



# Wire-Mesh Sensor Data Processing Software

User Manual and Software Description



Institute of Fluid Dynamics  
Helmholtz-Zentrum Dresden-Rossendorf  
01314 Dresden, Germany

M. BEYER, L. SZALINSKI, E. SCHLEICHER, C. SCHUNK

Date: 05/11/2018

Version: 1.3 pre

The present manual contains software and data file descriptions for software modules allowing to process, evaluate and display data of wire-mesh sensor electronics. The described modules run as stand-alone command line programs or can be integrated into the "*WMS Framework*" delivered with the wire-mesh sensor electronics.

---

1	General remarks.....	4
2	FAQ.....	5
3	Theoretical basis.....	6
3.1	Data calibration and calculation of void fractions for conductivity-based measurements (*.mes).....	6
3.2	Data calibration and calculation of volumetric phase fractions for capacitance based measurement data (*.ces).....	9
3.3	Void fraction profiles .....	12
3.4	Bubble identification .....	15
3.5	Bubble property analysis.....	16
3.6	Bubble size distribution.....	18
3.7	Mixtures .....	19
4	Handling of the Wire Mesh Sensor Framework.....	21
4.1	The Wire Mesh Sensor Framework.....	21
4.2	Main Window .....	21
4.3	Welcome Screen.....	23
4.4	Project Tree .....	23
4.5	Project Wizard.....	25
4.6	Project Notes .....	30
4.7	Applying modules.....	31
4.8	Node views.....	31
4.9	Plot2D view .....	33
4.10	Slice view .....	34
4.11	3D Voxel view .....	35
4.12	Table view.....	36
4.13	Batch Processor .....	37
5	Modules for the Wire Mesh Sensor Framework.....	39
5.1	General remarks .....	39
5.2	FileConverter module, converts raw measurement files into single sensor files.....	39
5.3	Geo & GeoAdv module, generation of geometry configuration files.....	41
5.4	Void module, calibration and void fraction distributions.....	42
5.5	CapVoid Module .....	45
5.6	CapVoid2 Module .....	45
5.7	Velocity Module, calculation of the averaged local gas velocities .....	46

---

5.8	BubIdent Module, identification and labelling of single bubbles .....	47
5.9	BubProp Module, bubble property analysis.....	48
5.10	BubSizeDis Module, Calculation of Bubble size distributions.....	50
5.11	MixCalib Module, calibrate mixing measurements .....	51
5.12	MixCond Module, calculation the conductivity of a mixture.....	52
5.13	MixRatio Module, calculating mixing ratios .....	52
5.14	Tansform module, applying various transform operations.....	53
5.15	Trim Module, trimming the size of files.....	54
5.16	FixUp Module, fixing faulty pixels .....	54
5.17	NRemoval Module .....	56
5.18	Reduce Module, reduces multiple frames to a single frame .....	56
5.19	ImageMaker module.....	57
5.20	OptFlow module.....	57
5.21	Histogram Module .....	58
6	Editors of the Wire Mesh Sensor Framework.....	60
6.1	SEI Editor .....	60
6.2	Mask Editor .....	61
7	Internal Data Formats .....	65
7.1	Module Information Files.....	65
7.2	Editor Information files .....	69
7.3	Template information files .....	69
7.4	Template Script Files.....	71
7.5	Command Reference.....	72
7.6	Examples.....	77
7.7	Readable File Formats.....	78
8	Summery .....	81
9	References .....	82

## 1 General remarks

The actual software package described hereby can be used for the data analysis of wire-mesh sensor data files produced by *SGITT-100* and *WMS200* devices. In the last decade, a comprehensive set of software algorithms have been developed at *Helmholtz-Zentrum Dresden-Rossendorf* for wire-mesh sensor data processing. These algorithms comprise tools for reorganization of the data, extraction of single sensors' data from a matrix, calibration routines, calculation of void fractions, void fraction profiles, bubble velocities, bubble size distributions etc. Since the sensor geometries, installation situations, measurement conditions etc. vary widely, not all algorithms can be used for any case. Thus, for instance, bubble velocity calculation is very difficult or even impossible in horizontal stratified flows from wire mesh sensor data. Therefore, the usage of the software algorithms always requires an experienced user with some profound knowledge on multiphase flows. Up to the calculation of so called void files (\*.v) the algorithms are universal. Other tools like velocity calculations, bubble size distributions or interfacial area calculations require special sensor configurations or special flow situation and are not included in the basic package. These tools are available on demand and might need adoption to the special user purpose.

***The use of the software package is at your own risk. HZDR covers no warranty for the calculated results or damage to any system due to the use of the software package.***

## 2 FAQ

- I got an “*OpenGL/GLSL: Unable to compile vertex/fragment shader.*” error message when I try to run the application. What can I do?
  - Please make sure you have installed the latest graphics driver and your graphics card supports at least *OpenGL 2.1*. The implementation conformance for *OpenGL* varies from driver to driver.
- Running the *geometry module* is very slow on my computer. The geometry files generation should take only a few seconds. What can I do to make it faster?
  - Make sure the output directory isn't a network drive. A local hard drive is much faster.
- I don't have an internet connection where I installed the *Wire Mesh Sensor Framework* software. How can I get a registration key?
  - Just copy the content of the organization/name field and the machine id to an USB stick and use an internet-capable computer to send it to the license registration: [license@hzdr-innovation.de](mailto:license@hzdr-innovation.de). It's also possible to take a screenshot or make a photo of the registration dialog. At least three information are needed to finalize the registration process: The used software, a licensee name and the machine identifier.
- The software crashes right after the start-up with a runtime error. What causes this?
  - These errors are hard to reproduce and occur in very rare cases. We discovered that these issues are most likely connected to buggy graphics drivers and how *Qt* handles *OpenGL* contexts. Mostly multi monitor setups with *Intel* graphics cards seem to be affected by this issue. Currently there is no real solution for this kind of problem. Sometimes a driver update seems to work.
- I need a new feature! How to submit a proposal?
  - Please send a message to the software support. E-mail to [c.schunk@hzdr.de](mailto:c.schunk@hzdr.de) (Christoph Schunk) or [license@hzdr-innovation.de](mailto:license@hzdr-innovation.de).
- I've found a bug in the software! I need a fix!
  - Please send a message to the software support. E-mail to [c.schunk@hzdr.de](mailto:c.schunk@hzdr.de) (Christoph Schunk) or [license@hzdr-innovation.de](mailto:license@hzdr-innovation.de).

### 3 Theoretical basis

In this chapter, we will discuss the theoretical background of the measurement system and how the acquired data is saved and processed.

Raw measurement data of the wire-mesh sensor system is saved in a binary format. Assume a sensor of  $64 \times 64$  wires and a measurement of 10 s length with a frame rate of 2500 Hz. For such a sensor, the measurement data is saved as a sequence of 25,000 “frames”, each frame consisting of  $64 \times 64 = 4096$  data values, and each 12-bit deep. The signal is a proportional measure of each crossing-point’s electrical conductivity. The following sections describe how the raw data will be further processed to obtain a set relevant two-phase flow parameters.

**Important Note:** In thermal fluid dynamics the volumetric void fraction (gas, steam) is often designated with the symbol  $\varepsilon$ , this nomenclature is also used in all software modules for temporal or cross sectional averaged void fraction profiles. In electrical engineering, the symbol  $\varepsilon$  stands for the dielectric permittivity of a medium. To avoid misunderstandings, especially in equations related with the capacitance type of WMS, in chapter 3.1 and 3.2 the more general phase fraction designator  $\alpha$  has been used instead. In all further equations  $\varepsilon$  is used again, to match with the module output parameters.

#### 3.1 Data calibration and calculation of void fractions for conductivity-based measurements (\*.mes)

The purpose of this procedure is the conversion of the sensor raw signals from the \*.mes data files obtained from the WMS200 measurement system into volumetric gas fractions. For this, two procedures can be used:

##### Water calibration method

The measured values of the two-phase flow can be weighted with calibration values of a pure water flow, since the electrical conductivity of air is negligibly small, the local instantaneous void (gas) fraction is then given by

$$\alpha_{i,j,k} = \frac{U_{i,j}^W - U_{i,j,k}^{\text{meas}}}{U_{i,j}^W} = 1 - \frac{U_{i,j,k}^{\text{meas}}}{U_{i,j}^W} \quad (3.1)$$

with

$\alpha_{i,j,k}$  the local instantaneous volumetric void fraction,

$U_{i,j}^W$  the time averaged sensor signals of a calibration measurement (water) and

$U_{i,j,k}^{\text{meas}}$  the local instantaneous sensor signal of the measured value.

For the determination of the calibration values for the individual mesh points of the sensor, the data in the calibration file are checked to be free of gas. Afterwards, the signals of the gas free frames are averaged. This method offers the advantage, that it can be applied to all measurement data independent from the superficial gas velocity. The disadvantage consists of the fact that calibration files must be obtained, which is usually done with some temporal shift to the actual measurements. Varying operating conditions (pressure, temperature, conductivity of the water) can thus lead to errors.

### Histogram calibration method

On the other hand, there is the possibility to do a histogram calibration. With this method histograms of the digitized voltage signals of all frames of a measurement file are numerically analysed for each mesh point of the sensor. The histograms usually have two maxima. One maximum lying close to the zero value stands for the gas value, a second maximum for the water value.

