setup

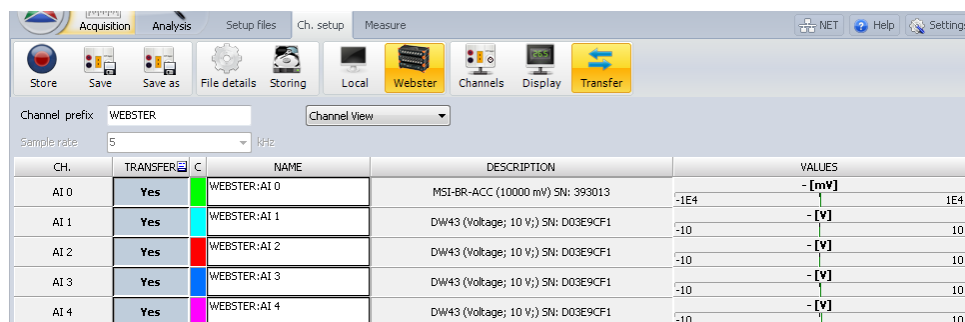
Now click the **Ch. setup** button. If you know what the **DEWESoft Setup screen** normally looks like, you will notice a subtle but important *difference*:



Note the buttons for **LOCAL** and **WEBSTER**. If you have *more* than one **MU**, their *names* will be shown on this bar as well. The local computer is our **master client**, so it does *not have* any real measurement *channels* of its own. However, it still has a **Math** buttons. It is interesting to note that you can *perform math functions* in *real time* on this **client** using *any channels* that are *transferred* from the **MU**'s! You can even combine channels from *more* than one **MU** here in *math channels* – as long as the **MU**'s are *synchronized* !

WEBSTER setup

Click on the **WEBSTER** button so that we can proceed to **set this MU up**.

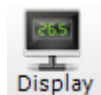


You can see that **WEBSTER** has four channels set up to be **transferred** in real time to the **client**. Of course they will also be **stored** on the **local MU**, because this has been selected by *default* on the *hardware setup* screen, **NET** page.

Note the three buttons on NET toolbar beside selected **WEBSTER**:



- **Channels** setup, activation, and scaling of the actual channels that will be stored



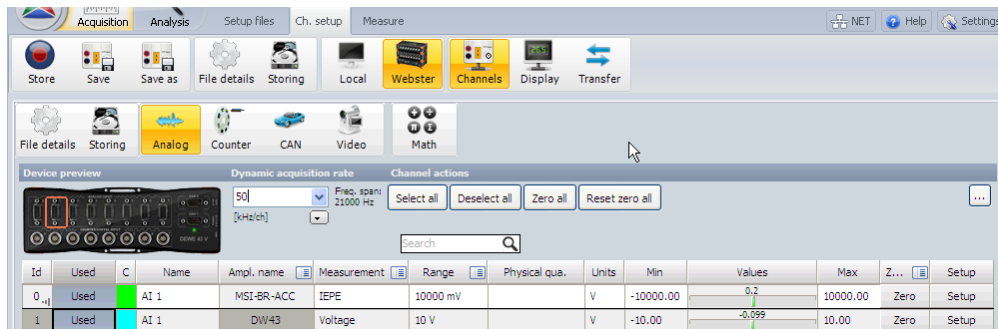
- **Display** to have a *display screen* on the MU for *local* observers



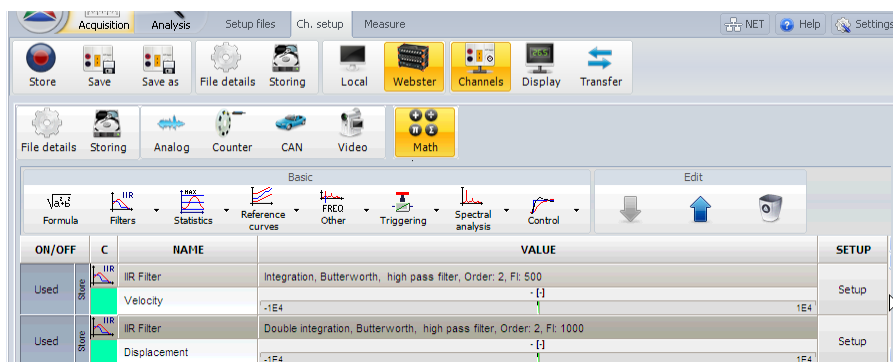
- **Transfer** to set up the *transfer* from the MU

