

IbaAnalyzer 5.19.0 new functionality description

Macros

Description

In the current version of IbaAnalyzer it is possible to define a '*macro*'. A macro is an IbaAnalyzer function that you can create and define yourself.

You do this by specifying what the macro needs to calculate through IbaAnalyzer expressions using regular IbaAnalyzer functions or other macros. The expressions you can use in a macro differ from regular IbaAnalyzer expressions in the sense that in the macro expressions you can refer to placeholder 'input' channels, which will be replaced by actual channels when the macro is used.

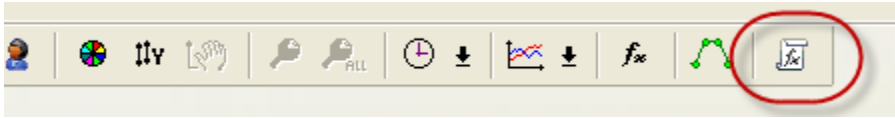
Although theoretically there is nothing you can do with macros that you couldn't have done with regular IbaAnalyzer expressions, macros offer the following advantages:

- When needing to perform the same type of calculations repeatedly on different input signals, the amount of typing can be significantly reduced.
- The complexity of the calculations can be hidden away in the macro definition, allowing you to type shorter and more comprehensible expressions in the signal definitions grid.
- The macros can be stored globally, making them accessible in every analysis. This allows you to build a library of commonly used calculations.
- The macros can be exported and imported, allowing you to share calculations with other IbaAnalyzer users. This is easier than sharing analysis files, which are typically only applicable to specific .dat files.
- New IbaAnalyzer functions can be created with only knowing how to use the existing IbaAnalyzer functions. No programming experience is required.

The macro design dialog

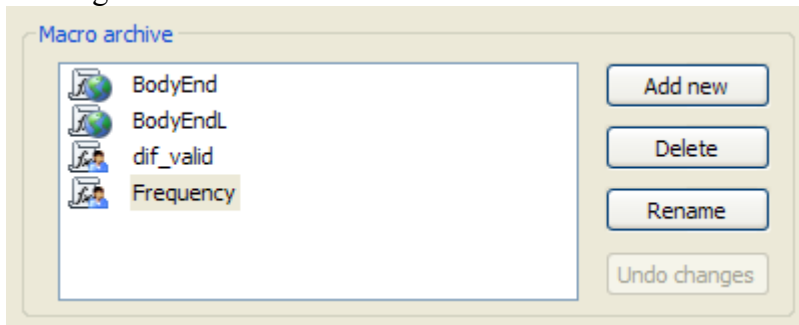
You can create or modify macros through the 'Macro design' dialog.

This dialog can be accessed through the main menu ('Setup' → 'Macro design ...') or by clicking the button in the toolbar.

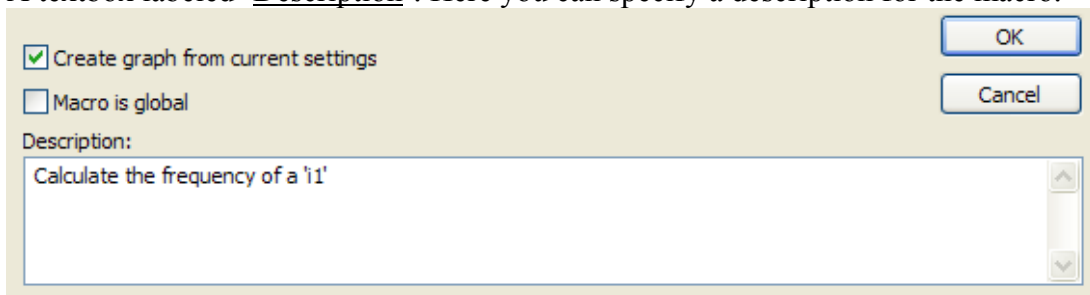


The *macro design* dialog consists out of the following elements:

- Macro archive: Here the created macros are listed. You can add new macros here or load existing macros to modify them. Here you can also rename or delete existing macros.



- A checkbox labeled "Create graph from current settings": By checking this checkbox you can instruct ibaAnalyzer to immediately create a signal that uses the macro. This is similar as to the option with the same name in the filter design dialog. This option will not be available (i.e. it will be grayed out) if you have specified invalid expressions anywhere or if you have omitted specifying example expressions for all mandatory input arguments (See pg. 4).
- A checkbox labeled "Macro is global": Here you can specify whether a macro is 'global' or not. A global macro is always available and can be used at any time in any ibaAnalyzer expression while a nonglobal macro is part of the current analysis and can only be used in expressions belonging to that analysis. You can import and export macro's through the new 'Export settings' tab in the preferences dialog (See pg. 16).
- A textbox labeled 'Description': Here you can specify a description for the macro.



- The ‘Inputs’ part: Here you specify the arguments of the macro. We’ll discuss this part in more detail in a following subchapter (See pg. 4).

Inputs: Number of mandatory arguments: 1

	Show	Name	Example expression	Default	Comment
1	<input checked="" type="checkbox"/>	i1	f_w [17:12]	--	input voltage

- The ‘Intermediate values’ part: Here you can specify expressions to be used as intermediate values in other calculations (either other intermediate values or the result expression). We’ll discuss this part in more detail in a following subchapter (See pg. 5)

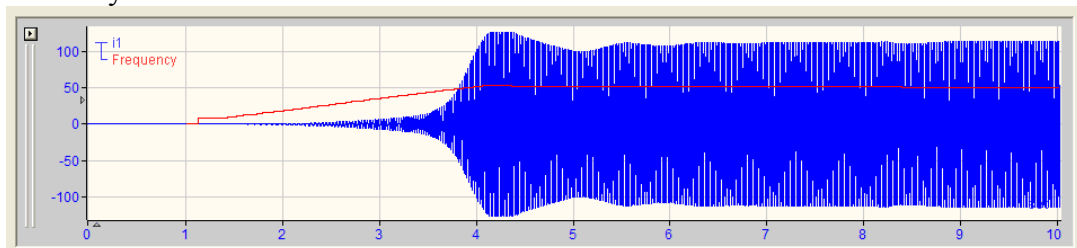
Intermediate values:

	Show	Name	Expression	Comment
1	<input type="checkbox"/>	count	f_w Count([i1],0,0.5,1)	
2	<input type="checkbox"/>	periodstart	f_w Diff([count])<=>0	
3	<input type="checkbox"/>	integration	f_w Int(1,[periodstart])	
4	<input type="checkbox"/>	period	f_w SampleAndHold([integration],[periodstart])	

- The ‘Result’ expression: Here you should specify the final expression corresponding to what should be returned as the result of the macro. The expression you specify should be a valid ibaAnalyzer expression, using ibaAnalyzer functions or other macros. If you use other macros, be careful not to create a circular reference (i.e. a macro calling a macro that eventually calls again the initial macro) or your macro will not work. In the *Result* expression you can refer to input arguments or intermediate values by putting their name between square brackets (e.g.: [input1]) similarly as you can refer to other channels in regular ibaAnalyzer expressions. In the *Result* expression you can also refer to other channels (i.e.: signals of the loaded .dat files, logical signals, trend query results or expressions in the signal grid) like you can in regular ibaAnalyzer expressions. This is however not recommended since your macro will not work if those channels are no longer present.