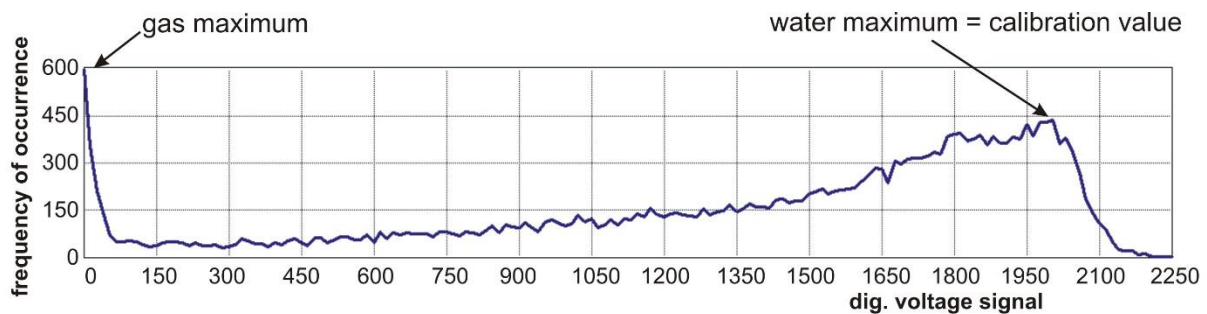


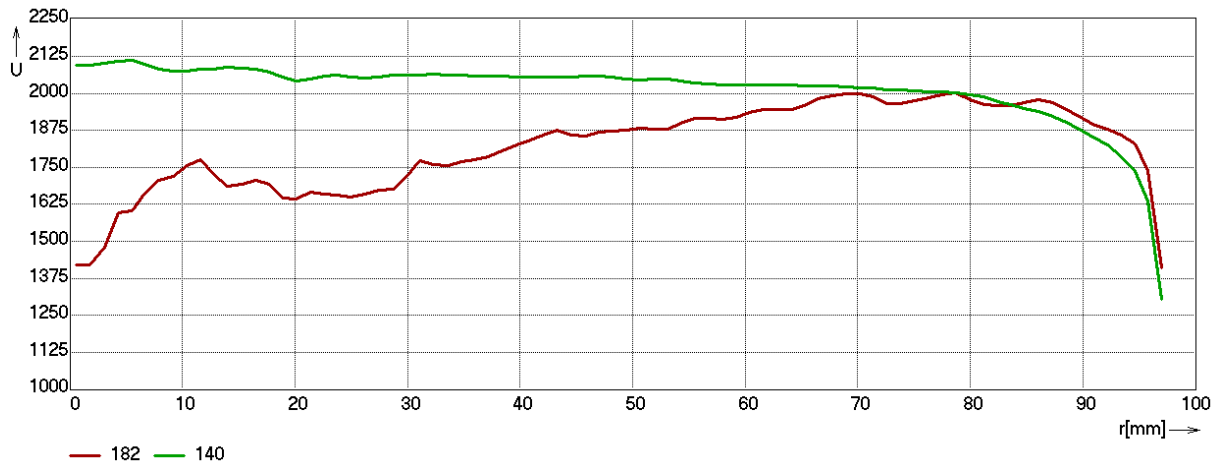
Figure 1: Histogram of the mesh point 43 x 43 for a selected measuring point

This second maximum is used as calibration value for the current mesh point. Figure 1 shows exemplarily a histogram for a test point (e. g.  $j_w = 1.02 \text{ m/s}$ ,  $j_G = 0.53 \text{ m/s}$ ).

There are advantages and disadvantages of this method. The substantial advantage is the determination of the calibration values directly from the measurement data. Errors by changing the boundary conditions are minimised with this method. A disadvantage of the histogram calibration results from the limitation in the volumetric gas content, i. e. at high gas volume fraction only few values for pure liquid are available (in which the measurement volume around the mesh point is totally filled with water). Another example is stratified horizontal flow. In such a case, no clear water maximum can be determined in the histogram for each crossing point and the histogram calibration gives an incorrect calibration value. In order to be able to use the advantages of this method, the calculated matrix composed of calibration values is azimuthally averaged for each test point and the radial profiles are checked.

Figure 2 shows two radial profiles of calibration values for two measurement points in green and red, respectively. The green curve describes an almost ideal calibration profile and confirms the applicability of the method for this measuring point. This curve remains approximately constant for a long distance from the pipe centre ( $r = 0$ )

and only drops steeply directly at the sensor boundary. On the contrary, the red calibration profile goes down in the centre. This curve shows that in the centre of the flow no correct calibration values can be obtained. At the mesh points in the centre of the pipe, there are hardly any frames with pure water in the measurement data according to a high superficial gas velocity. Thus, the histogram calibration method can never be used for stratified or annular flows.



**Figure 2: Comparison of radially averaged calibration profiles for two measuring points ( $j_w = 1.02$  m/s,  $j_G = 0.53$  m/s - green and  $j_w = 0.4$  m/s,  $j_G = 3.2$  m/s - red); histogram calibration**

The decrease in the calibration profiles at the edge of the wire-mesh sensor results from a nonlinearity in the measurements coming from the proximity of the mesh points to the electrically grounded pipe wall. An additional reason for this effect is that many of the measurement volumes are smaller for the mesh points at the edge of the sensor than the standard volume on the inside (cp. Figure 4).

Therefore, it is recommended that the calibration of the measurement data is accomplished in the following steps:

1. Compute radial calibration profiles for all data sets by histogram calibration
2. Check radial calibration profiles for having no radial dependencies except a steep decrease at sensors boundary; beginning at the test points with high superficial gas velocities
3. Depending on the quality of the calibration profiles select the measurements with unsatisfactory results and repeat the calibration using calibration files.

The results of the calibration are stored in the form of an *ASCII* file (\*.*uvw*), which contains the calibration values for each mesh point as a matrix. In addition, the radial calibration profiles (\*.*uvwrad\_80*) are available after successful histogram calibration.

With the help of this calibration matrix, the values for the volumetric void fraction are computed from the measurement data by equation 3.1 with the software *Void* module (user instructions see section 5.4). Due to the presence of signal noise in the



measurement data the calculated volumetric void fraction is filtered before further processing. In order to be able to distinguish small real signals from the signal noise a special filter is used. This filter considers the void fraction values of the surrounding mesh points. That means before a value of the void fraction  $\alpha_{i,j,k}$  ( $i, j$  are the indices of the grid points in the measurement plane and  $k$  is the number of the frame), which is smaller than the threshold of the filter (e. g. 20%), is set to zero, the filter analyses the environment of this void fraction value. Only if all 26 surrounding values are also below the threshold the void fraction  $\varepsilon_{i,j,k}$  is set to zero. If this condition does not apply, then it can be assumed that the measurement signal belongs to the edge region of a bubble. In this case, the measured value remains unchanged.

The threshold of the noise filter (calculated by the software e.g. 10%) theoretically restricts the sensitivity of the wire-mesh sensor used here to a minimum equivalent diameter of 3 mm related to the bubble size. This is the most unfavourable case, if the gas bubble goes through the measurement plane of the sensor in such a way that it begins to cut four measurement volumes equally. In reality the wire-mesh sensor, however, still registers most of these bubbles, since the probability that a bubble penetrates the sensor exactly symmetrically between four wire electrodes and thus producing a void fraction less than the threshold in all of the four measurement volumes is very small. With further decreasing of the bubble size also the probability, that the bubble is still recorded by the sensor, decreases.

After filtering, the void fraction is limited to 0% and 100%, respectively. The data are saved in byte-format in a binary file with the extension \*.v successively for each frame of the measurement. For numerical reasons, these files also contain values for points, which are outside of the circular measurement cross-section. These points are marked with the number 255.

### 3.2 Data calibration and calculation of volumetric phase fractions for capacitance based measurement data (\*.ces)

Since the raw data of the capacitance based measurement system “CapWMS200” are logarithmic scaled and the measured values for gas only measurements are not zero as for the conductivity based system the calibration and phase fraction calculation is a little more complicated. The different steps are described in the following sections.

#### Calibration and calculation of local instantaneous relative permittivity values

It can be shown [1] that the measured logarithmic values can be noted in form of a logarithmic linear function of the capacitance and thus the local permittivity in the crossing points of the wire-mesh sensor.

$$V^{\text{LOG}} = a \cdot \ln(\varepsilon) - b \quad (3.2)$$

with the parameters  $a$  and  $b$  containing all variables in local geometry, variations in electronic components as well as gain and offset settings of the data acquisition system. Hence, the local instantaneous permittivities  $\varepsilon_m$  of each sampling point  $i, j, k$  is calculated from the measured local instantaneous voltages  $V_{i,j,k}^{\text{LOG}}$  as

$$\varepsilon_{i,j,k} = \exp\left(\frac{V_{i,j,k}^{\text{LOG}} - b_{i,j}}{a_{i,j}}\right). \quad (3.3)$$

In order to obtain the two parameters  $a_{i,j}$  and  $b_{i,j}$  these values can be pre-calculated from two calibration data matrices  $V_{i,j}^{\text{H}}$  and  $V_{i,j}^{\text{L}}$  with known permittivity values  $\varepsilon_{\text{L}}$  and  $\varepsilon_{\text{H}}$ .  $V_{i,j}^{\text{H}}$  and  $V_{i,j}^{\text{L}}$  are the temporal averaged cross sections of the calibration input data:

$$V_{i,j}^{\text{L}} = \frac{1}{N} \sum_{k=0}^{N-1} V_{i,j,k}^{\text{LOG,L}} \quad (3.4)$$

and

$$V_{i,j}^{\text{H}} = \frac{1}{N} \sum_{k=0}^{N-1} V_{i,j,k}^{\text{LOG,H}} \quad (3.5)$$

The input data ( $V_{i,j,k}^{\text{LOG,L}}$ ) can be obtained by covering the whole sensor with the low permittivity substance. For the high calibration input data ( $V_{i,j,k}^{\text{LOG,H}}$ ), the substance with high permittivity is used to cover the sensor. With these calibration matrices, the two parameters  $a_{i,j}$  and  $b_{i,j}$  can now be calculated by

$$a_{i,j} = \frac{V_{i,j}^{\text{H}} - V_{i,j}^{\text{L}}}{\ln(\varepsilon_{\text{H}}) - \ln(\varepsilon_{\text{L}})} \quad (3.6)$$

and

$$b_{i,j} = \frac{V_{i,j}^{\text{L}} \ln(\varepsilon_{\text{H}}) - V_{i,j}^{\text{H}} \ln(\varepsilon_{\text{L}})}{\ln(\varepsilon_{\text{H}}) - \ln(\varepsilon_{\text{L}})} \quad (3.7)$$

For further details, please cp. [1]

### Phase fraction calculation

In order to calculate the volumetric phase fractions of a two-phase mixture, several dielectric mixing models have been developed [1]. These models are based on different assumptions onto how the two phases are geometrically distributed. The most widely used model is the *parallel* model, which is the only model implemented in the classical *CapVoid* module. Furthermore, the *CapVoid2* module supports several additional models.

For better understanding of the used models, we will describe the used parameters a little more in detail, where

$\alpha$  is phase fraction of the mixture ( $0 \leq \alpha \leq 1$ ),

$\varepsilon_1$  is the permittivity of the first phase (usually the lower value),

$\varepsilon_2$  is permittivity the second phase (usually the higher value) and

$\varepsilon_m$  the measured permittivity of the mixture.

