New features in ibaAnalyzer 5.0

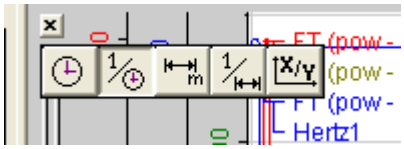
FFT Support

Introduction

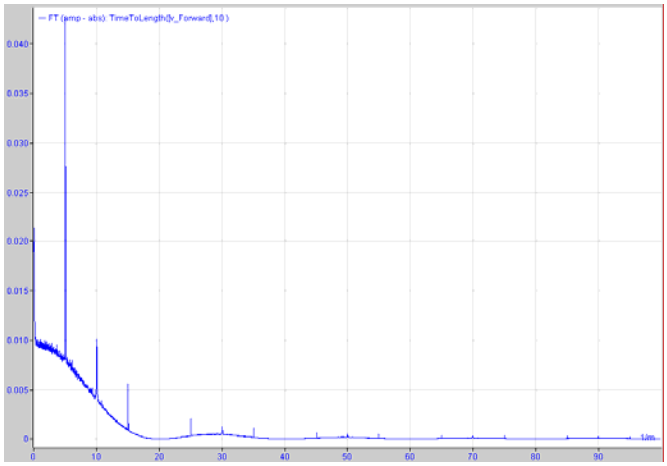
The most significant changes made in ibaAnalyzer were made to support Fast Fourier Transforms of both time and length based signals and visualizing the results. These changes were made to accommodate requests from clients to manipulate the results from the already available FFT functionality. Specifically requests from an ibaBenelux client called StoraEnso triggered the implementation of the massive FFT support. Included screenshots and accompanied DAT- and PDO files are the intellectual property of this company, so please *neutralize* any data before using them in official documentation.

Frequency and Inverse Length Axes

To the left of a graph just under the close graph little button is a button with an arrow on it, clicking it pops up more buttons where you can select the axis mode:



The fourth button is new; for signals that can be displayed length based, it is now possible to view an FFT representation where the FFT is done over length, so the unit of such a graph is 1/m (or 1/inch if the inch would be given as metric unit). Such representations are useful when one wants to examine phenomena that reoccur periodically over the length of something, for example irregularities in the thickness of a finished aluminum plate. In spike.pdo such irregularity is given every 0.2 meter (5 1/m). Note also the harmonic spikes at every harmonic (multiples of 5 1/m)



FFT functions

As explained in the previous paragraph, you can have FFT displays of your data. However they are only just that, displays. You cannot manipulate the results any further since you have no actual (logical) signal representing the FFT results. Therefore you can create new logical signals with the aid of the following functions you can type in the signal definition window or expression builder dialog.

```
FftAmpl (Expression, Resolution, Window = 0, SuppressDC = false)
```

-`Expression`: The expression you wish an FFT from.
-`Resolution`: The number of desired frequencies.
-`Window`: The window function you wish to apply before applying the FFT, default this is none (=0), although a window is recommended.
-`SuppressDC`, the constant component of an FFT (i.e. the average of the signal) is most of the time quite large and dwarfs out the other components, so you can choose to suppress that component (default false).

Computes the amplitude for each FFT component, i.e. the modulo of the complex number. Contrary to 'normal' FFT (i.e., the ones you get by selecting the Frequency or Inverse Length axes) this takes all samples into account instead off the number of samples in the recorder window (which stands to reason, as Expressions need not to be displayed in the recorder window)