Result: f_w 1/[period]

- The ‘Preview’: If you have specified valid example expressions for all mandatory input arguments (See pg. 4), you can view the example inputs, the intermediate values and the macro result applied to the example inputs in this graph. You can drag and drop the signals between different Y-axes, auto scale X and Y axes, zoom in and zoom out and pan similarly like you can for a graph in the ibaAnalyzer recorder window.







Inputs

In the ‘*Inputs*’ part of the macro designer you have firstly a spinner labeled ‘*Number of mandatory arguments*’. The number you can specify here is minimum 1 and maximum the total number of input arguments. If the amount of mandatory arguments is less than the total number of input arguments, the resulting macro will have optional arguments similarly as in some of the regular ibaAnalyzer functions (e.g.: *FftInTime* where the last 5 arguments of the 8 arguments in total are all optional.)

Secondly there is a grid present where you can specify the input arguments of the macro. The grid has the following columns:

- Show: If you have provided a valid ‘*Example expression*’ you can have it depicted in the preview by checking the checkbox in this column. If you haven’t provided an example expression but the argument is optional, the value specified in the ‘*Default*’ column will be depicted if you check this checkbox.
- Name: Here you can specify the name of the argument. This name must be unique (i.e. differ from other argument names or intermediate value names).
- Example expression: Here you can specify an example expression whose evaluation will be used as the argument to test the macro. This column is optional, however when this is a mandatory argument and the sample expression is not present, the argument, any intermediate values referring to this argument and the result expression cannot be shown. Nor can the option ‘*Create graph from current settings*’ be selected. The example expression is a regular ibaAnalyzer expression and cannot refer to other arguments or intermediate values.
- Default: For the optional arguments, a numeric constant must be specified here so that when the macro is used with the optional argument omitted, this value is used to evaluate the macro.
- Comment: Optionally you can provide a comment or description for the argument.

Next to the grid there are four buttons:

-  : This button adds a new argument to the macro by inserting a new line to the bottom of the grid.
-  : This button removes the argument corresponding to the currently selected row by removing the row.
-  : This button moves the currently selected row up one row. The corresponding arguments of the currently selected row and the row above it then swap positions in the macro argument list.
-  : This button moves the currently selected row down one row. The corresponding arguments of the currently selected row and the row below it then swap positions in the macro argument list.

Intermediate values

The '*intermediate values*' part of the macro design dialog consists out of a grid with the following columns:

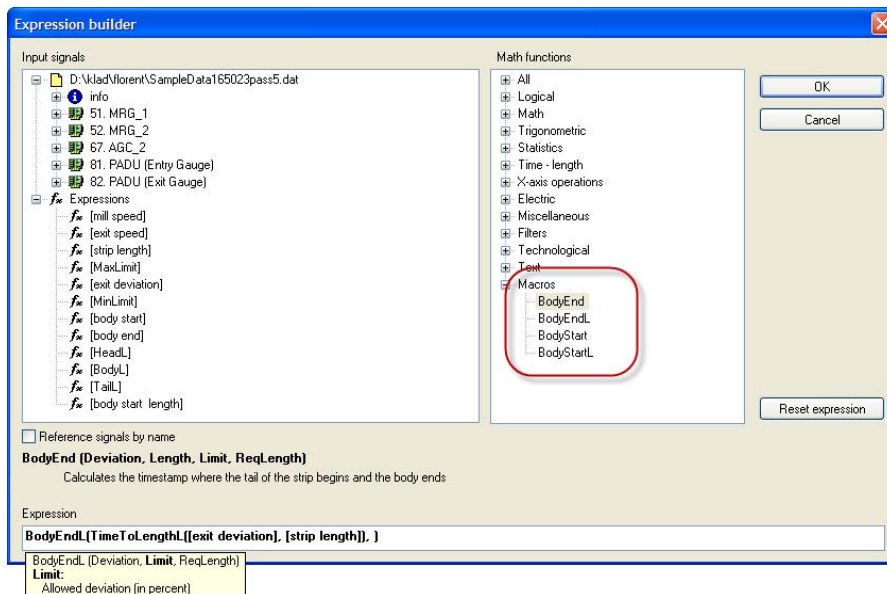
- Show: You can have the intermediate value depicted in the preview by checking the checkbox in this column. For the intermediate value to be actually depicted the following conditions must also be fulfilled:
 - The specified '*Expression*' must be valid.
 - If the specified expression refers to mandatory input arguments, valid example expressions must be specified for those arguments.
 - If the specified expression refers to other intermediate values, the previous two conditions must also be fulfilled for those intermediate values.
- Name: Here you can specify the name of the argument. This name must be unique (i.e. differ from other intermediate value names or input argument names).
- Expression: Here you can specify the expression that will be used to evaluate the intermediate value. The same conditions that apply to the *Result* expression also apply to the intermediate value expression, i.e.:
 - The expression you specify should be a valid ibaAnalyzer expression, using ibaAnalyzer functions or other macros. If you use other macros you should however be careful not to create a circular reference (i.e. calling a macro that eventually calls again the macro this intermediate value belongs to) or your macro will not work.
 - You can refer to input arguments or other intermediate values by putting their name between square brackets (e.g.: [input1]) similarly as you can refer to other channels in ordinary expressions.
 - You can refer to other channels (signals of the loaded .dat files, logical signals, trend query results or expressions in the signal grid) like you can in ordinary expressions. This is however not recommended since your macro will not evaluate if those channels are no longer present.
- Comment: Optionally you can provide a comment or description for this intermediate value.