Using these parameters, the *parallel model* is defined as:

$$\alpha_P = \frac{\varepsilon_2 - \varepsilon_m}{\varepsilon_2 - \varepsilon_1} \quad (3.8)$$

The *series model* is defined as:

$$\alpha_S = \frac{\varepsilon_1 \varepsilon_2 - \varepsilon_m \varepsilon_1}{\varepsilon_m \varepsilon_2 - \varepsilon_m \varepsilon_1} \quad (3.9)$$

And the *logarithmic model* is defined as:

$$\alpha_L = \frac{\log(\varepsilon_2) - \log(\varepsilon_m)}{\log(\varepsilon_2) - \log(\varepsilon_1)} \quad (3.10)$$

The *Maxwell-Garnett model* is based on spherical inclusions dispersed randomly in a homogeneous medium. We have two equations here, one for the low permittivity phase dispersed in the high permittivity phase:

$$\alpha_{M_1} = \frac{\left(2 + \frac{\varepsilon_1}{\varepsilon_2}\right) \left(1 - \frac{\varepsilon_m}{\varepsilon_2}\right)}{\left(1 - \frac{\varepsilon_1}{\varepsilon_2}\right) \left(2 + \frac{\varepsilon_m}{\varepsilon_2}\right)} \quad (3.11)$$

And one for the high permittivity phase dispersed in the low permittivity phase:

$$\alpha_{M_2} = 1 - \frac{\left(2 + \frac{\varepsilon_2}{\varepsilon_1}\right) \left(1 - \frac{\varepsilon_m}{\varepsilon_1}\right)}{\left(1 - \frac{\varepsilon_2}{\varepsilon_1}\right) \left(2 + \frac{\varepsilon_m}{\varepsilon_1}\right)} \quad (3.12)$$

Figure 3 shows the permittivity plot for the models described above. The two-phase mixture consists of oil for the low permittivity value ( $\varepsilon_L = 2$ ) and water for the high permittivity value ( $\varepsilon_H = 80$ ). In Figure 3: Permittivity of a 2-phase mixture of oil and water for different dielectric mixture models. Figure 3 you can clearly see the different curves for the permittivity models described above.

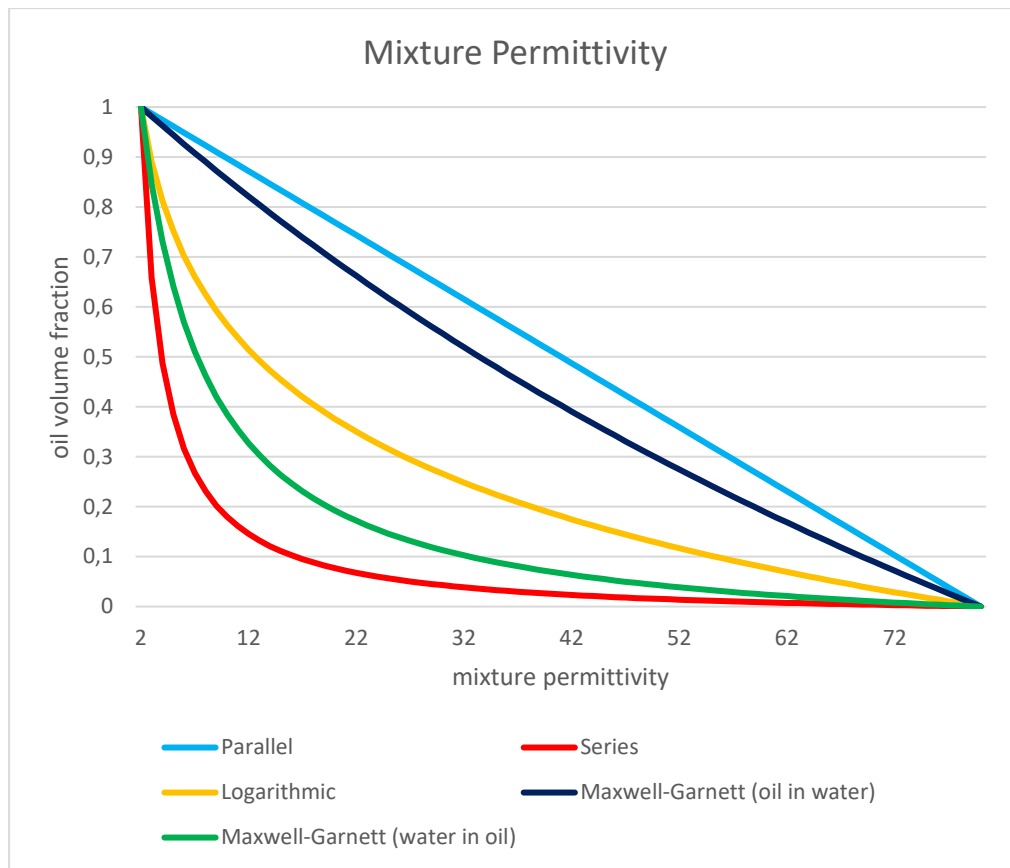


Figure 3: Permittivity of a 2-phase mixture of oil and water for different dielectric mixture models.

Please have a look at chapter 5.5 to see how the different models are selected in the *CapVoid2* module.

### 3.3 Void fraction profiles

In order to obtain quantitative information on the flow, a time and spatial averaging of the void fraction data can be done. These procedures are also contained in the *Void* module. Contrary to the void fraction values stored in the v-files, the data used for averaging are not limited at 0% and 100%.

The averaging is based on weight coefficients that define the contribution of each crossing point of wires ( $i, j$ ) in the sensor matrix to the size of the domain, over which the averaging has to be performed. The definition of the weight coefficients ( $a_{i,j}$ ) necessary to obtain a cross-section averaged void fraction is shown in Figure 4. The average can be calculated for each sampling period individually with

$$\bar{\varepsilon}_k = \varepsilon(t) = \sum_i \sum_j a_{i,j} \cdot \varepsilon_{i,j,k} \quad (3.13)$$

The result of the data evaluation with equation 3.13 is a sequence of instantaneous average volumetric gas fractions, which is available with the full measurement frequency. These values are stored in a separate file (\*.epst).

Another option is the averaging in time. Two-dimensional averaged void fraction distributions are computed from

$$\bar{\varepsilon}_{i,j} = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \varepsilon_{i,j,k} \quad (3.14)$$

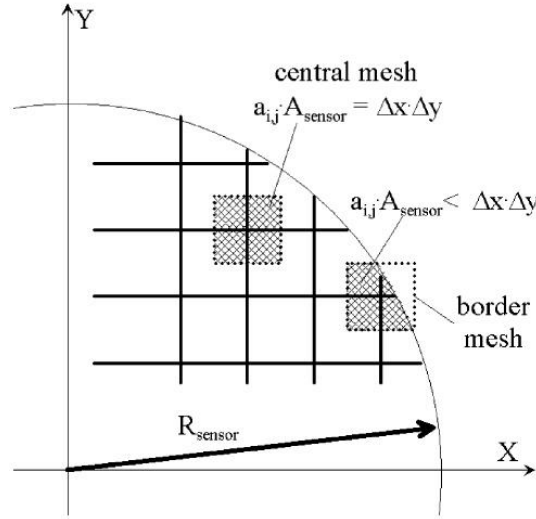


Figure 4: Weight coefficients for the cross-section averaging of local gas fractions measured by the wire-mesh sensor

With equations (3.13) and (3.14) an average void fraction for the total measurement cross-section can be obtained as

$$\bar{\varepsilon} = \sum_i \sum_j a_{i,j} \cdot \bar{\varepsilon}_{i,j} = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \bar{\varepsilon}_k \quad (3.15)$$

Moreover, a radial gas fraction profile can be calculated by averaging the local instantaneous gas fractions over the measurement period and over a number of ring-shaped domains ( $m$ ). The latter is done by

$$\bar{\varepsilon}_m = \frac{1}{k_{\max}} \sum_k \sum_i \sum_j a_{i,j,m} \cdot \varepsilon_{i,j,k} \quad (3.16)$$

where  $a_{i,j,m}$  are weight coefficients denoting the contribution of each measurement point with the indexes  $i, j$  to a ring with the number  $m$  (cp. Figure 5). This ring-shaped averaging domain covers a given radial distance  $r$  from the centre of the sensor with

$$(m-1) \cdot \frac{R_{\text{sensor}}}{m_{\max}} \leq r \leq m \cdot \frac{R_{\text{sensor}}}{m_{\max}}, \quad (3.17)$$

where  $m_{\max}$  is the total number of radial steps (e. g. for DN200 wire-mesh sensors a value of  $m_{\max} = 80$  is recommended) and  $R_{\text{sensor}}$  is the radius of the sensor grid.

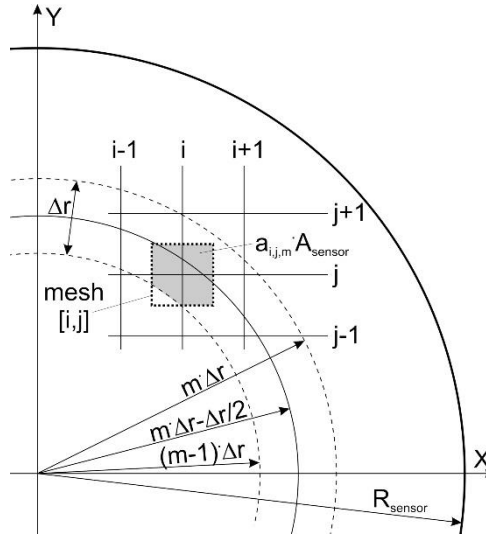


Figure 5: Weight coefficients for the cross-section averaging of local gas fractions over a number of ring-shape domains

Thus, azimuthally and time averaged gas volume fraction profiles (*\*.epsrad\_80*), as well as the time averaged local gas fractions for each mesh point of the pipe cross-section (*\*.epsxy*) are available. In the file (*\*.epst*) the cross sectional averaged void fractions for each time step are stored in two columns. Furthermore, the time and space averaged gas volume fractions according to equation (3.15) (*eps\_all.asc*) can be used for the comparison of the individual experiments among themselves.