6.2.1 Channel Setup on the MU

This is perhaps the *most critical* step of the three, because it is the **setup**, **activation**, and **scaling** of the *actual* channels that will be *stored* on **WEBSTER**. So let's start there. Click the **Channels** button on NET toolbar beside selected **WEBSTER**.



On this screen you are doing exactly what you would be doing if **WEBSTER** was a *stand-alone MU* activating *channels* with the **Used / Unused** buttons, scaling them using the **Set ch. #** buttons, and so on. You can also *set the dynamic and reduced sample rates*, choose a *filename*, and more. In this example, **WEBSTER** is a computer with *2 analog input channels*. In addition, two *Math channels* have been *created*:

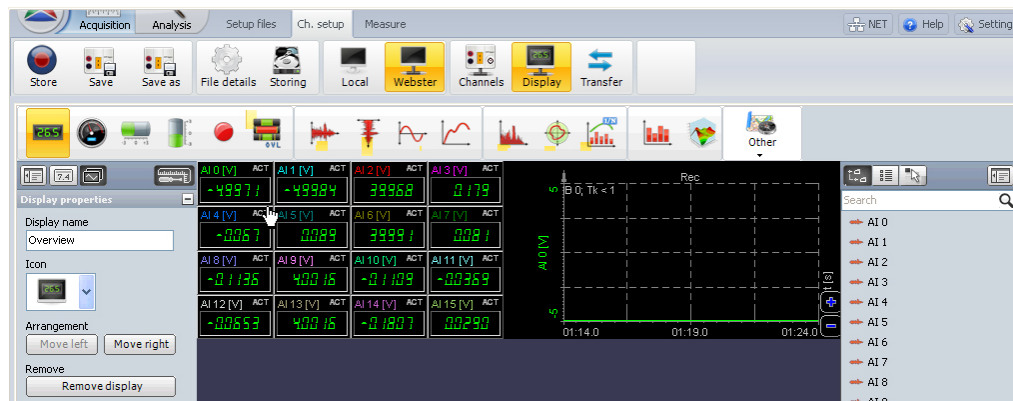


Therefore it will be possible to transfer up to **4 channels** to the **master client**.

6.2.2 Display Setup on the MU

This is only important if you want to have a **display screen** on the **MU** for *local observers* to *see*. If there is *no one looking* at the local display on the **MU** (perhaps it is in a remote location without any people near it), then you can skip this step.

But if you want a local display on the **MU**, click the **Display** button on NET toolbar beside selected **WEBSTER** and then **set up the screen** as you desire, using the normal **DEWESoft** methods and conventions for *screen design*.



It is really important that the CLIENT computer have a display which has MORE RESOLUTION than the MU's! If your MU's have 1024x768 screens, your client should have the next size up or greater, else you may run into trouble seeing some of the screen objects near the bottom when remotely controlling MU's from the client.

The display above is the one that will *appear* on the screen of the remote MU called WEBSTER! It is *not* the display that you will see here on the *client*.

6.2.3 Transfer Setup of a MU

Finally we will set up the **transfer** from the MU. What does this mean? Transfer "*which channels* will be **sent** across the network *during recording*, for display on the *client*".

That is the entire scope of what transfer means. It has *no effect* on the actual Storage of data on the client (assuming that local storage is *enabled* – the default and highly *recommended* setting).

Therefore, you can have *multiple* MU's, each with dozens or even *hundreds* of *channels*, and transfer only a **few** channels – or even *no* channels – to the *client*. It will have *no effect* on the storage of *all channels* on the MU's! This is important to understand.