Next to the grid there are the same four buttons as next to the inputs grid, having the same function for this grid.

Note that the order in which you specify the intermediate values in the grid does not matter for the resulting evaluation of the macro. An intermediate value expression can refer to another intermediate value specified after it in the grid. You should however be careful not to create a circular reference (i.e. an intermediate value expression referring to another intermediate value that eventually refers back to the initial intermediate value) or your macro will not work.

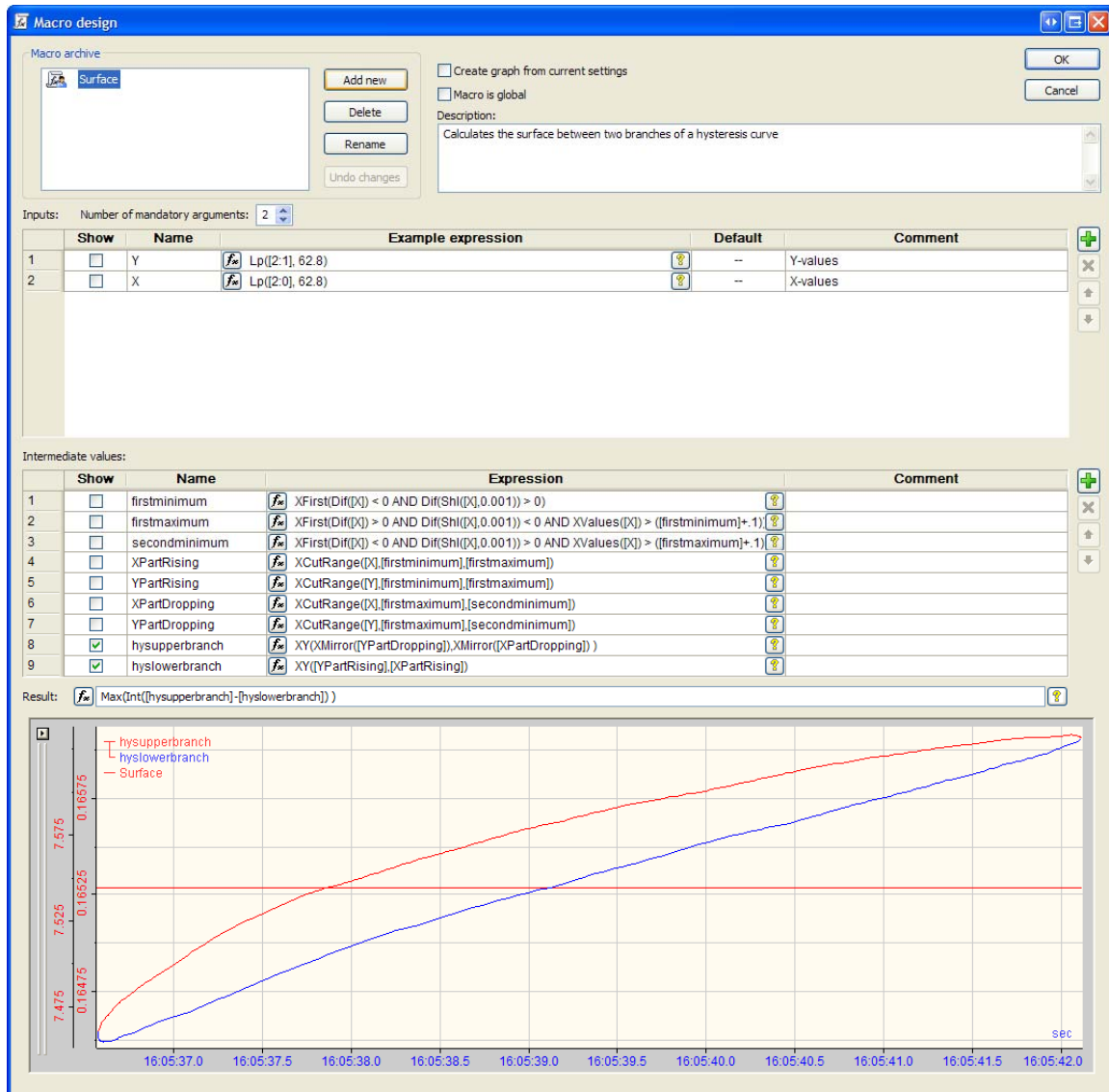
Using macros in expressions

After you have designed a macro in the macro designer, it becomes available for you to use in ibaAnalyzer expressions like a regular ibaAnalyzer function. The macros will be available from the function tree in the *Expression builder* dialog. While typing an expression, the auto completion will suggest the macro if you start typing the first letters of the macro name. If you have provided a macro description and comments for the input arguments in the macro designer, they will be available as help windows while editing an expression that contains the macro; this is similar as for editing expressions containing regular ibaAnalyzer functions.



Example 1: Hysteresis calculation

This macro calculates the area between the upper branch and lower branch of a hysteresis loop.



The macro function takes the following input parameters:

- **Y:** The output signal, which is expected to be a function of X
- **X:** The input signal.

The output signal has the property that it takes on lower values while the input signal is increasing then it does when the input signal is decreasing. Hence the XY plot of the output signal versus the input signal shows a lower branch (blue part of the loop) when the input signal is increasing and an upper branch (red part of the loop) while the input signal is decreasing.

The following intermediate values are used in the macro:

- firstminimum: This is the point where the input signal starts increasing. With the aid of the ibaAnalyzer *XFirst* function, the first sample point of the input signal where the derivative (ibaAnalyzer *Dif* function) changes sign from negative to positive is determined.
- firstmaximum: This is the point where the input signal starts decreasing. Again with the aid of the ibaAnalyzer *XFirst* function the first sample point after *firstminimum* where the derivative changes sign from positive to negative is determined.
- secondmaximum: This is the point where the input signal has finished decreasing. Again with the aid of the ibaAnalyzer *XFirst* function the first sample point after *firstmaximum* where the derivative changes sign from positive to negative again is determined.
- XPartRising: With the aid of the ibaAnalyzer *XCutRange* function the part where the input signal is rising is cut out between *firstminimum* and *firstmaximum*.
- YPartRising: With the aid of the ibaAnalyzer *XCutRange* function the part of the output signal where the input signal is increasing is cut out between *firstminimum* and *firstmaximum*.
- XPartDropping: With the aid of the ibaAnalyzer *XCutRange* function the part where the input signal is decreasing is cut out between *firstmaximum* and *secondminimum*.
- YPartDropping: With the aid of the ibaAnalyzer *XCutRange* function the part of the output signal where the input signal is rising is cut out between *firstmaximum* and *secondminimum*.
- hyslowerbranch: The lower branch of the hysteresis loop is determined by the XY plot of *YPartRising* versus *XPartRising*. This can be done with the aid of the ibaAnalyzer *XY* function.
- hysupperbranch: The upper branch of the hysteresis loop is determined by the XY plot of *YPartDropping* versus *XPartDropping*. This can be done with the aid of the ibaAnalyzer *XY* function. The ibaAnalyzer *XY* function requires its X argument to be strictly increasing, however *XPartDropping* is strictly decreasing. This is amended by applying the ibaAnalyzer *XMirror* function on both *XPartDropping* and *YPartDropping* before supplying them as resp. the X and Y parameter of the *XY* function.