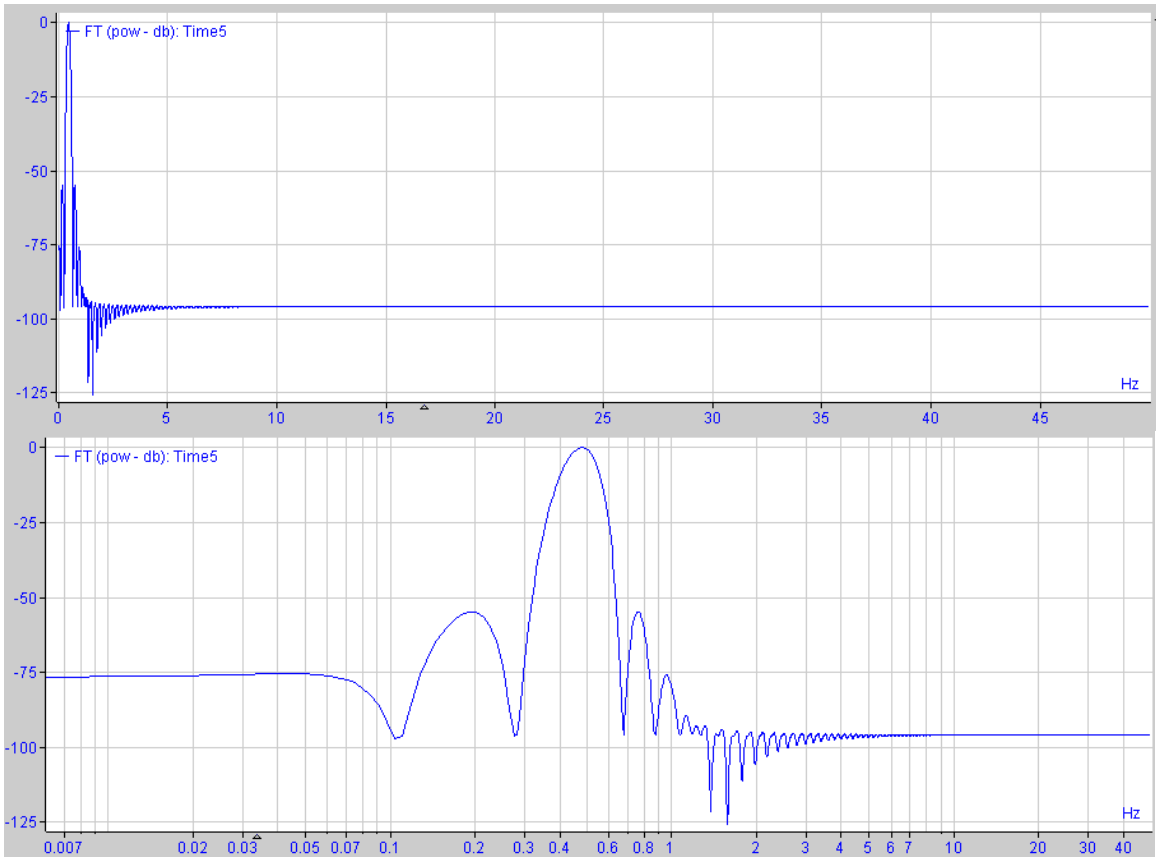
```
FftPower (Expression, Resolution, Window = 0, SuppressDC = false)
```

This is the same function as `FftAmpl` but instead of taking the modulo it gives the square of the modulo, i.e. the square of the real part plus the square of the imaginary part.

While testing it's important to test that any signal derived with these functions is displayed in the correct FFT axis and that functions applied to these also work and are displayed correctly.

Logaritmic scaling

FFTs are sometimes or even most of the time better displayed on a logarithmic scale, You can select this by right clicking on the graph, then selecting Setup (or in preferences for non-PDO specific settings), then selecting the X-axis tab, then selecting the Frequency or 1/Length tab. Following two screenshots show the difference:



(to reproduce screenshots, open scaling.pdo)

Dynamic marker and marker grid

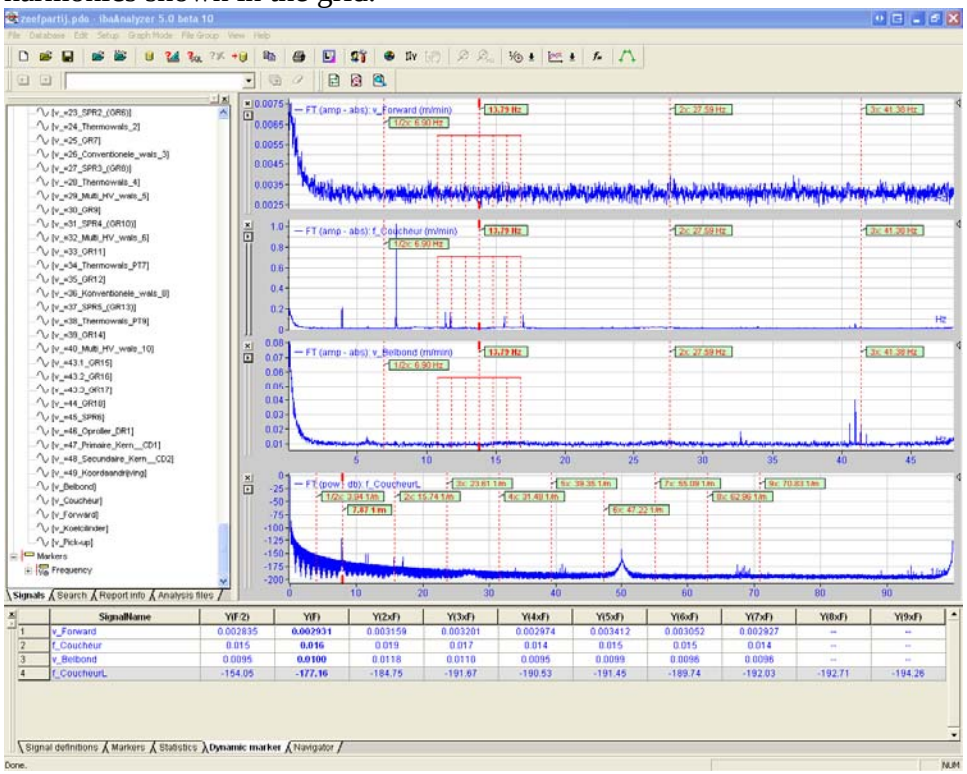
There is a new tab in the signal definition grid called *Dynamic Marker*. Clicking it will display in any FFT graph a marker similar to the two markers from the second and third tab. The Y-values per signal will be displayed as also the Y-values of the *harmonics* (i.e. constant multiples or fractions) if they are available (otherwise `--` is depicted). You can drag the marker around by the thick ends at the top and bottom and to the left of its label. There are actually two markers, one for frequency based and one for inverse length based signals. Clicking the dynamic marker's label (little green box with X-value in it) will display side band markers and/or harmonic markers.