Due to **bandwidth limitations** of any network, we recommend being prudent about transferring channels – keep the bandwidth in mind and *select only* those channels that you *really need* to **see** on the *client* in order to *monitor* and *control* the test.

In this case we will transfer all four channels. Note that all four channels are set as **Yes** in the **Transfer setup** screen:

| CH. | TRANSFER | C | NAME | DESCRIPTION | VALUES |
|------|----------|---|--------------|------------------------------------|------------|
| AI 0 | Yes | | WEBSTER:AI 0 | MSI-BR-ACC (10000 mV) SN: 393013 | - [mV] 1E4 |
| AI 1 | Yes | | WEBSTER:AI 1 | DW43 (Voltage; 10 V;) SN: D03E9CF1 | - [V] 10 |
| AI 2 | Yes | | WEBSTER:AI 2 | DW43 (Voltage; 10 V;) SN: D03E9CF1 | - [V] 10 |
| AI 3 | Yes | | WEBSTER:AI 3 | DW43 (Voltage; 10 V;) SN: D03E9CF1 | - [V] 10 |
| AI 4 | Yes | | WEBSTER:AI 4 | DW43 (Voltage; 10 V;) SN: D03E9CF1 | - [V] 10 |

6.3 Measurement

In this section we can see procedures to **control** the **Acquisition** from the **Client**:

- **Creating a Display on the CLIENT**
- **Storing Data on the MU(s)**
- **Transfer Stored Data to the Client**
- **NET menu option**

Creating a Display on Client

Before you begin storing, you may want to **set up** the **local** display. In previous steps you may have configured the display of one or more **MU**'s, but you probably want to see data here, too!

You certainly know how to do this. Use the *Overview*, *Scope*, *Recorder* (et al) buttons at the top of your own screen here on the **Client** to create displays with *any combination* of *channels* from *any* and *all* **MU**'s!

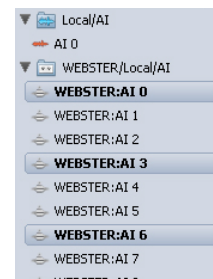
As mentioned previously, all **MU**'s must have a **SYNC** method in place in order to ensure these three things:

- truly **synchronized** *data files* from *multiple* **MU**'s
- ability to **display** *channels* from more than *one* **MU** on the *client*
- ability to **create math channels** on the *client* with *channels* from more than *one* **MU**

Note that the **CHANNELS** list is now showing *channels* with the "name" of the **MU** that they come from automatically. This is so that you know the *source* of *every channel* in an easy and convenient way.

In our example, the name of the **MU**, "**WEBSTER**" is *added* in front of the transfer channels from that **MU** by default, as you can see on the right picture.

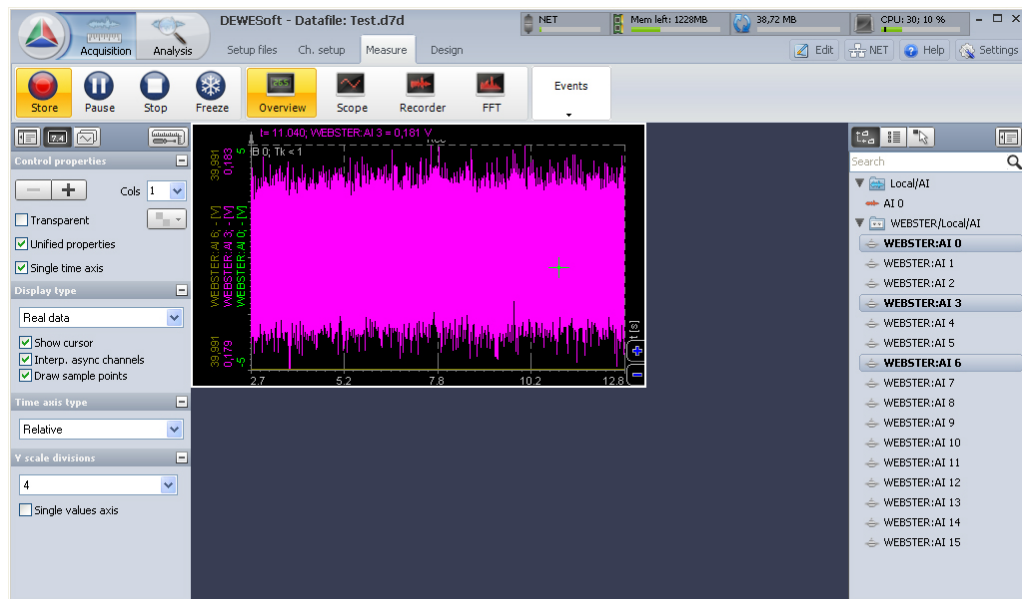
The *channels* from each **MU** will be shown this way *automatically*. This is the only thing that differs from setting up a screen in the standalone mode of **DEWESoft**.