The resulting area is then simply determined by integrating the difference between *hysupperbranch* and *hyslowerbranch*.

Example 2: Head-body-tail calculations

One can consider an aluminum coil to consist out of three sections, namely an initial ‘*head*’ of the coil, followed by a ‘*body*’ consisting out of the largest part of the coil and an ending ‘*tail*’ part of the coil.

Let us define the ‘head’ section of the coil as follows:

- Starting from the beginning of the coil evaluating toward the end of the coil, a given contiguous length of material must be determined for which a quality parameter is within tolerance.
- The head section of the coil is then defined as the section from the beginning of the coil until the length point at which the contiguous section began.

Let us define the ‘tail’ section of the coil as follows:

- Starting from the end of the coil evaluating toward the beginning of the coil, a given contiguous length of material must be determined for which a quality parameter is within tolerance.
- The tail section of the coil is then defined as the section from the length point at which the contiguous section ended (the point of the contiguous section closest to the end of the coil) until the end of the coil.

The body of the coil is then that section of material that lies between the head and tail sections.

We now present two macros that calculate the positions where the body section begins and ends.

Macro to calculate the start of the body section

The macro takes the following input parameters:

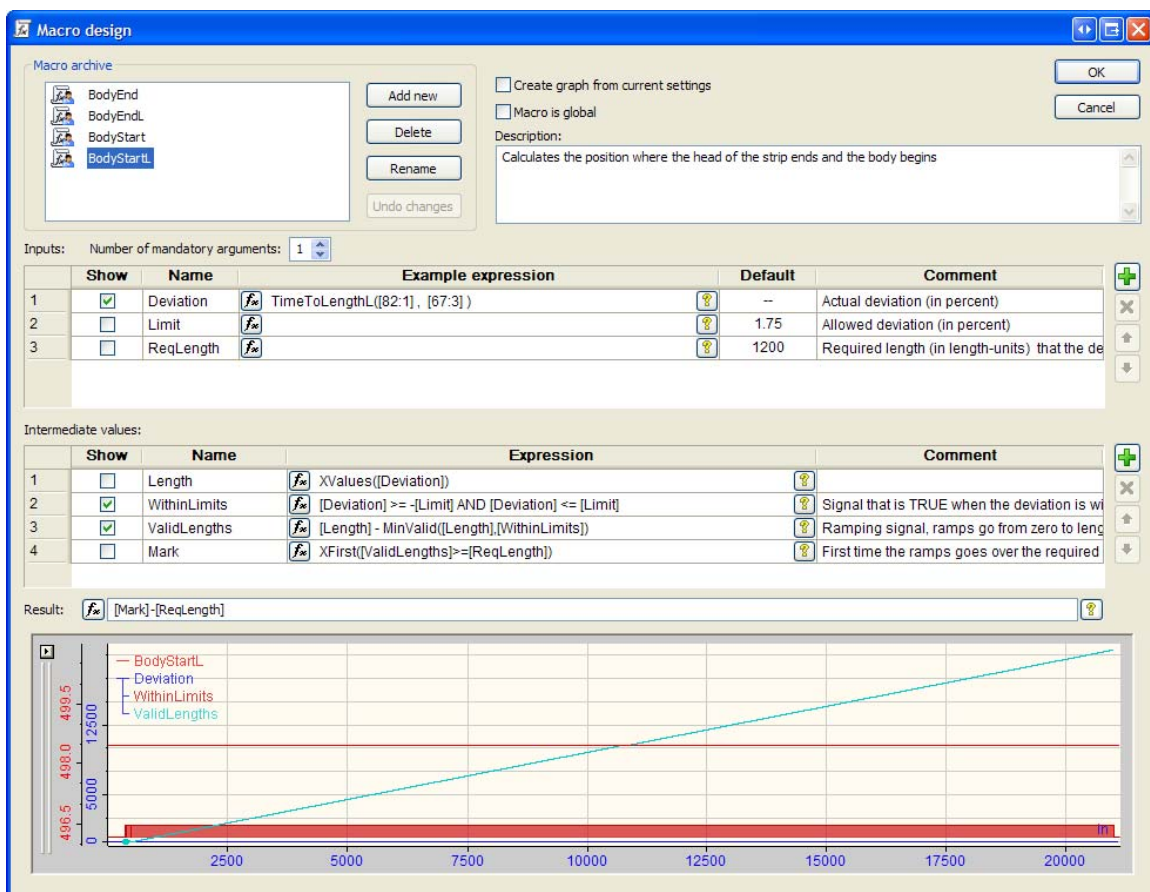
- Deviation: This is the quality parameter that should be within a specified tolerance. This is a length based signal expressed as a percentage.
- Limit: This is the allowed deviation. This is a constant percentage. A default value of 1.75% is specified for this parameter; hence this parameter will be an optional argument of the macro and can be omitted.
- ReqLength: This is the required contiguous length for which the quality parameter must be within tolerance. A default value of 1200 ft is specified for this parameter; hence this parameter will be an optional argument of the macro and can be omitted.

The following intermediate values are used in the macro:

- Length: With the aid of the *ibaAnalyzer XValues* function, this expression gives the length positions of the samples in *Deviation*.
- WithinLimits: This expression determines when the quality parameter is within limits, it is TRUE when $-Limit \leq Deviation \leq Limit$.