Harmonic markers are located at the harmonics and look like the dynamic marker except that they have no thick ends, their labels also are prefixed with what harmonic it is (e.g. 2x, 3x, 1/2x, …).

Sideband markers are located at equidistant intervals of the dynamic marker, are only two thirds the length of any other marker and are connected to each other with a horizontal line at their tops.

The number of sideband and harmonic markers and for sideband the distance between them can be altered by right clicking the graph, selecting setup, selecting the X-axis tab and selecting Frequency or 1/Length tab. (Or in preferences for non-PDO specific settings)

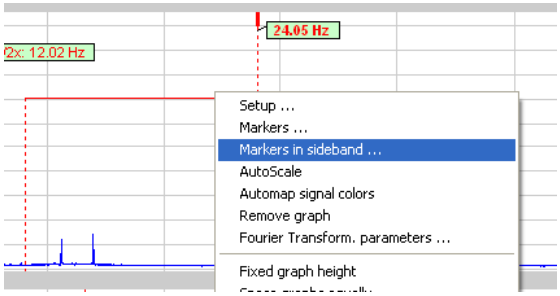
Altering the number of harmonics (below or above) will also reset the number of harmonics shown in the grid.

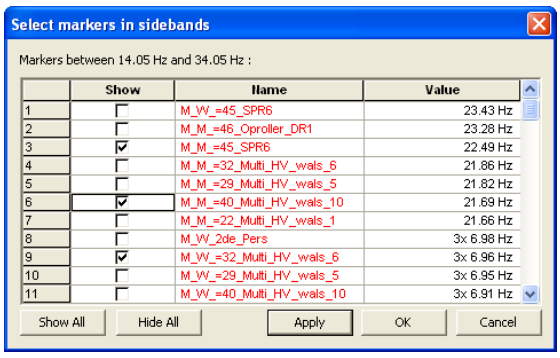


As said, the harmonics and sidebands are displayed by clicking the dynamic markers label, this is cyclic:

- clicking once shows harmonics
- clicking twice shows harmonics and sideband
- clicking thrice shows only sideband
- At the fourth click you're back at the initial state, i.e. neither harmonics nor sidebands are shown.

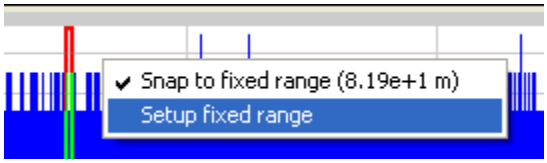
When right clicking on an FFT graph and the dynamic marker tab is selected, you have the option to select *markers in sideband*. This opens a new dialog where you can activate or deactivate the X-axis markers (X-axis markers are yet another type of markers explained later in this document) in the range from the leftmost to rightmost sideband markers. X-axis markers, whose harmonics fall in that range, are also included. The markers are sorted from closest to furthest from the dynamic marker.



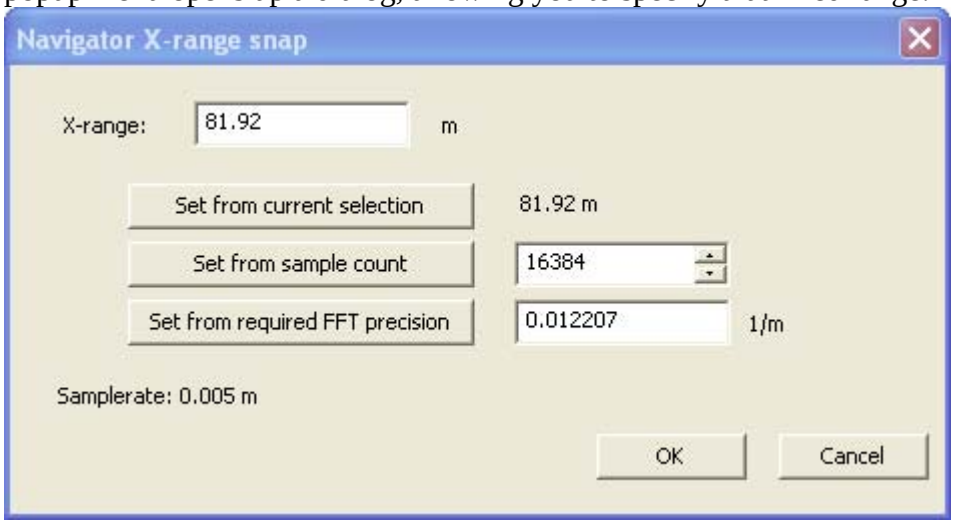


Navigator

The navigator (last tab in signal definition window) is also adapted to further support FFT operations. When showing a signal on an FFT axis, the FFT is actually computed with the number of samples in the current zoom (rounded to the nearest power of two). Since you can select the zoom in the navigator, it makes sense to be able to manipulate the navigator in such way that the navigator rectangle has a width appropriate to have a desired number of samples. Right clicking on the navigator window shows following popup menu:



If you check the first item in the popup menu, you can no longer alter the width of the navigator rectangle (it is snapped to a fixed range). Selecting the second item in the popup menu opens up a dialog, allowing you to specify that fixed range.