Consult the complete **Display settings tutorial** if you need further help in setting up a display.

Storing Data on the MU(s)

With the **client** and **MU** properly configured, we can now **store data**. Just click **Store** in the toolbar in the normal way.



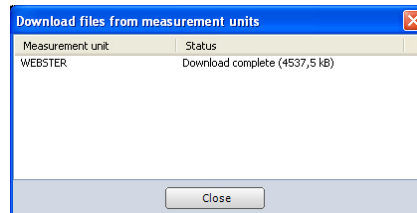
You can see from the "event log" above that we *started* storing, *added a notice event* by clicking the spacebar during the recording and that we *stopped* it.

Transfer Stored Data to the Client

As soon as storing is *stopped*, an important button appears automatically, called **Transfer**:



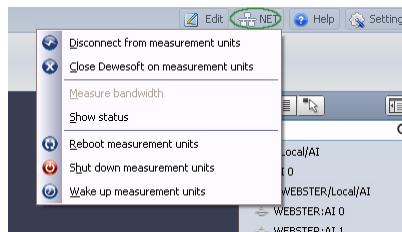
Please click it, and the *data file(s)* from *all MU(s)* that we just *used*, will be **downloaded** to the *client* for you. A "*transfer box*" appears to show the *progress*, and eventual *completion* of the download:



In this case we only had *one MU*, so *only one* file needed to be downloaded.

NET menu option

Click the **NET** menu option to see the list of options:



Notice from the menu that you have several useful capabilities:

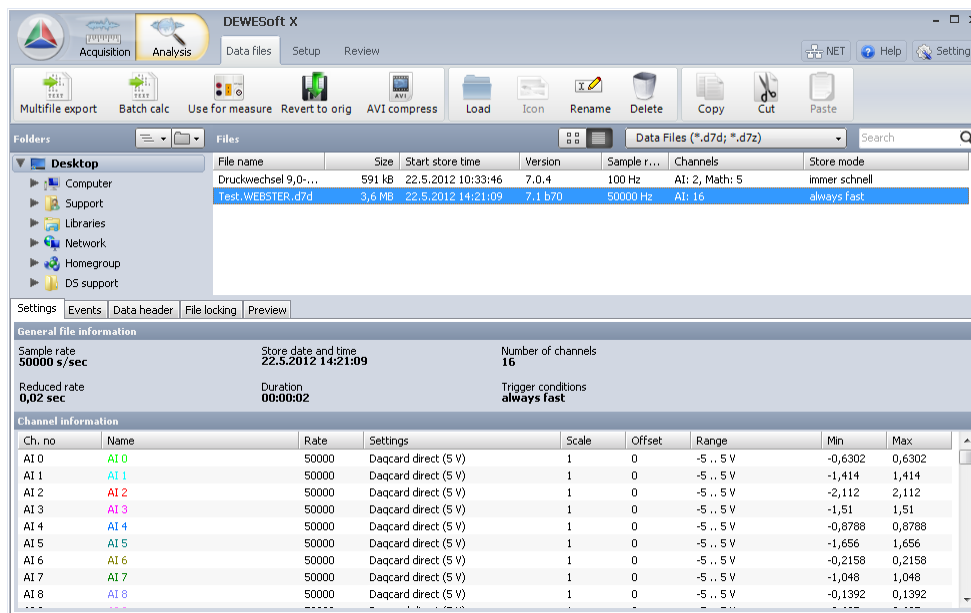
- **Connect / Disconnect from measurement units** - *connect to all MU's / releases the connection*
- **Close DEWESoft on measurement units** - *closes the DEWESoft application on all MU's*
- **Measure transfer speed** - *measuring the bandwidth (transfer speed) between the MU(s) and this client*
- **Show status** - *(displays window with current status of all measurement units)*
- **Reboot measurement units** - *reboots the MU computers (useful if they have crashed or hung up)*
- **Shut down measurement units** - *shuts them MUs (requires ACPI power system on the MU's)*
- **Wakeup measurement units** - *starts MUs (requires 'Wake-up on LAN' option enabled on the MU's)*