- **ValidLengths:** The ibaAnalyzer *MinValid* function in this expression returns for each interval where *WithinLimits* is TRUE the minimum value of *Length* in that interval, and returns no value elsewhere. Hence the entire expression ramps up from zero to the entire length that *WithinLimits* has been TRUE in each interval where *WithinLimits* is TRUE and returns no value elsewhere.
- **Mark:** With the aid of the ibaAnalyzer *XFirst* function the first location where *ValidLengths* equals or exceeds *ReqLength* is determined. It should be clear that this is then the end position of the required length of contiguous material for which the quality parameter is within tolerance.

The resulting body start position is then simply *Mark* minus *ReqLength*.



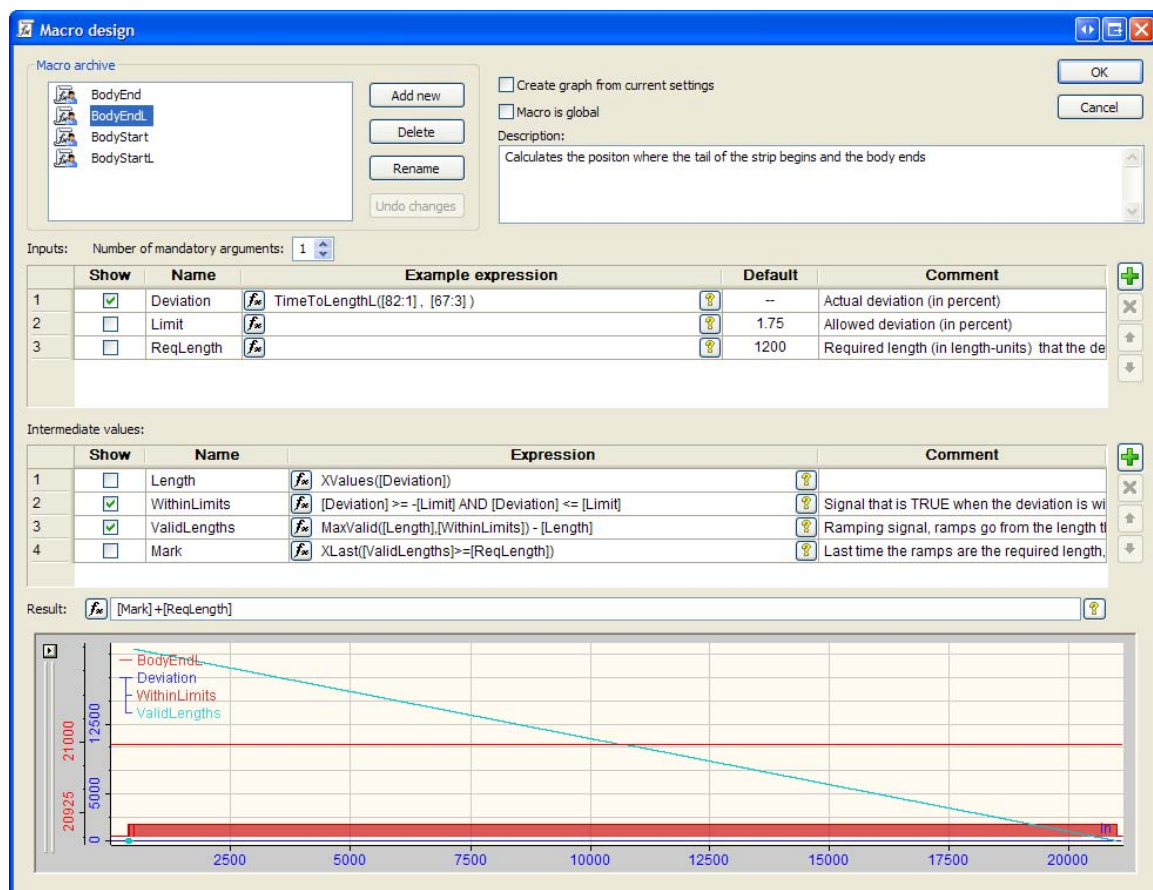
Macro to calculate the start of the body section

This macro has the same input parameters defined as the previous macro.

The following intermediate values are used in the macro:

- **Length:** This expression is the same as in the previous macro.
- **WithinLimits:** This expression is the same as in the previous macro.
- **ValidLengths:** The ibaAnalyzer *MaxValid* function in this expression returns for each interval where *WithinLimits* is TRUE the maximum value of *Length* in that interval, and returns no value elsewhere. Hence the entire expression ramps down from the entire length that *WithinLimits* will be TRUE to zero in each interval where *WithinLimits* is TRUE and returns no value elsewhere.
- **Mark:** With the aid of the ibaAnalyzer *XLast* function the last location where *ValidLengths* equals or exceeds *ReqLength* is determined. It should be clear that this is then the start position of the required length of contiguous material for which the quality parameter is within tolerance.

The resulting body end position is then simply *Mark* plus *ReqLength*.