### Gas velocities

The use of a sensor with two measurement planes or two sensors one after another allows the determination of time and azimuthally averaged gas velocities. To do this, the signals from both measurement planes are *cross-correlated* separately for each pair of mesh points, which are located above each other. For time-discrete series of this fluctuation components of the local instantaneous void fraction from the first measurement plane ( $\varepsilon'_{1,i,j,k}$ ) and the second plane ( $\varepsilon'_{2,i,j,k}$ ) of the sensor, the cross-correlation can be defined as

$$F_{i,j,\Delta k} = \frac{\sum_k \varepsilon'_{1,i,j,k} \cdot \varepsilon'_{2,i,j,k+\Delta k}}{\sqrt{\sum_k \varepsilon_{1,i,j,k}^2} \cdot \sqrt{\sum_k \varepsilon_{2,i,j,k}^2}}. \quad (3.18)$$

The index  $\Delta k$  corresponds to the time-shift of  $\Delta t = \Delta k / f_{\text{meas}}$ . Fluctuation components are calculated by subtracting the time-average from the instantaneous value  $\varepsilon'_{i,j,k} = \varepsilon_{i,j,k} - \bar{\varepsilon}_{i,j}$ . The cross-correlation is carried out by means of *Fast Fourier Transformation (FFT)*. The obtained cross-correlation functions are averaged in circumferential direction for different radii ( $m$ ) using the same weight coefficients as for the calculation of radial gas fraction profiles

$$F_{m,\Delta k} = \frac{1}{k_{\max}} \sum_i \sum_j a_{i,j,m} \cdot F_{i,j,\Delta k} \quad (3.19)$$

In the next step, the location of the maximum in the cross-correlation functions averaged by equation (3.19) is determined. The average gas-phase velocity for the given radius is calculated from the corresponding time-shift

$$w_G(r) = w_G(m) = \frac{\Delta L}{\Delta k_{\max}} \cdot f_{\text{meas}} \quad (3.20)$$

With  $\Delta k_{\max}$  corresponding to  $F_{m,\Delta k_{\max}} = \max(F_{m,\Delta k})$ .

In equation (3.20)  $\Delta L$  is the axial distance between the two measurement planes. The technique of averaging the cross-correlation functions before searching for the maximum has proven to supply more stable velocity values than if the velocities were directly deduced from the result of a point-to-point cross-correlation according to equation (3.18).

For the documentation of the evaluation process, the results of the point-to-point cross-correlation are stored in an *ASCII* file containing the time averaged velocity for each crossing point (*\*.velxy*) in a two dimensional matrix. The azimuthally averaged gas velocities and the middle radii of the  $m$  rings are stored in the *ASCII* files *\*.vel*.

### 3.4 Bubble identification

The bubble identification and the determination of important characteristics for the bubbles can be done using special evaluation algorithms. Thereby, a bubble is defined as a region of connected gas-containing elements in void fraction data  $\varepsilon_{i,j,k}$  which is completely surrounded by elements containing the liquid phase. To each element which belongs to one bubble, the same identification number is assigned. Different bubbles receive different identification numbers. These numbers are stored in the elements  $b_{i,j,k}$  of a second array that is saved in a binary file (32-bit signed integer) of the type *\*.b*. This array has the same dimension as the void fraction array. After the bubble recognition algorithm is completed, each element  $b_{i,j,k}$  carries the number of the bubble to which the given element with the indices  $i, j, k$  belongs.

Local instantaneous gas fractions can have values between 100% (gas) and 0% (liquid), if the corresponding measurement volume formed by two crossing wires contains both gas and liquid at the same time. Furthermore, signal noise may also lead to such intermediate values. Consequently, a sharp distinction between elements filled with gas and elements filled with water is not possible. To recognize the unique bubbles even under these difficult conditions *HZDR* implemented a so called recursive extended decremental fill algorithm. It is based on the idea that the local gas fraction can only decrease or remain equal if the identification is started at the point of the

highest gas fraction found inside a bubble. After successful completion of the bubble detection the results (bubble identification matrix) are saved in the binary file \*.b .

### 3.5 Bubble property analysis

Using the bubble identification matrix together with the information about the void fraction, now important parameters can be determined for each bubble. They are stored in text files of the type \*.a . It has to be noted that in the code the index  $i$  refers to the serial number of the frames, while  $j$  and  $k$  , in this case, serve as indices in the measurement plane. This assignment of the indices is also used in the equations given in this section.

The *volume of a bubble* with the number  $n$  is obtained by integrating the local void fraction of all elements owning the given bubble number

$$V_{b,n} = \Delta x \Delta y \Delta t \cdot w_b \sum_{i,j,k} \varepsilon_{i,j,k} \quad \forall [i,j,k]: b_{i,j,k} = n. \quad (3.21)$$

The sum of void fractions is multiplied by the measurement volume, which is the product of the distance of the electrodes in  $x$  and  $y$  directions and the sampling period, as well as the bubble velocity

$$\Delta t = \frac{1}{f_{\text{sample}}}. \quad (3.22)$$

Due to the fact that the individual velocity of bubbles is unknown, the gas phase velocity obtained by *cross-correlation* is taken as an approximation at the location of the centre of mass of the given bubble

$$w_b = w_G(r_n) \quad \text{with} \quad r_n = \sqrt{(x_{\text{CM},n} - x_0)^2 + (y_{\text{CM},n} - y_0)^2}. \quad (3.23)$$

Here  $x_0$  and  $y_0$  are the centre coordinates of the wire mesh sensor.

The *coordinates of the centre of mass* can be obtained by averaging the measurement coordinates of all elements belonging to the selected bubble using the local void fraction values as a weight function

$$x_{\text{CM},n} = \frac{\sum_{i,j,k} j \cdot \Delta x \cdot \varepsilon_{i,j,k}}{\sum_{i,j,k} \varepsilon_{i,j,k}}; \quad y_{\text{CM},n} = \frac{\sum_{i,j,k} k \cdot \Delta y \cdot \varepsilon_{i,j,k}}{\sum_{i,j,k} \varepsilon_{i,j,k}} \quad (3.24)$$

$$z_{\text{CM},n} = \frac{\sum_{i,j,k} i \cdot \Delta z \cdot \varepsilon_{i,j,k}}{\sum_{i,j,k} \varepsilon_{i,j,k}}; \quad \Delta z = w_b \cdot \Delta t \quad \forall [i,j,k]: b_{i,j,k} = n$$



After that, the equivalent diameter of the bubble can be determined, which is defined as the diameter of a sphere that has the volume according to equation (3.21)

$$D_{b,n} = \sqrt[3]{\frac{6V_{b,n}}{\pi}}. \quad (3.25)$$

For the evaluation of asymmetries of the bubble, moments for each bubble are calculated. Likewise, the void fraction served as weight function

$$rm_{x,n} = \sqrt{\frac{5 \cdot \sum_{i,j,k} \varepsilon_{i,j,k} \cdot (j \cdot \Delta x - x_{CM,n})^2}{\sum_{i,j,k} \varepsilon_{i,j,k}}}; \quad rm_{y,n} = \sqrt{\frac{5 \cdot \sum_{i,j,k} \varepsilon_{i,j,k} \cdot (k \cdot \Delta y - y_{CM,n})^2}{\sum_{i,j,k} \varepsilon_{i,j,k}}} \quad (3.26)$$

$$rm_{z,n} = \sqrt{\frac{5 \cdot \sum_{i,j,k} \varepsilon_{i,j,k} \cdot (i \cdot \Delta z - z_{CM,n})^2}{\sum_{i,j,k} \varepsilon_{i,j,k}}}; \quad \Delta z = w_b \cdot \Delta t; \quad \forall [i, j, k]: b_{i,j,k} = n$$

From the moments for the coordinates  $x$  and  $y$  in the measurement plane of the wire-mesh sensor, the radial moment is

$$rm_{r,n} = \sqrt{rm_{x,n}^2 + rm_{y,n}^2}. \quad (3.27)$$

Further information on the distortion of the bubble can be obtained by calculating the maximum equivalent diameter in the  $x$ - $y$  plane. For this matter, the area being occupied by the bubble in the  $x$ - $y$  plane is added. Similar to equation (3.21) the sum of the local instantaneous void fractions of the measurement volumes belonging to the bubble is multiplied by the area of the measurement volume in the  $x$ - $y$  plane. This procedure is done for each single sampling time characterised by index  $i$

$$A_{xy,n,i} = \Delta x \Delta y \sum_{j,k} \varepsilon_{i,j,k} \quad \forall [i, j, k]: b_{i,j,k} = n. \quad (3.28)$$

Afterwards, the maximum area is found and converted into the *diameter of an area equivalent circle*

$$D_{xy,n} = \sqrt{\frac{4A_{xy,n,\max}}{\pi}} \quad \text{with} \quad A_{xy,n,\max} = \max(A_{xy,n,i}). \quad (3.29)$$

In addition to these bubble characteristics the *minimum and maximum coordinates* of the bubbles are determined. For the calculation of these values, it is necessary to define a threshold value of the gas fraction which represents the bubble interface. Experiences at HZDR show that for bubble sizes  $> 20$  mm a threshold 50% is a good approximation. If bubble diameters are smaller this value is reduced to approx. 20%. The maximum of gas fraction in the bubble (starting at 100% for large bubbles) also reduces with decreasing bubble diameter. This reduction is observed for bubbles with a diameter

less than approx. 20 mm. This effect results from the limited spatial resolution of the wire-mesh sensor which is e. g. 3 x 3 mm. Small bubbles cannot completely fill the associated measurement volumes. For this reason, maximum gas fractions lower than 100% are observed. Taking these boundary conditions into consideration, as a compromise, the gas fraction threshold representing the bubble interface is taken as half of the maximum gas content of the bubble.

Another important parameter for the characterisation of gas bubbles is the *volume fraction of the bubble related to the total volume of the flow*

$$\varepsilon_{b,n} = \frac{V_{b,n}}{V_{\text{all}}}; \quad V_{\text{all}} = t_{\text{meas}} \cdot f_{\text{meas}} \cdot \Delta t \cdot \bar{w}_G \cdot A_{\text{sensor}}; \quad \bar{w}_G = \frac{J_G}{\varepsilon}. \quad (3.30)$$

Apart from the already mentioned parameters, the maximum gas fraction and the number of measurement volumes per bubble are determined. All values are stored in an *ASCII* file (\*.a) as table for each identified bubble.

### 3.6 Bubble size distribution

After evaluation of the data described in the previous chapters, three-dimensional distribution of gas fraction as well as characteristic parameters for each single bubble are available. Additionally, a list with characteristics of each bubble is generated for each measurement.