You have two options

- Type in the fixed range in the first edit field.
- Calculate the fixed range from other parameters:
    - Set it from the current selection, i.e. the width of the current navigator rectangle.
    - Specify the number of samples, the width of the navigator rectangle is calculated to accommodate that number of samples.
    - Specify the required FFT precision, the width of the navigator rectangle is calculated so the number of samples is sufficient to have FFT data at every multiple of the given precision between minimum and maximum frequency (which are in turn specified in the Setup of the FFT axes, scaling options)

Note that any value you type can be adjusted so that the number of samples is a power of two or a minimum number of 128 samples is respected.
Also note that only the fixed range is persistent, any values you used to calculate the fixed range are reset the next time you open this dialog.

Zooming

Holding shift key while zooming

… will stretch your zooming rectangle to the height of the graph, just like in PDA

Wheeling the mouse while the mouse is over an Y- or X-axis

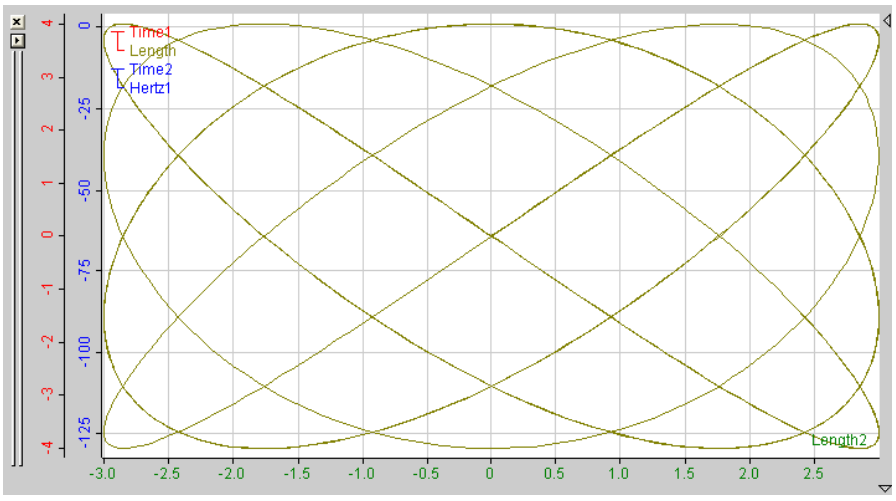
… will zoom in or out in resp. Y or X direction.

XY-view

It is now possible to represent length based signals in an XY view. Although not much can be said about this it took in my opinion considerable programming effort and should take considerable testing effort
Suggested things to test:

- In no circumstance should another signal then a time or length based serve as X-axis, nor should a frequency or inverse length based signal ever be displayed in an XY-view. I've chosen this so because I cannot see any practical purpose for this.
- Time and length based signals should not be mixed.
- The XY mode button should not be available to select if there are less then two compatible signals in the graph.
- Switching from XY to other mode or back should never produce an empty graph.
- Dragging signals from and to XY views and the X axis should always work, if necessary change the mode (if not enough valid and compatible signals) and certainly not cause crashes.

Accompanied with this document is meters.pdo; a PDO with all types of signals (sines) in which besides generating beautiful Lisajous figures you should also be able to extensively test this new feature.



Markers

Classic markers

The *classic* markers are the ones visible when selecting the second or third tab in the signal definition window. When pressing shift while dragging one of the two markers, the other marker will move along so the distance between them is kept constant.