ibaAnalyzer new and adapted functions

Variable frequency parameter for the electric functions

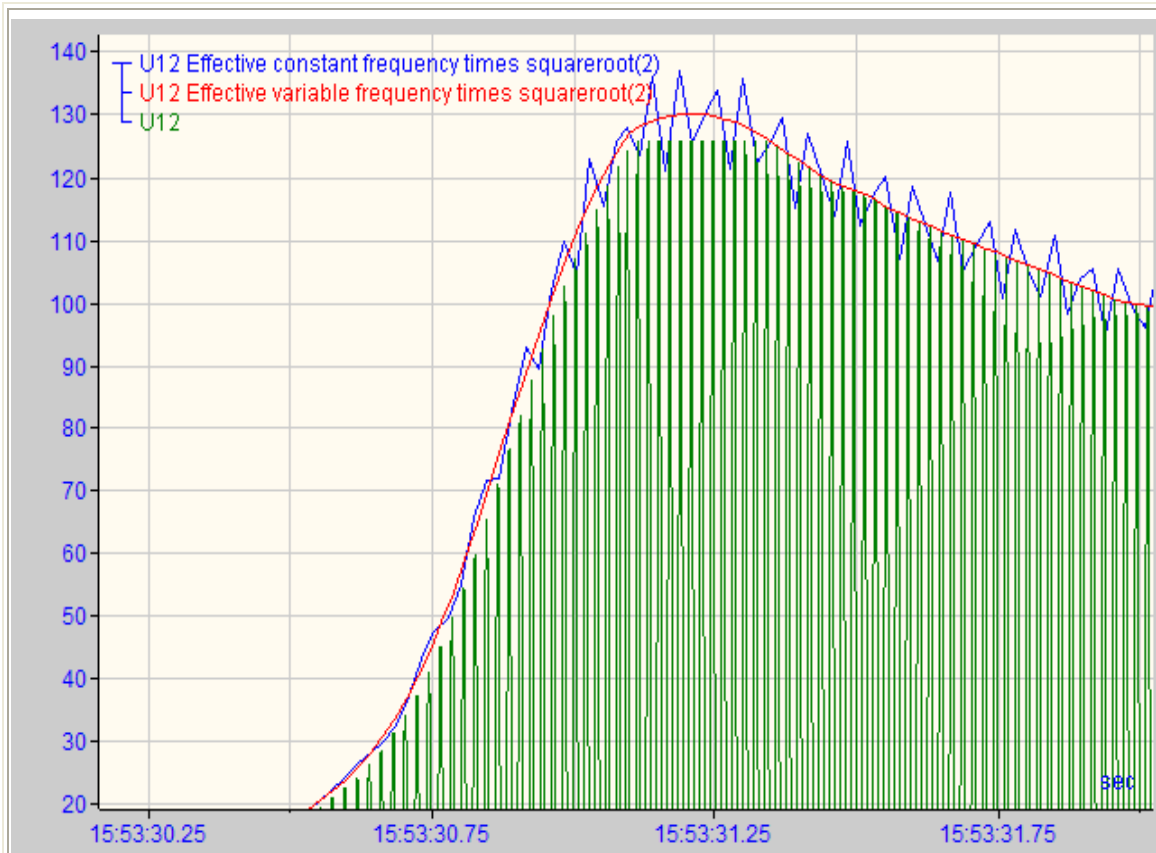
In previous versions of ibaAnalyzer, the '*frequency*' parameter for all electric functions was assumed to be constant. If a non-constant expression was given as '*frequency*' parameter, it was first averaged before being used in the calculations, resulting in inaccurate results if the frequency did indeed vary. In the current version of ibaAnalyzer the electric functions have been adapted to allow for a variable frequency where possible.

The following functions have been adapted:

- *Eff*
- *DeltaCollectiveUeff*
- *DeltaCollectiveIeff*
- *DeltaActiveP*
- *DeltaApparentP*
- *DeltaReactiveP*
- *DeltaActivePFactor*
- *DeltaReactivePFactor*
- *StarCollectiveUeff*
- *StarCollectiveIeff*
- *StarActiveP*
- *StarApparentP*
- *StarReactiveP*
- *StarActivePFactor*
- *StarReactivePFactor*

The following electric functions have not been adapted because their definitions did not allow for implementing a variable frequency meaningfully:

- *DeltaReactivePS*
- *DeltaReactivePFactorS*
- *StarReactivePS*
- *StarReactivePFactorS*
- *TIF*



	Show	SignalName	Expression	Color
1	x	U12 Effective constant frequency times squareroot(2)	Eff([U1_125],Avg([freq_macro])) *Sqrt(2)	Blue
2	x	U12 Effective variable frequency times squareroot(2)	Eff([U1_125],[freq_macro]) *Sqrt(2)	Red
3	x	U12	[17:12]	Green

Adapted GetRows functions

An additional 'Step' parameter is added to the *GetRows* function. It allows you to retrieve every 'Step' row from an array, each time skipping ('Step' - 1) rows. If this parameter is omitted, it has the default value of 1, in which case the *GetRows* function does not skip any rows and behaves like it used to. The first row returned is still the one with the index specified by 'From' (with the first row of the array having index zero).

E.g.: *GetRows*([expr],1,3,2) has parameters 'From' = 1, 'RowCount' = 3 and 'Step' = 2; it will return the 2nd, 4th and 6th row from the array.

Adapted FFT functions

The *'FFTInTime'* and the *'FFTOrderAnalysis'* functions have been extended with an additional optional parameter named *'SuppressDC'*. This parameter has the same meaning as the parameter with the same name in the *'FFTAmpI'* and *'FFTPower'* functions. If it is supplied and set to *TRUE*, the DC component is subtracted from the signal before the FFT calculations are performed.

FFTPeaksInTime

Similar to the *'FFTInTime'* function, *'FFTPeaksInTime'* calculates the FFT of a signal over a moving time interval. However rather than returning an array with the amplitudes of the specified frequency components, it returns an array with the frequency points and amplitudes at those frequencies for which a 'peak' occurs. We say that a 'peak' occurs at a frequency point if the FFT amplitude of the signal at that frequency point is significantly higher than its neighboring frequency points.

You can specify the number of peaks. The frequency values are interwoven with the amplitude values in the returned array; the frequency values are placed in the even numbered rows (starting from 0) while the corresponding values are placed in the odd numbered rows. You can use the *'GetRows'* function to retrieve the desired values from the resulting array.