The combination of these data makes it possible to obtain bubble size distributions. To do this, histograms are calculated in which the void fraction per bubble class is summed. This is done related to the volume equivalent diameter according to equation as well as related to the area equivalent diameter of the gas bubbles according to equation (3.25). This information is available in a representation with a linear bubble class width of 0.25 mm and also for a logarithmically increasing width of the bubble classes. The smallest bubble size class for the logarithmic representation has a lower boundary of 0.1 mm. The bubble size distributions are stored in *ASCII* files with the extensions \*.his\_lin and \*.his\_log, respectively. The linear distributions are preferably used for the numerical investigations and the logarithmic information for visualisation.

In both types of bubble size distributions, the void fraction that is related to the bubble class width is represented by  $(\Delta\varepsilon / \Delta D_b)$ , which gives

$$\varepsilon_{\text{all}} = \sum_0^{D_{b,\text{max}}} \frac{\Delta\varepsilon}{\Delta D_b} \cdot \Delta D_b. \quad (3.31)$$

In addition, these distributions related to the total gas content  $(\Delta\varepsilon / \Delta D_b / \varepsilon_{\text{all}})$  are listed in both files. Furthermore, the files contain the bubble number distributions with

which the absolute number of bubbles per bubble class is referred to the bubble class width and the total measurement time.

### 3.7 Mixtures

Calculating mixtures is done in three steps. A calibration step where a set of calibration references is used to determine a linear relationship between the measured ADC values and their corresponding conductivity values. You can use the *mixcalib* module to do a calibration. The second step uses the calibration matrices  $\hat{m}$  and  $\hat{n}$  to calculate a conductivity. The *mixcond* module is used to calculate the conductivity. The last step determines the mixing ratio for a given conductivity from a base conductivity and a tracer fluid. Mixing ratios are determined by the *mixratio* module.

#### Calibration

Mixtures are modelled using a linear relationship between the conductivity  $C_{r,j,k}$  and the measured ADC value  $V_{r,j,k}$  of the sensor by

$$V_{r,j,k} = m_{j,k} C_{r,j,k} + n_{j,k}, \quad (3.32)$$

where  $r$  is the calibration reference and  $j, k$  are the measured crossing points. The parameters  $m_{j,k}$  and  $n_{j,k}$  are calculated by using a simple linear regression for each measurement point.

The regression line is defined as

$$\hat{V}_{j,k} = \hat{m}_{j,k} \hat{C}_{j,k} + \hat{n}_{j,k}. \quad (3.33)$$

with

$$\hat{m}_{j,k} = \frac{\sum_{r=1}^N (C_{r,j,k} - \bar{C}_{j,k})(V_{r,j,k} - \bar{V}_{j,k})}{\sum_{r=1}^N (C_{r,j,k} - \bar{C}_{j,k})^2}. \quad (3.34)$$

and

$$\hat{n}_{j,k} = \bar{V}_{j,k} - \hat{m}_{j,k} \bar{C}_{j,k}. \quad (3.35)$$

The variables  $\bar{V}_{j,k}$  and  $\bar{C}_{j,k}$  are the average of  $V_{r,j,k}$  and  $C_{r,j,k}$ .

To have an estimate on how good the linear fitting was, we calculate the coefficient of determination  $R_{j,k}^2$  for each measure point by

$$R^2_{j,k} = \frac{\sum_{r=1}^N (\hat{V}_{r,j,k} - \bar{V}_{j,k})^2}{\sum_{r=1}^N (V_{r,j,k} - \bar{V}_{j,k})^2}. \quad (3.36)$$

Values around 0 indicate a low determination and values near 1 indicate a high determination.

### Conductivity Calculation

If the used calibration lead to a satisfying result, the parameters  $\hat{m}$  and  $\hat{n}$  can now be used to calculate the conductivity for each crossing point by using

$$C_{j,k} = \frac{V_{j,k} - \hat{n}_{j,k}}{\hat{m}_{j,k}}. \quad (3.37)$$

### Mixing Ratio Calculation

The mixing ratio  $\theta_{i,j,k}$  is calculated by using a base reference fluid  $\bar{C}_{j,k}^M$  and a tracer fluid  $\bar{C}_{j,k}^T$ .

$$\theta_{i,j,k} = \frac{C_{i,j,k} - \bar{C}_{j,k}^M}{\bar{C}_{j,k}^T - \bar{C}_{j,k}^M}. \quad (3.38)$$

The resulting values of  $\theta_{i,j,k}$  range from 0 (no tracer present) and 1 (fluid is completely filled with tracer).

## 4 Handling of the Wire Mesh Sensor Framework

### 4.1 The Wire Mesh Sensor Framework

The *Wire Mesh Sensor Framework* has been designed and developed to use the stand-alone command line-based data processing modules, which have been developed for batch processing, in a user-friendly GUI environment. The framework allows the user to use standard file open and file save dialogs and generates an easy assessable interface to the command line parameters of the single modules. Therefore, for each module a "module information file" (\*.minf) has to be provided. The MINF-file describes the input and output specifications and the required parameters of corresponding module. The modules are automatically loaded and registered after the program start-up. MINF -files can be opened and edited with any text editor. The easiest way to learn how to write your own modules and MINF -files is to open an existing MINF -file and study the parameters and its structure. An extensive documentation of the MINF -file format and other file formats related to the *Wire Mesh Sensor Framework* can be found in the Internal Data Formats section (cp. 7.1). In the following, the different sections of the main window, views and dialog boxes of the framework are described. Terms from the GUI are printed *italic* for easy location.

### 4.2 Main Window

The main window is divided into three major parts. The left side is the project tree which contains all relevant files like measurement and geometry files. It also contains all generated intermediate files in a hierarchical structure.

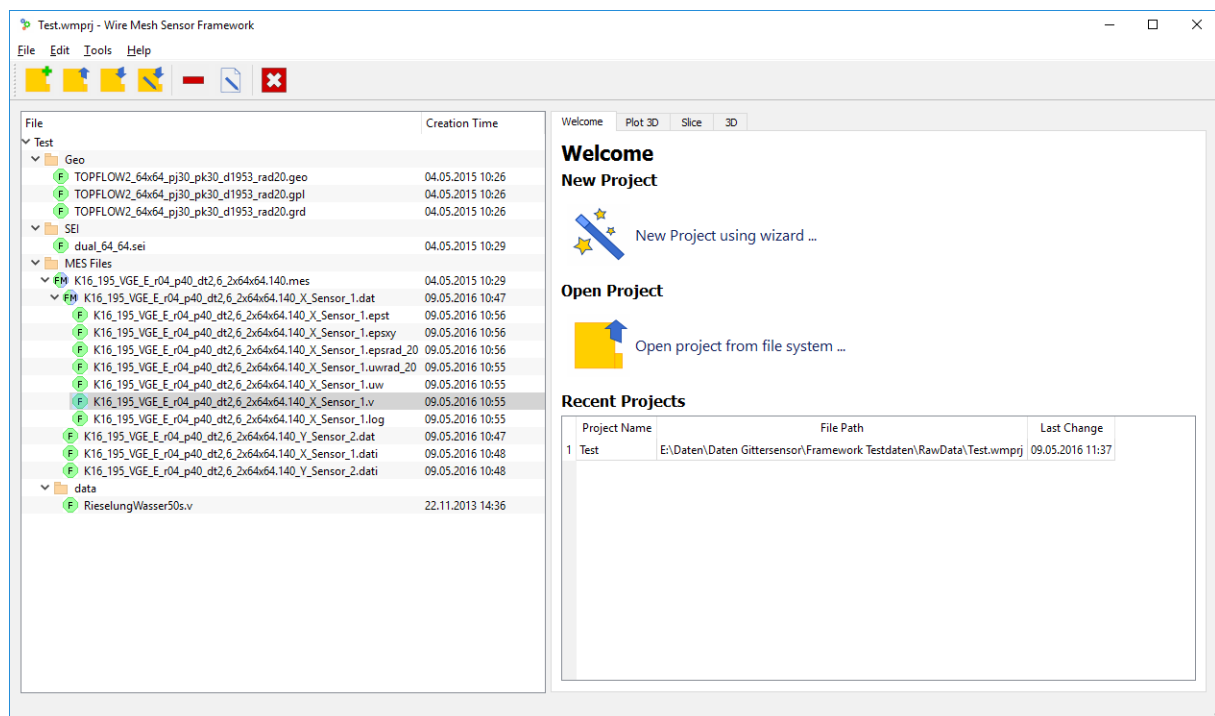


Figure 6: The main window of the Wire Mesh Sensor Framework

The right side of the main window is the view panel. It shows a visualization of the selected files in the project tree. Usually, the framework exposes different types of visualizations for every file. These visualizations can be selected in the register card at the top of the viewing area. At the start-up, the framework contains only a welcome screen which is described later in the document.

The third area of the main window is the toolbar and the program menu at the top. The main menu offers the following functionality:

- **File**
  - **New project ...:** Creates a new empty project. This will trigger the new project wizard described later in the document.
  - **Open project ...:** Opens an existing project from the file system.
  - **Save project ...:** Saves the current project to the file system. The program asks for a file name if it wasn't already saved.
  - **Save project as ...:** Saves the project under a new file name.
  - **Exit:** Closes the application. If the project file was modified, the framework asks to save the file.
- **Edit**
  - **Add files...:** Adds a list of files to the current node. The selected node must be a folder or a file node. Imported files are just links to files in in file system. They are not copied to the project folder.
  - **Create folder:** Creates a new folder in the project tree. These folders are not saved in the file system.
  - **Rename node:** Changes the name of the current node. Works only for folders and the project nodes.
  - **Delete node:** Deletes the current node. If there are any existing files in this node, the framework asks for their deletion.
  - **Project notes...:** Shows a dialog with the project notes and the authors name.
  - **Settings:** Shows a settings dialog of the wire mesh sensor framework application. Currently it only contains a list of all installed data providers.
- **Tools**
  - **SEI Editor:** Opens the SEI-file editor. A SEI-file defines the layout of a sensor matrix. See section 6.1 for more information about the SEI-editor.
  - **Mask Editor:** Opens the mask editor. See section 6.2 for more information about the mask editor.
  - **GEO Module:** Opens the GEO-file generator module. GEO-files are used to describe the geometry of a single wire mesh sensor. See section 5.3 for more information about the geo module.