6.4 Analysis

Replaying Captured Data

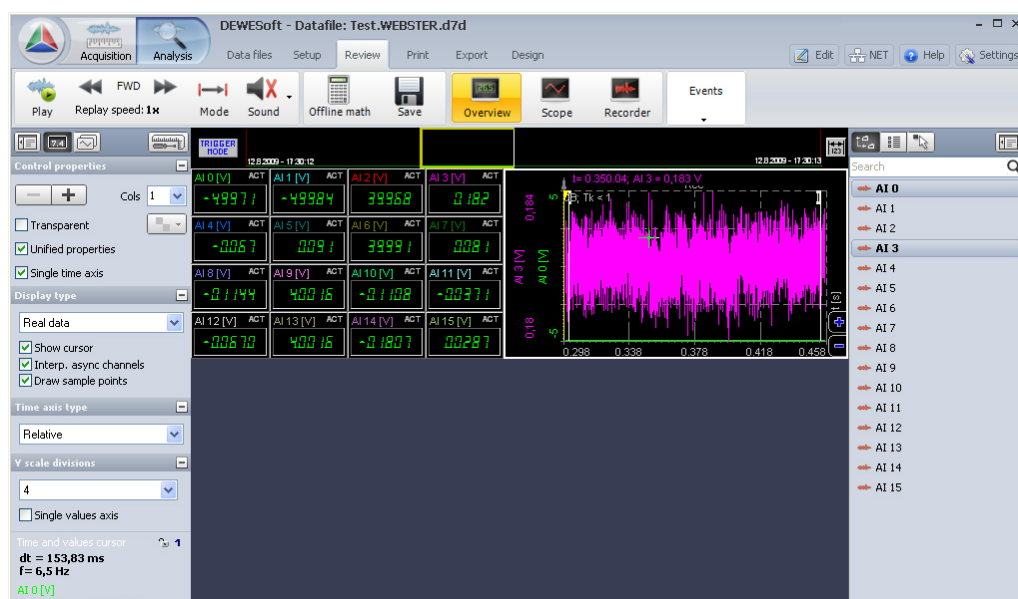
Once the captured data files are *downloaded* to the client, you can **replay** them there.

Click the **Analysis** button and *locate* any *transferred* files that you have. Notice that we also put the *name* of the **MU** into the *filename* by default, so that you can see that this file came from a MU called **WEBSTER**:



The filename was set to the **test**. So the name shown here is: **Test.WEBSTER.d7d** (*.**d7d** = DEWESoft7 datafile).

Double-click it to **open** and use the normal tools for analyzing, reviewing, printing, and more.