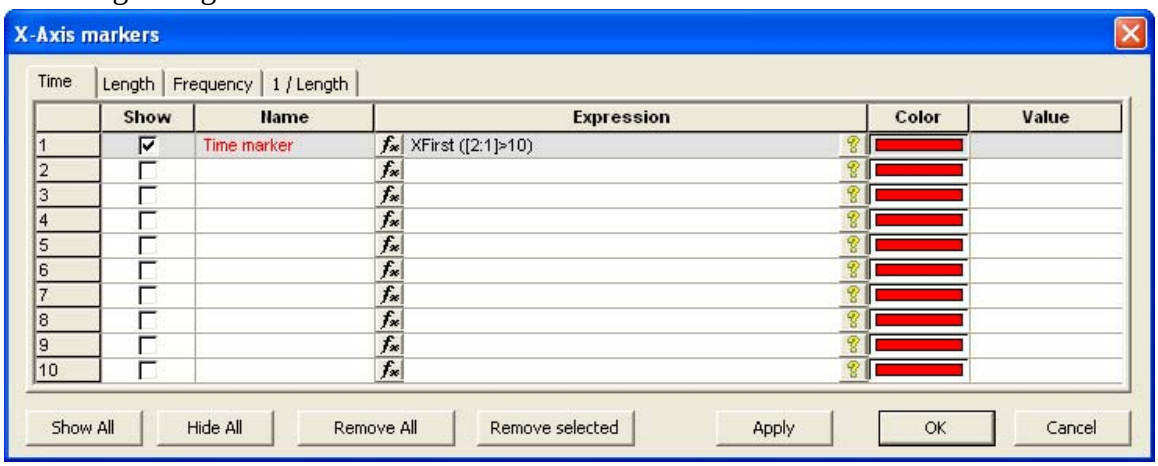
Dynamic markers

See discussion on FFT

X-axis markers

For each graph and each Axis of that graph (Time-, Length-, Frequency- or Inverse Length based) a number of additional markers can be defined. These are represented as thin vertical lines in the graph with a selected color (default 1st color, depending on settings this is red). A label containing the marker name and its value is also depicted. The idea is to mark special X-values in the graph. The markers are saved in the PDO.

To define markers you can right click on a graph, and select *Markers*, to get the following dialog:

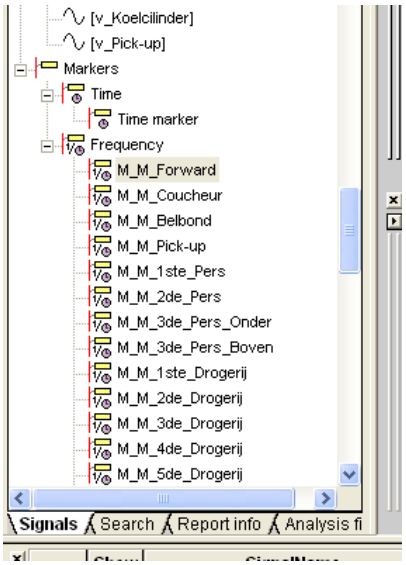


- First column you can select if the marker is actually shown or disabled momentarily.
- Second column you can name the marker.
- In the third column you can type the expression that will be evaluated to place the marker, mind that this expression is *navigator controlled*, meaning that when evaluating the expression only the part of the signal in the red rectangle of the navigator will be evaluated. E.g.: the expression in the screenshot will result in a

marker placed at the first value larger than 10 inside the navigator rectangle. Note that the expression control has two buttons at its sides (like the expression control in the signalgrid). The left button opens the expression builder dialog. The right button allows you to check any errors in your expressions.

- In the fourth column you can select the color of the marker
- If the position for the marker can be evaluated, that value will be placed in the fifth column (after pressing the Apply button).

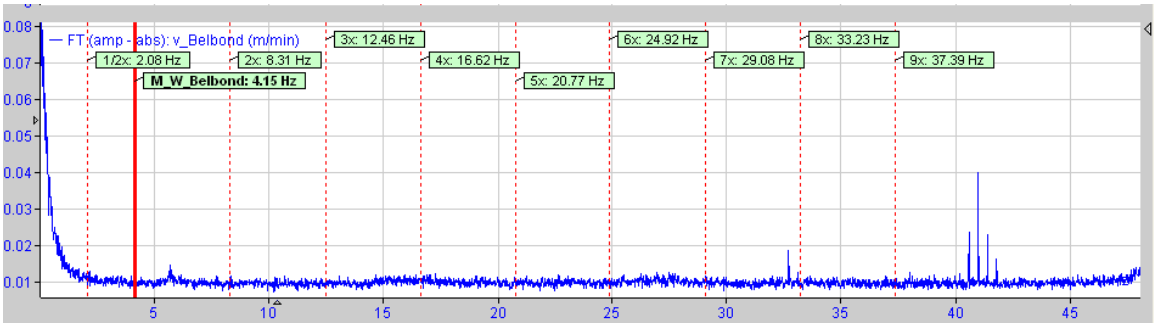
X-axis markers cannot be repositioned unless by changing their expressions, but you can however drag markers from one graph to another provided the graphs have the same X-axis base. You do this by dragging the marker's label, the mouse cursor will change appropriately.



All markers are displayed in a tree below the signal tree. You can drag the markers from there to a graph provided that the X-axis base of the marker and the graph match. You can also drag a marker out of a graph, i.e. not on another graph, which will result in the marker being removed from the graph. However the marker will not be removed from the signal tree.

Besides the markers dialog, you can also define markers by right clicking, selecting Setup and then the X-Axis tab. In either the marker dialog or this tab you can right click on the markers, where you can select to import or export the markers respectively from and to a text file.

Markers on axes that are either frequency or length based, have the special ability that when you click on the marker's label, harmonic markers are shown. The number of markers is the same that you selected for the dynamic marker.