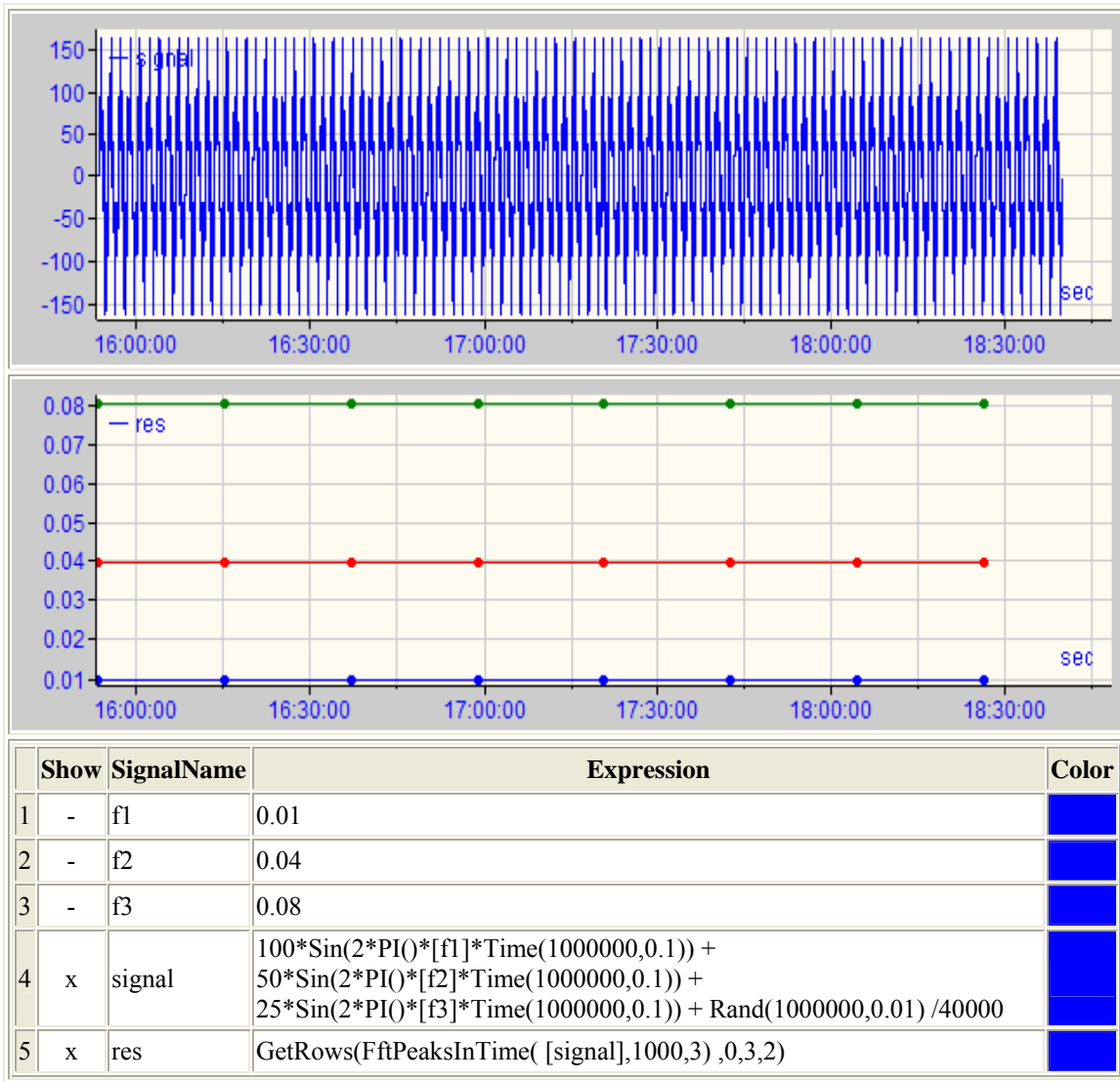
E.g.: suppose you have asked to report 3 peaks and the expression where you stored the *'FFTPeaksInTime'* calculation is named *'res'*:

- *GetRows([res],0)* returns the frequencies where the highest peaks occurred.
- *GetRows([res],1)* returns the amplitudes of the highest peaks.
- *GetRows([res],2)* returns the frequencies where the second highest peaks occurred.
- *GetRows([res],3)* returns the amplitudes of the second highest peaks.
- *GetRows([res],4)* returns the frequencies where the third highest peaks occurred.
- *GetRows([res],5)* returns the amplitudes of the third highest peaks.

The function takes the following parameters:

- **Expression:** The expression to take the FFT of.
- **Time:** This parameter multiplied with the overlap factor determines the time (or length) interval at which a new FFT should be calculated each time. This parameter multiplied with the sample rate of *'Expression'* also determines roughly the amount of samples that are taken into account for generating the FFT function.
- **Peaks:** The number of peaks that need to be reported. The number of rows returned in the resulting array will be twice as large (a row for the frequency and amplitude values each). If there are less peaks in the FFT than this value at a given time point, the last rows will have no value at that time point.
- **Min frequency:** The lowest frequency to be considered, peaks occurring at a lower frequency will not be reported. This parameter is optional, if it is omitted all frequencies calculated by the FFT are considered.

- **Max frequency:** The highest frequency to be considered, peaks occurring at a higher frequency will not be reported. This parameter is optional, if it is omitted all frequencies calculated by the FFT higher than '*Min frequency*' are considered.
- **Window:** As with all FFT functions you can specify a window function that is multiplied with the data before taking the FFT (rectangular or no window = 0, Bartlett = 1, Hamming = 2, Hanning = 3, Kaiser = 4). This parameter can be omitted and has a default value of 0 (no window).
- **Overlap:** Parameter that determines how much the data for calculating each individual FFT overlaps, this is a value between 0 and 1 where 0 = no overlapping and 1 = complete overlapping. This parameter can be omitted and has a default value of 0 (no overlapping).
- **SuppressDC:** If this parameter is supplied and set to *TRUE*, the DC component is subtracted from the signal before the FFT calculations are performed.

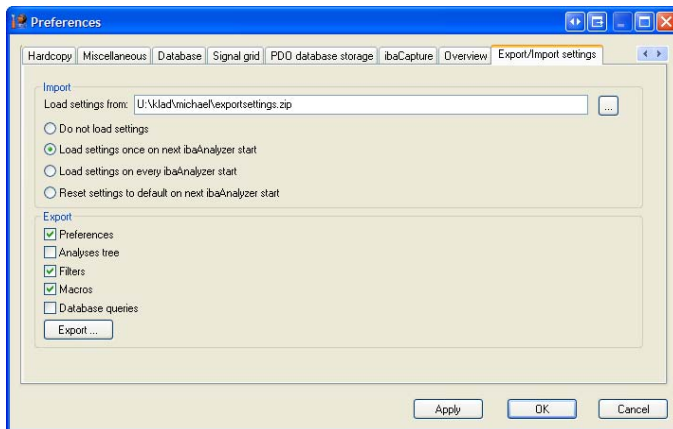


Example: The signal has as main frequency components 0.01, 0.04 and 0.08 Hz. With the *GetRows* function, the frequency rows of the *FFTpeaksInTime* function are shown, reporting these frequencies.

Miscellaneous new ibaAnalyzer features

Exporting and importing ibaAnalyzer settings

In previous versions of ibaAnalyzer it was already possible to export and import the ibaAnalyzer preferences to and from an `.ini` file. Unfortunately other ibaAnalyzer settings, including but not limited to the analyses tree settings, query builder settings, global filters and macro's where not included in this `.ini` file. In the current version of ibaAnalyzer you can export all these settings to files that are placed in a `.zip` file. You can then choose to have ibaAnalyzer extract this `.zip` file and copy all settings from it on the next ibaAnalyzer start. The export and import settings are available through a new tab in the preferences dialog:



In the **import** part you can specify from where the settings should be loaded, alternatively you can browse for the file with the browse button. You can specify which action should be taken on the next ibaAnalyzer start.