Index

- B -

Basic tutorials 2
 display settings 11
 post processing 16
 simple measurement 2
 storing strategies 20

- C -

CAN bus acquisition 109
 channel setup 109
 measurement 112

Combustion analysis 152
 analysis 162
 channel setup 152
 combustion setup 154
 measurement 161

Counter measurement 77
 counters in automotive applications 96
 encoder 83
 event counting 77
 frequency / super-counter 92
 period and pulsewidth 88
 sensor mode 95

- D -

Display settings 11
 Dynamic signal analysis tutorials 129
 human vibration 142
 order tracking 147
 sound level 129
 torsional vibration 134

- E -

Encoder 83
 X1, X2, X4 modes 84
 zero pulse 86

Event counting 77
 gated event counting 80
 simple event counting 78
 up/down counting 82

Exporting recorded data 9

- G -

GPS acquisition 114
 channel setup 115
 measurement 116

- H -

Human vibration 142
 input channel setup 143
 measurement 146
 output channel setup 145

- M -

Measurement tutorials 29
 CAN bus acquisition 109
 counter 77
 GPS acquisition 114
 strain gage 63
 temperature 46
 vibration 50
 video 101
 voltage and current 30
 voltage/current/digital output sensors 41

Microphone calibration 131

- N -

Networked data acquisition tutorial 163

Networked data acquisition tutorial 163
 analysis 175
 measurement 172
 remotely controlling slave MU 168
 setup measurement units and client 164

- O -

Order tracking 147
 channel setup 148
 measurement 150

- P -

Period and pulsewidth 88
 Period and pulsewidth measurement
 duty cycle 91
 period 88
 pulsewidth 90
 Post processing 16
 Power module tutorials 119
 single phase power measurement 119

- R -

Remotely controlling slave MU 168
 controlling slave MU 170
 display Setup on the MU 170
 transfer Setup of a MU 171
 Reporting recorded data 9

- S -

Sensor correction 126
 Simple measurement 2
 analysis 7
 channel setup 3
 exporting data 9
 first measurement 5
 making a nice screen 7
 reporting data 9

video setup 4

Single phase power measurement
 channel setup 121
 measurement 123
 sensor correction 126

Sound level 129
 channel setup 130
 measurement 133
 microphone calibration 131

Storing data strategies 20

Storing strategies
 always fast 21
 always slow 22
 fast on trigger 24
 fast on trigger, slow otherwise 27

Strain measurement 63, 66
 full bridge measurement 74
 full bridge setup 72
 load cell setup 76
 quarter bridge measurement 68
 quarter bridge setup 67

- T -

Temperature 46
 channel setup 47
 measurement 48

Torsional vibration 134
 rotational order extraction 137
 rotational vibration measurement 136
 rotational vibration setup 135
 torsional order extraction 141
 torsional vibration measurement 140
 torsional vibration setup 139

Triggered storing 38
 fast on trigger 24
 fast on trigger, slow otherwise 27

Triggering
 glitch triggering 35
 triggered storing 38

Tutorials 1

| | |
|--------------------------------------|-----|
| Tutorials | 1 |
| basic tutorials | 2 |
| combustion analysis | 152 |
| dynamic signal analysis tutorials | 129 |
| measurement tutorials | 29 |
| networked data acquisition tutorials | 163 |
| power module tutorials | 119 |

- V -

| | |
|--|-----|
| Vibration | |
| analysis | 61 |
| channel setup | 53 |
| Math setup | 57 |
| measurement | 59 |
| video setup | 59 |
| Video acquisition | 101 |
| channel setup | 102 |
| measurement | 104 |
| video post synchronization | 107 |
| Voltage and current | |
| channel setup | 31 |
| glitch triggering | 35 |
| measurement | 33 |
| triggered storing | 38 |
| Voltage/current/digital output sensors | 41 |
| analysis | 44 |
| channel setup | 42 |
| measurement | 44 |



地址：北京市海淀区小营西路27号金领时代大厦12层
电话：136 1171 664；010-5361 2036
传真：010-5635 3026
网站：www.chinaksi.com
电邮：ksi@chinaksi.com