- **Help**
  - **User manual...:** Open this user manual.
  - **About...:** Shows the program version and copyright information.



Figure 7: The main toolbar.

The toolbar buttons (Figure 7) are shortcuts to items in the main menu and they provide the following functionality (from left to right order):

- **New project.**
- **Open project...**
- **Save project**
- **Save project as...**
- **Delete node**
- **Edit project notes...**
- **Exit Application**

### 4.3 Welcome Screen

This screen shows a list of recently edited projects. They can be loaded by double clicking on them.

Furthermore, the welcome view consists of two buttons:

- **New project using wizard:** Creates a new project using the project wizard.
- **Open project from file system:** Opens a project from the file system.

### 4.4 Project Tree

The project tree is a hierarchical representation of the current project. It consists of folders, file nodes and modules. The project tree uses the following icons for each node:

- **Folders:** Folders have no special functionality. Folders group files and other folders. They are not part of the file system.
- **Files:** Files are imported by the user or generated by a module. A green file node indicates the existence of a file in file system. An orange node indicates the non-existence of a file in the file system.
- **Module:** Represents a single module if more than one module is applied to a file node. A module node is always a sub-node of a file node.
- **File/Module:** This node represents a file with a single module applied to it. Multiple node applications are represented by module sub-notes under a file node.

File	Creation Date
▲ <b>New Project</b>	
▲ <b>Sensor Information</b>	-
F test.sei	Fr Jan 11 11:06:30 2013
▲ <b>Geometry</b>	-
F test.geo	Mi Feb 26 12:38:59 2014
F test.ggd	Mi Feb 26 12:38:59 2014
F test.gpl	Mi Feb 26 12:38:59 2014
F test.grd	Mi Feb 26 12:38:59 2014
▲ <b>Measurement files</b>	-
FM test_001.mes	Mi Feb 26 12:37:51 2014
FM test_001.dat	Di Mrz 4 10:54:37 2014
FM test_002.mes	Mi Feb 26 13:48:00 2014
FM test_002.dat	Di Mrz 4 10:54:52 2014
F test_002.epst	Mo Mrz 17 16:31:54 2014
F test_002.epsxy	Mo Mrz 17 16:31:54 2014
F test_002.eprad_5	Mo Mrz 17 16:31:54 2014
F test_002.uwrad_5	Di Mrz 4 11:37:59 2014
F test_002.uw	Di Mrz 4 11:37:59 2014
F test_002.v	Di Mrz 4 11:37:59 2014

Figure 8: The project tree consisting of several imported and generated files.

By left clicking on the tree view a context menu with the following operations can be opened:

- **Create file(s):** Creates a file or a set of files using an editor or a module. This works for folder or root nodes only. With this option, new sensor information files or geometry files can be created.
- **Apply module...:** Applies a module to a file node (works for file nodes only).
- **Rename Node:** Renames the node (works for folders and root nodes only).
- **Delete Node:** Deletes the selected node. If the selected node is a file node it asks for the deletion of the file itself. This operation cannot be undone! (works for all nodes except the root node).
- **Copy Node:** Copies a node. For file nodes only the link is copied not the file itself (works for folders and file nodes).
- **Paste node:** Pastes a copied node (works for file and folder nodes only).
- **Add files...:** Adds one or more file links to the current node (folder and root nodes only).
- **Create Folder:** Creates a new folder (works for folder and root nodes only).
- **Open corresponding folder:** Opens the systems file manager with the folder containing the selected file (works for file nodes only).
- **Use node as template for batch processor...:** Uses this node as a template for the batch processor described later (works for file nodes only).



## 4.5 Project Wizard

The project wizard provides a convenient way to create new projects. These newly created projects contain all needed files for further data processing.

The wizard will ask for the following information:

- Author: The authors name.
- Description: The description of the project. Any text can be entered here.
- Project path: The path where the project is saved.
- SEI-file: The sensor information file used in the project.
- Geometry files: A set of geometry files.
- Measurement files: A set of \*.mes files.

The detailed usage of the wizard is described below.

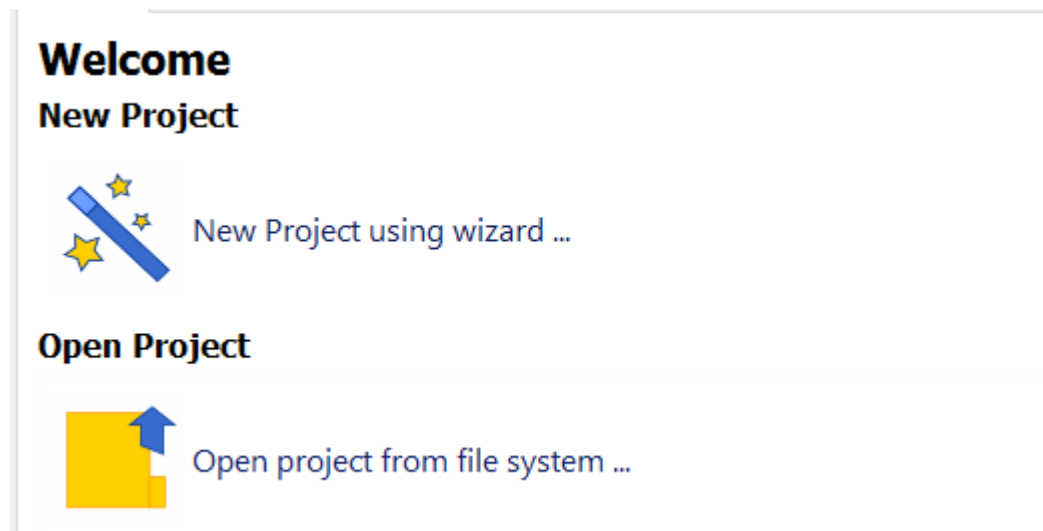


Figure 9: Welcome view.

First, you have to open the wizard. Click on the “*New Project using wizard*” button on the welcome page.

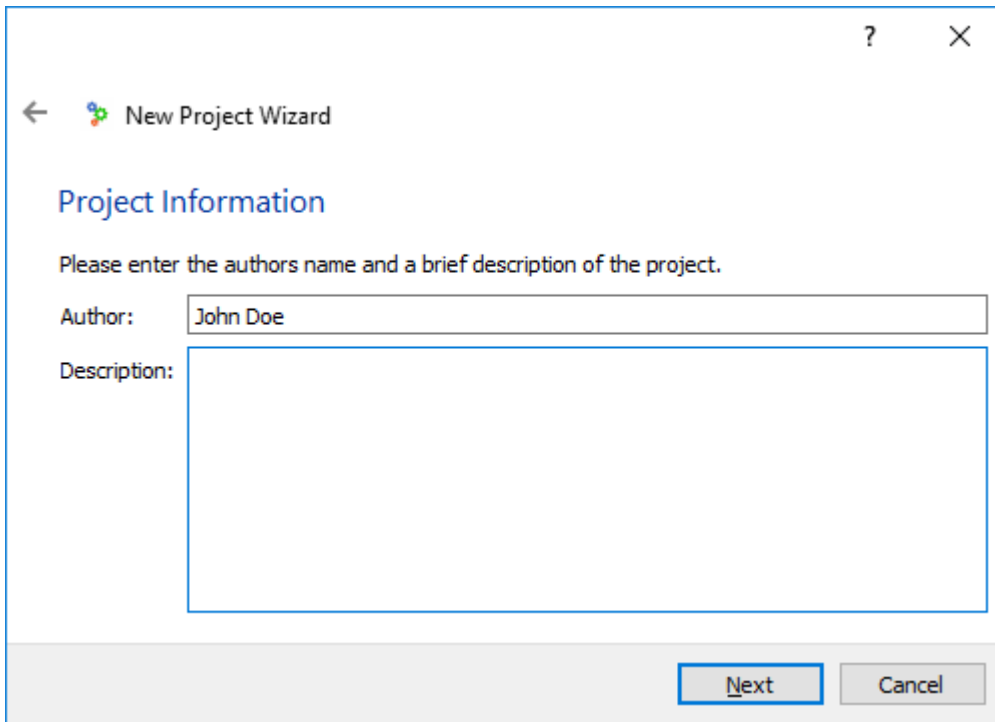
### Select a Project Template

After you have started the *Project Wizard*, you can choose between two types of project templates: an empty project or a simple project skeleton. To create a project with all the needed files, select the “*Simple Project File*” item and click *Next*.

### Project configuration

On this wizard page, you have to enter the authors name and an optional description text for the project. Click “*Next*” to continue.

The author and the description field can also be edited later in the project notes dialog (*Edit* → *Project notes...*).

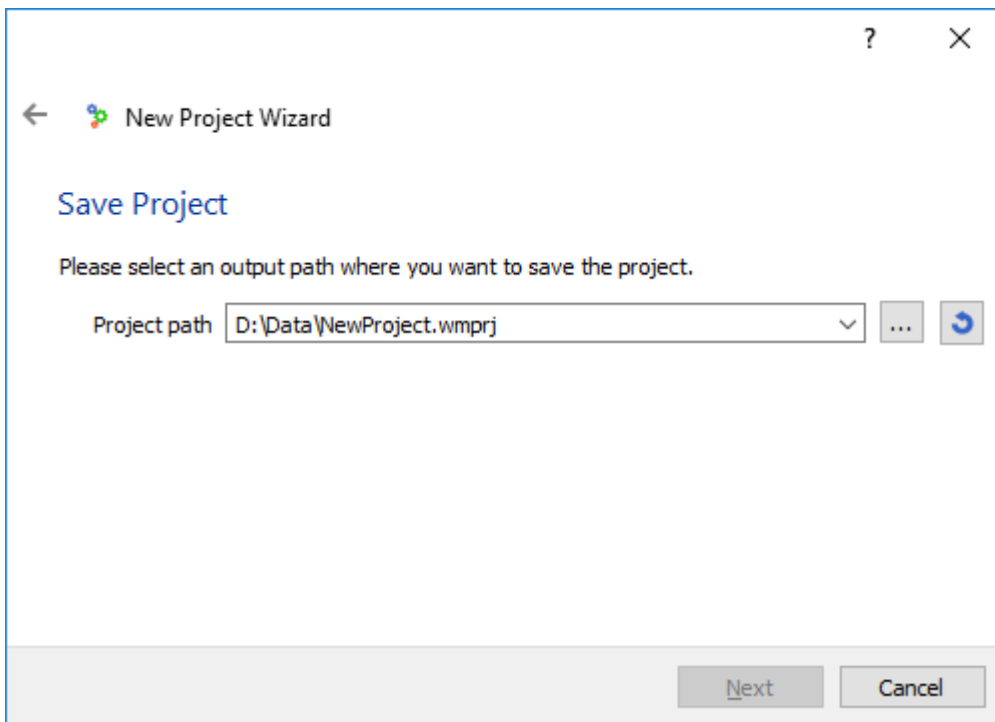


The screenshot shows a dialog box titled "New Project Wizard" with a back arrow and a help icon. The main heading is "Project Information". Below it, the instruction reads: "Please enter the authors name and a brief description of the project." There are two input fields: "Author:" with the text "John Doe" and "Description:" which is an empty text area. At the bottom right, there are two buttons: "Next" and "Cancel".