- Do not load settings: No specific action is taken regarding the settings.
- Load settings only once on next ibaAnalyzer start: On the next ibaAnalyzer start, ibaAnalyzer will import the settings from the specified `.zip` file. This will not happen again on subsequent ibaAnalyzer starts unless you set this or the next option again in the dialog. If you revisit the dialog after the ibaAnalyzer restart it will have the option 'Do not load settings' selected.
- Load settings on every ibaAnalyzer start: If you have selected this option, ibaAnalyzer will load the settings each time you restart ibaAnalyzer until you chose another option in this dialog.
- Reset settings to default on next ibaAnalyzer start: On the next ibaAnalyzer start, ibaAnalyzer will clear all settings. IbaAnalyzer will behave like it is freshly installed on a PC where never before an ibaAnalyzer has been installed and load its default settings. This will not happen again on subsequent ibaAnalyzer starts unless you set this option again in the dialog. If you revisit the dialog after the

ibaAnalyzer restart it will have the option '*Do not load settings*' selected. This option will not remove any global filter or macros; you can remove them yourself in their respective dialogs.

Important note: Note that no settings will be imported or restored immediately after exiting this dialog. You have to explicitly restart ibaAnalyzer.

In the **export** part you can do the actual export of the settings by clicking the 'Export button'. You will be prompted to specify a location and name to save the .zip file. In the export part you can specify what settings will be exported to the .zip file.

- Preferences: This includes all settings that do not fall under the other categories. Specifically this includes the settings you could already export to an .ini file, however it includes some other settings as well. Excluded from this are dialog positions and toolbar placements.
- Analyses tree: These are the settings regarding the analyses tree, i.e. the tree in the tab 'Analyses files' in the signal tree tabwindow. Note that this is not the only way to export or import analyses trees, the option to import and export the trees manually is still available by going to the analyses tree, right clicking on it and selecting the import or export action from the context menu.
- Filters: This causes the filters to be exported that are marked 'global', i.e. those not embedded in the analysis but those that are always available regardless the loaded analysis. You can create a filter and mark it global through the filter dialog.
- Macros: This causes the macros to be exported that are marked 'global'.
- Database queries: These are the settings regarding the trend and ordinary database queries. This includes the settings in the query builders and file query dialogs. Note that this is not the only way to export the trendquery builder settings, in the current version of ibaAnalyzer, import/export functionality has been added to the trend query dialog (see Pg. 19).

Note: Alternative to using the export/import, global filters and macros can also be copied from and to the ibaAnalyzer application directory, which is typically:

C:\Documents and Settings\user name\Application Data\iba\ibaAnalyzer

Macros will have the file extension .mcr and filters will have the extension .fil .
In previous versions of ibaAnalyzer, the global filters are not placed in the ibaAnalyzer application directory but in the directory where you installed ibaAnalyzer.

Copy-/pasting the ibaAnalyzer graphs between different ibaAnalyzer instances

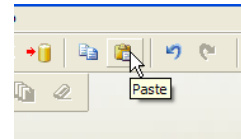
It was already possible to copy the graphs from ibaAnalyzer and paste them in e.g. Ms Word. In the current version of ibaAnalyzer an internal representation of the graphs is copied to the clipboard alongside a HTML export. This internal representation can then be pasted from the clipboard in the same or another instance of ibaAnalyzer. This causes the copied graphs to be placed after the already present graphs in the ibaAnalyzer instance where the paste is performed. This is an easy way to merge several analyses to one analysis.

The copying of the graphs can be done by either

- Clicking the copy button in the toolbar
- Selecting 'Copy' from the 'Edit' submenu of the main menu.
- Right clicking the recorder window and selecting the option 'Copy'
- Pressing Ctrl+C while the recorder window has focus.

The pasting of the graphs can be done by either

- Clicking the new 'Paste' button in the toolbar.
- Selecting 'Paste' from the 'Edit' submenu of the main menu.
- Pressing Ctrl+V while the recorder window has focus.



Starting ibaAnalyzer from a hyperlink in a webpage

When installing the current version of ibaAnalyzer, the necessary registry settings are set so you can start ibaAnalyzer from a hyperlink in a webpage. You can specify parameters in the hyperlink similar to the ibaAnalyzer command line parameters, so this means you can effectively open .dat files and analysis .pdo files from a webpage.

The syntax for the hyperlink URL is "toibaanalyzer:" followed by the parameters separated by semi-colons (;). The parameters may contain spaces; you should not surround parameters containing spaces with quotes (which is illegal in html-syntax anyway) nor should you surround them with escaped quotes.

You can use all parameters that you can normally pass through the command line to ibaAnalyzer; this includes .dat files and an analysis .pdo file and for example a query instruction (/sql, /trendsql or /overviewsql).

E.g.: the following hyperlink in html

```
<a href="toibaanalyzer:\\file server\\user\\edasdata\\1278.dat;c:\\analyses\\x.pdo;/reuse">  
IBA data file on file server </a>
```

when clicked would open ibaAnalyzer in REUSE mode and load the .dat file 1278.dat from the UNC path \\file server\\user\\edasdata with the analysis x.pdo (located locally)

Importing/exporting trend query builder settings

In previous versions of ibaAnalyzer, only the settings for the last selected database table were saved to the registry in the trend query builder dialog. This meant that if you switched tables, you lost your settings.

This has not been modified in the current version of ibaAnalyzer, however as long as you do not leave the trend query dialog, any settings you have set for a table are kept in memory if you switch to another table and are placed back from memory if you eventually switch back to the initial table.

If you require the settings for other tables to be kept longer than the duration of the dialog, you have the option to export your settings for the currently selected table through the new 'export' button in the dialog. When clicking this button, you will get a file dialog prompting you to specify a file name and location to export the settings to an .ini file.

You can import settings saved in such .ini file through the 'import' button in the dialog. If the currently selected table is different than the table that was exported, the dialog will switch to the exported table if it exists, otherwise it will try to apply the settings to the currently selected table.

Trend Query builder

Timestamp field column:

☐ Microsecond field column:

Table:

☐ Filename field column:

Available fields:

- _COMPLETE
- _ERRORONEXTRACT
- _FILEID
- _FILENAME
- _FILETYPE
- _DATAOFFSET
- _LFNR
- _LOS
- _TLOS

Selected fields:

- BNR
- ALPHA
- BETA
- TREFFERCNT

Conditions:

Field	Cond	Value

☐ Merge signals on database sync field:

☐ Add to previous query result

☐ Place result in overview instead of signal tree

New Query Save As SQL ...

Import Export

Query Cancel