Graphics

Overlay

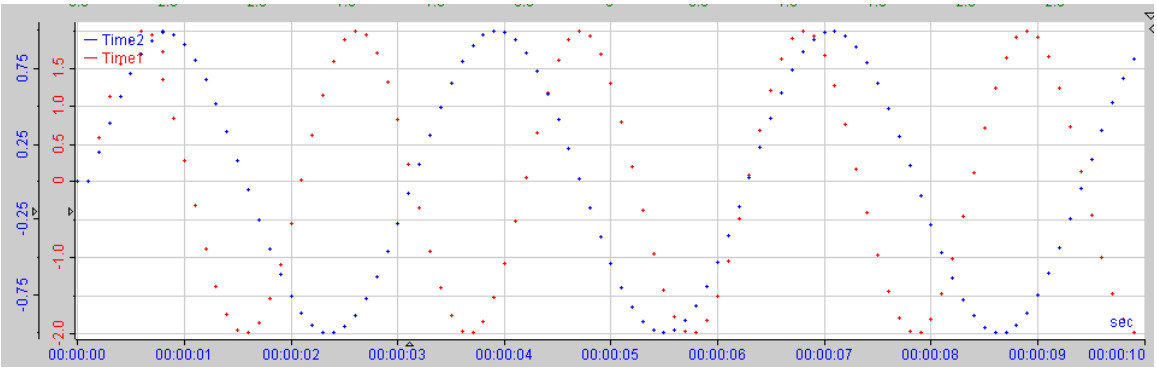
All types of markers, legends, units and mouse trackers on the graph are painted on a transparent layer that is on top of the graph called *the overlay*. This is so that when any of that data changes, the underlying graph needs not to be repainted. This feature caused problems with ATI graphic cards that are notoriously bad with the alpha blending needed to mix the overlay with the original graph.

Double buffering

All graphs are now first painted to a memory buffer before being copied to the screen. This results in no more flicker when redrawing graphs. Also the problems with the overlay and ATI graphic cards seem to be adequately fixed by this.

Points only viewing mode

You can select to display graphs as disconnected points, one for each sample. (right click on the graph and select this option in the popup menu)



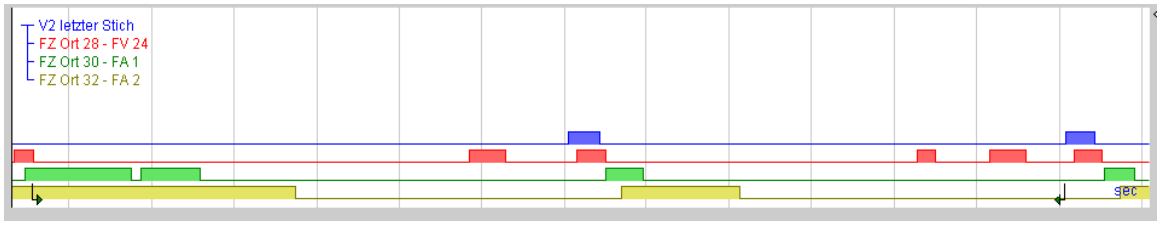
Triggers and file separators

… can now be turned off by selecting so in the *2D View* tab (either preferences for general settings or right clicking on the graph for PDO settings)

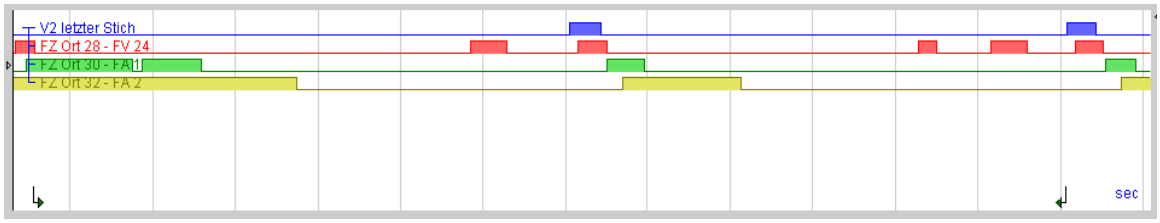
Digital signals: align on top

Again in the *2D View* tab, you can now select to display digital signals from the top of the graph, in sequence with the signal names in the legend.

Before:

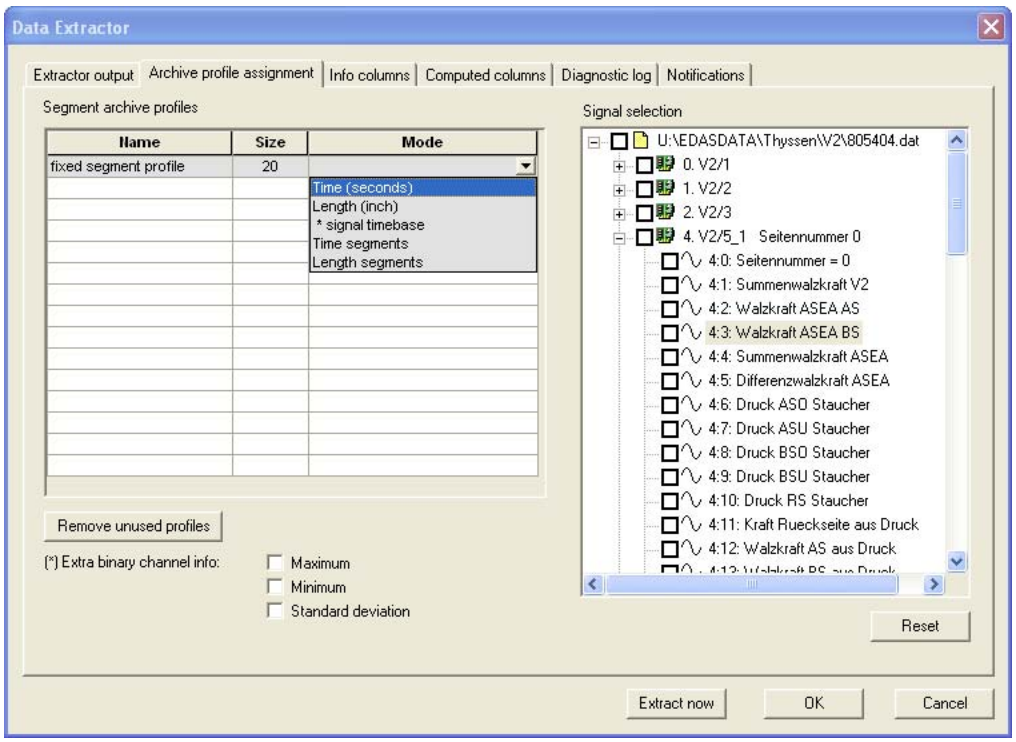


After “align on top”



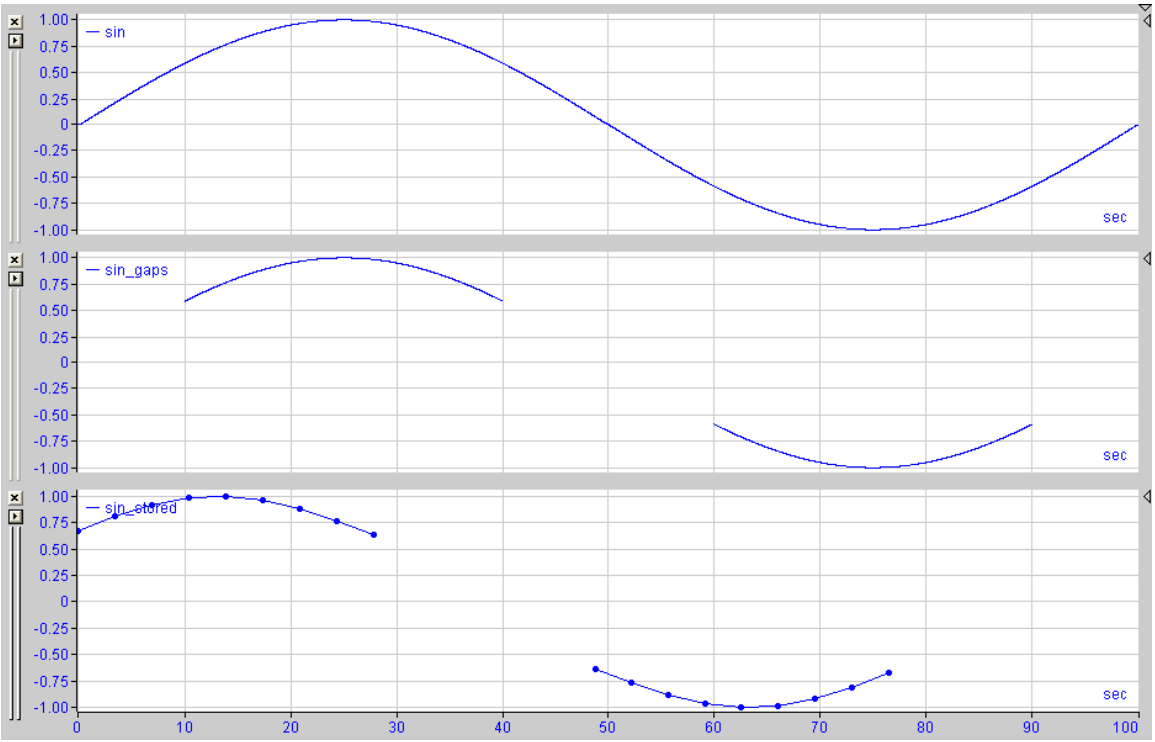
Data Extractor

Fixed number of segments profile



There are two new options when selecting the archive profile: *Time segments* and *Length segments*. When selecting either of these, a fixed number (*Size*) of records will be extracted to the database. If the start or end of the signal is invalid, no records will be

chosen from these parts (no lesser number of records), if however parts in the middle of the signal are invalid, records will be chosen from the gaps but they will not be extracted (lesser number of records)



(screenshot: sine with gaps in, when stored and queried from database with profile 23 Time Segments, 23 samples where indeed taken between begin and end gap, but samples where lost in middle gap)

Extracted module names

There used to be no direct way to extract the module names to the database. When the database was queried after an extraction in ibaAnalyzer, the queried modules where unnamed (resulting in a question mark for their names). Now by default the module names are extracted and placed in the "`deFile`" table. You can turn off this option by deselecting the option *extract module names* in the third tab (*Info Columns*) of the data extractor dialog. Note that for older PDO files, this option will be deselected by default so that extraction behavior is identical to the old behavior.