Figure 10: Author and description fields.

### Project File Path

On this page you can enter a path where the project file will be saved. Later, all files added to the project will be saved relative to this project file. It's recommended to use a local file path and not a network share. Network shares are known to be slow when working with the framework.



The screenshot shows the same dialog box, now at the "Save Project" step. The instruction reads: "Please select an output path where you want to save the project." There is a "Project path" label followed by a text box containing "D:\Data\NewProject.wmprj". To the right of the text box are three icons: a dropdown arrow, an ellipsis "...", and a refresh icon. At the bottom right, there are two buttons: "Next" and "Cancel".

Figure 11: Project file path.

## Add sensor information file

To correctly extract the \*.dat files from the measurement files, the framework needs to know the position of each sensor within the sensor matrix. The position is described in the SEI-file.

You have two options here:

- Load an already existing SEI-file from the local file system.
- Create a completely new one using the integrated SEI editor.

A detailed description the SEI editor can be found in the SEI Editor chapter (6.1). The format of the SEI-file is described in chapter 7.

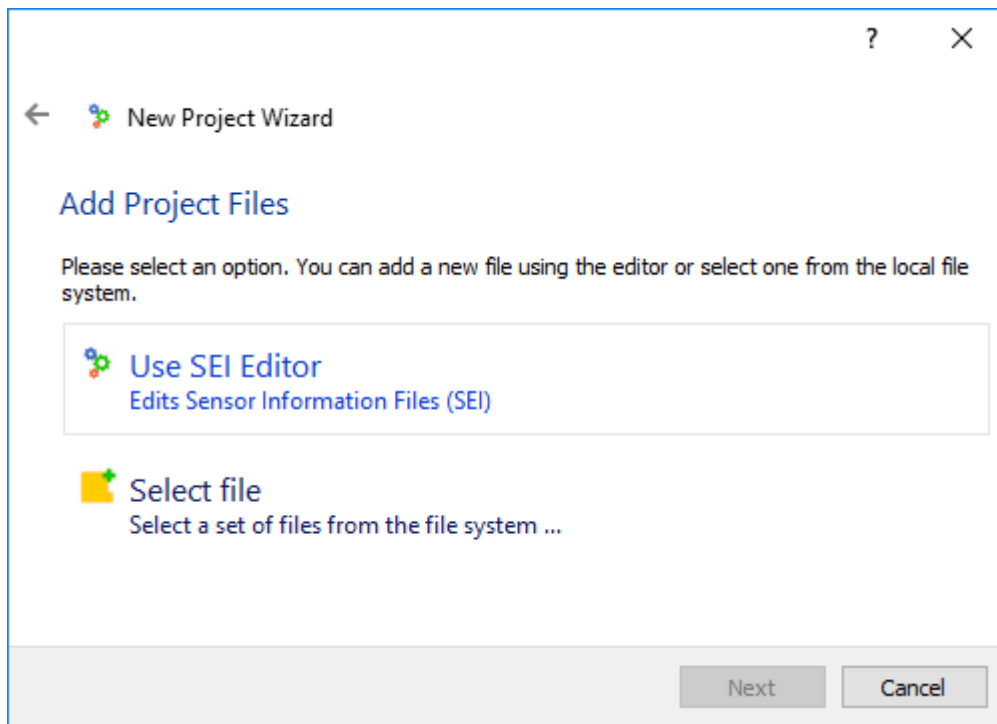


Figure 12: Use the SEI Editor or select a file from the file system.

## Add geometry files

The framework does not only need to know the positions of the sensors within the sensor matrix, it also needs to know the exact geometry of each sensor. In chapter 7 a detailed description of the geometry file format can be found.

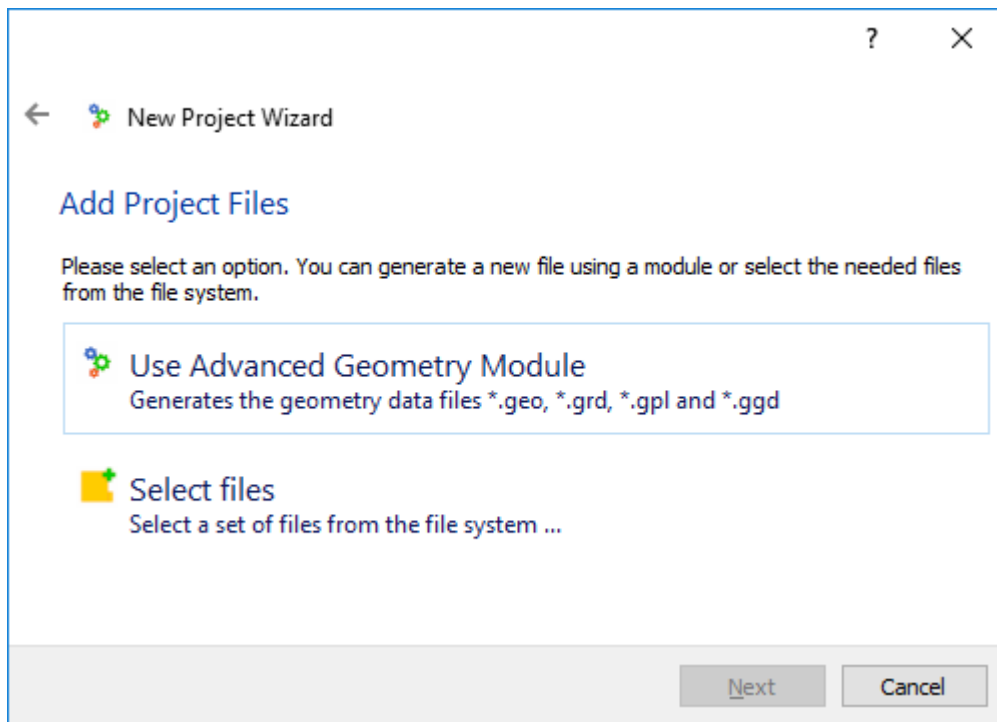


Figure 13: Use GEO module or select a set of files from the file system.

The next dialog lets you run the geometry module dialog. It creates the geometry files for the project. To do this you have to add a geometry file to the project. Like in the SEI file wizard page, you have to options: Load an already existing set of geometry files from the local file system or create a completely new set using the integrated geometry module.

The parameters of the geometry module are described in the Module chapter. Click "*Run*" to add them to the project.

A detailed description of the geometry module can be found in chapter 5.3.

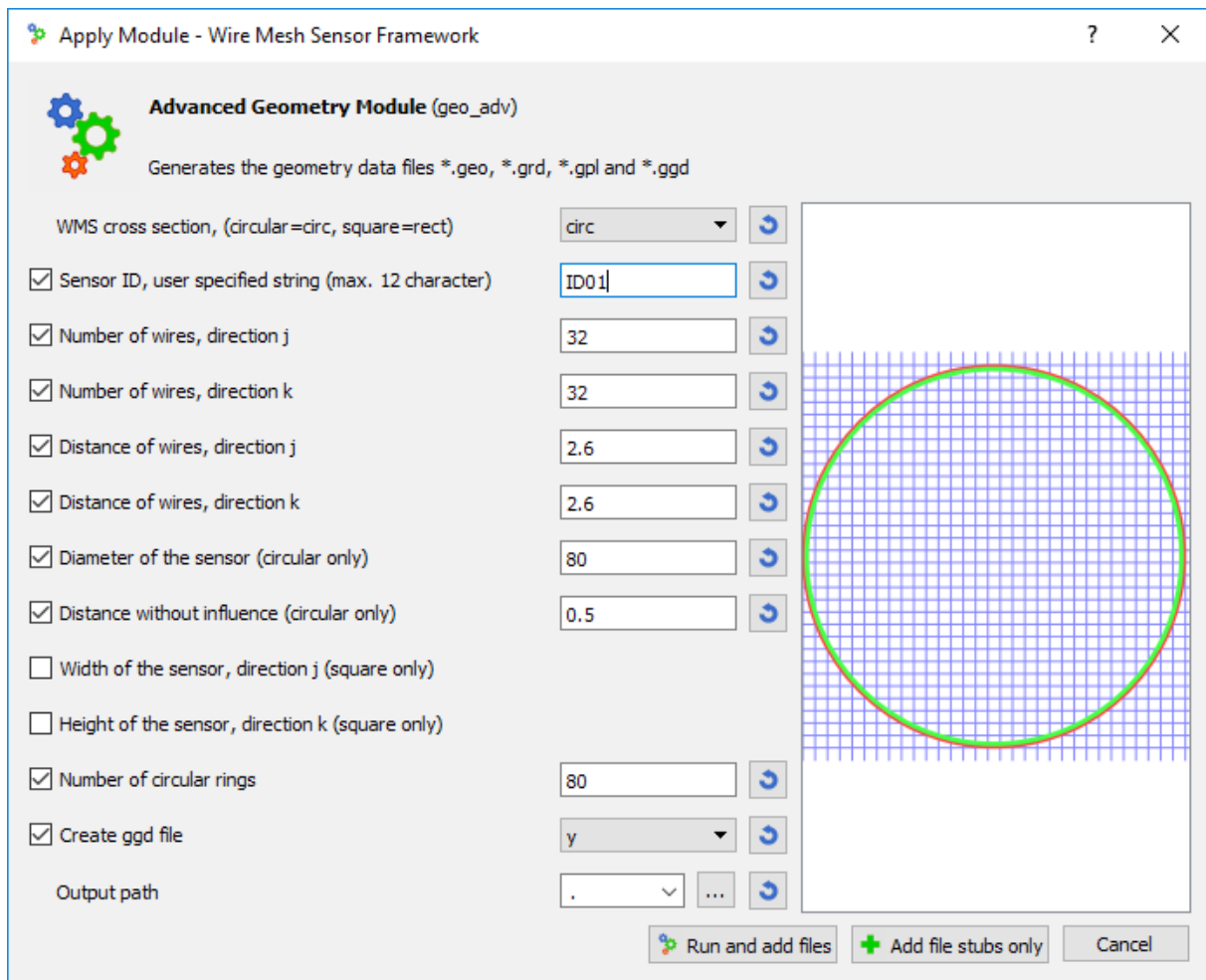


Figure 14: The geometry module in action.