Miscellaneous changes

Intellisense

When typing expressions in a grid (markers or signal definition grid and others) or in the expressionbuilder dialog, popup windows will appear allowing you to complete your

expressions without fully typing them, e.g. you only have to type the first letter of a mathematical expression or after typing '[' , a list with available signals appears.

Performance improvements

- Signals are now cached, so that they do not have to be reloaded or recalculated since last time they were needed if no intermediate changes to the signals themselves or their input data have happened.
- The signals tree is build up gradually as nodes are expanded by the user. This greatly improves the speed at witch a file and its signal tree is loaded.

Automatic switching of X-axis base

When modifying a signal expression, if the current graph is no longer appropriate another appropriate base should be selected. The only time an empty graph window should be shown is when there are actual errors in the signal expression.

Navigator

When in the navigator, pressing the left or right arrow keys will move the navigator rectangle to the left or to the right.

Reset option for the integrator function

When integrating an expression with the *Int* function, it is now possible to specify a second argument to that function, so that for each time the new argument evaluates to TRUE, the integration is reset at zero.

Sorting of locigals

Logicals used to be sorted alphabetically on their name. This is no longer so. This was done so that when importing locigals from a text file, their order would remain unchanged from the order they appeared in the text file.

Signal editing

When editing a signal, if one would leave the grid cell (e.g. by switching tabs, or right clicking and adding a row) while editing it, this left the grid in an erroneous state with very weird behavior. This should be remedied.

Also when double-clicking on the space between two column headers, the column to the left should resize to fit exactly the largest horizontal text size in the grid header itself or the cells below it.

Tooltips

In the signal tree, when hovering over a signal of a DAT file that was created with *comment* fields, these comments are displayed in a multiline tooltip.

Automation

After registering ibaAnalyzer by typing

```
ibaAnalyzer /regserver
```

ibaAnalyzer can serve as a COM component. Automation functionality is provided.
In a program you can create an instance of ibaAnalyzer as a com-component and then following functions are provided:

BSTR IbaAnalyzer::GetVersion ()		
Gets version of ibaAnalyzer		
Input arguments		none
Return value		Return the version of ibaAnalyzer as a string

BSTR IbaAnalyzer::OpenAnalysis (BSTR filename)		
Opens an analysis		
Input arguments	filename	absolute path of the .PDO file to open
Return value		none

BSTR IbaAnalyzer::CloseAnalysis ()		
Closes the currently open analysis		
Input arguments		none
Return value		none

BSTR IbaAnalyzer::GetLastError ()		
On error; get the error message of the last error		
Input arguments		none
Return value		the error message

void IbaAnalyzer::OpenDataFile (int index, BSTR filename)		
Opens a DAT file and places it at a given index		
Input arguments	index filename	index where to insert the DAT file (starting from zero) absolute path of the .DAT file to open
Return value		none

void Analyzer::CloseDataFile (int index)		
Closes the current DAT file at a given index		
Input arguments	index	index of the DAT file to close
Return value		none

void Analyzer::CloseDataFiles ()		
Closes all DAT files		
Input arguments		none
Return value		none

void IbaAnalyzer::AppendDataFile (int index, BSTR filename)		
Appends a DAT file to an already open DAT file at a given index		
Input arguments	index filename	index of the open DAT file to append the DAT file to Absolute path of the .DAT file to open
Return value		none

int IbaAnalyzer::RunSqlQuery (BSTR sql, BSTR sync)		
Performs an Sql instruction on the open database		
Input arguments	sql sync	the Sql instruction to perform field to *synchronize*, if this field is identical for several rows, these rows are merged in one virtual file.
Return value		the number of entries in the resulting set

int IbaAnalyzer::RunSqlQueryFromFile (BSTR filename, BSTR sync)		
Performs an Sql instruction read from a file on the open database		
Input arguments	filename sync	Absolute path of the file where the Sql instruction is to be read. field to *synchronize*, if this field is identical for several rows, these rows are merged in one virtual file.

BSTR IbaAnalyzer::GetQueryResult (int index)		
Get the name of the virtual file corresponding with the index in the resulting set of a performed SQL query, which can be used to open the virtual file with OpenDatFile()		
Input arguments	Index	index in the resulting set of the performed SQL query (starting form zero)
Return value		name of the virtual file which can be opened with OpenDatFile()

void IbaAnalyzer::Extract (BOOL fileExtract, BSTR output)		
Performs an extraction to a database		
Input arguments	fileExtract	if TRUE, performs an extraction to a file instead of a regular database
	output	absolute path of the file to extract to, ignored if fileExtract is FALSE
Return value		none

void IbaAnalyzer::Report (BSTR output)		
Creates a report		
Input arguments	output	absolute path of the report file, its extension determines the type of report generated, if this parameter is an empty string, the report is generated and printed on the default printer.
Return value		none

void IbaAnalyzer::Print ()		
Prints out the current open PDO file to the default printer		
Input arguments		none
Return value		none

A small example C# project using this COM interface is accompanied with this document (ibaAnalyzerTest.zip)