### Add measurement files

In the last step, the project wizard asks for measurement files to be added to the project. You can select one or more files here. After this step the project will be generated.

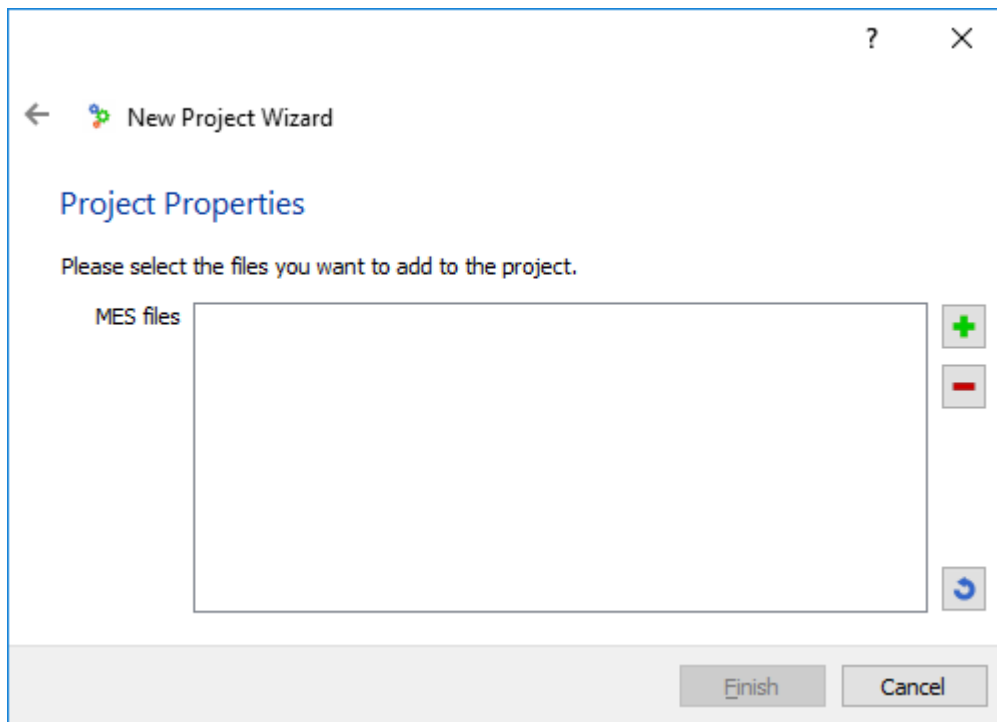


Figure 15: The wizard asks for measurement files.

## 4.6 Project Notes

To edit the project notes, click on *Edit* → *Project notes...* The dialog contains the name of the author and general notes about the project.

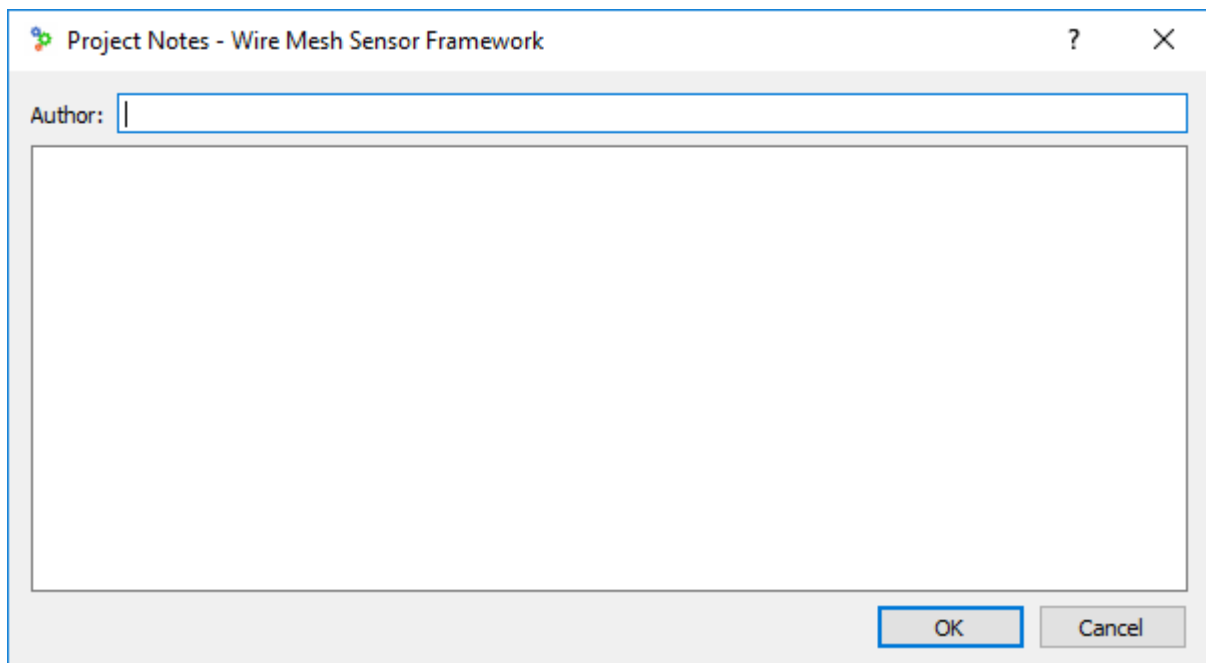


Figure 16: Project description and author.

## 4.7 Applying modules

Modules can be applied to files by right clicking on a project file and selecting “*Apply Module* → ...”. The framework shows only the modules which are compatible to the current file type.

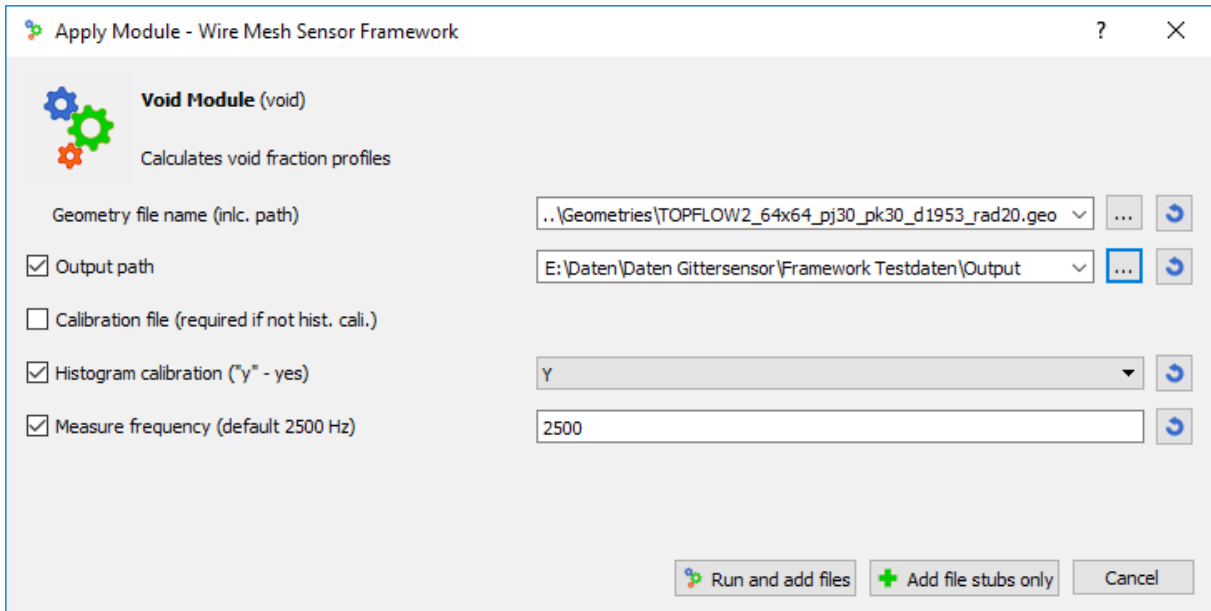


Figure 17: A typical module input mask (Void module in this example).

The module dialog shows an overview of all editable parameters. Each parameter can be activated or deactivated by clicking on the left check box. Deactivated parameters are not passed to the module.

The module dialog consists of several buttons at the bottom:

- **Run and add files:** Runs the module and adds the generated nodes to the project.
- **Add files without running module:** Adds only the file nodes generated by the module without running the module itself.
- **Cancel:** No module is applied to the file and the dialog closes.

## 4.8 Node views

The framework supports various visualization types. Which visualization is shown depends on the selected file type. The integrated views and some commonly used widgets are described below.

### Colour Scale

The colour scale widget is used in several views to configure the currently selected colour palette.

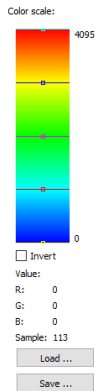


Figure 18: Colour palette. The Jet colour palette is selected.

The widget shows a linear colour map defined by several control points. Colour samples between these control points are linearly interpolated. They can be added, removed or modified. You can edit control points by right clicking on them and selecting *“Edit control point...”*.

New control points can be added by right clicking on the colour bar and selecting *“Add control point...”*. Control points can also be deleted by right clicking on them and clicking *“Delete Control Point”*. The sampling position of a control point can be changed by dragging it to a different position.

The *“Load...”* and *“Save...”* button can be used to load and save a user defined colour map:

- **Load...:** Loads a colour palette from the file system. The framework comes with a set of predefined colour palettes (e.g. blue, blue inv, hot, jet, ocean, ...).
- **Save...:** Saves the current colour palette to a file.

### 3D Plot View

The *3D Plot* shows a three-dimensional surface plot of the actual frame. You can use the player component at the bottom to select individual frames. The *“Play”*-Button starts an animation of the measured data.



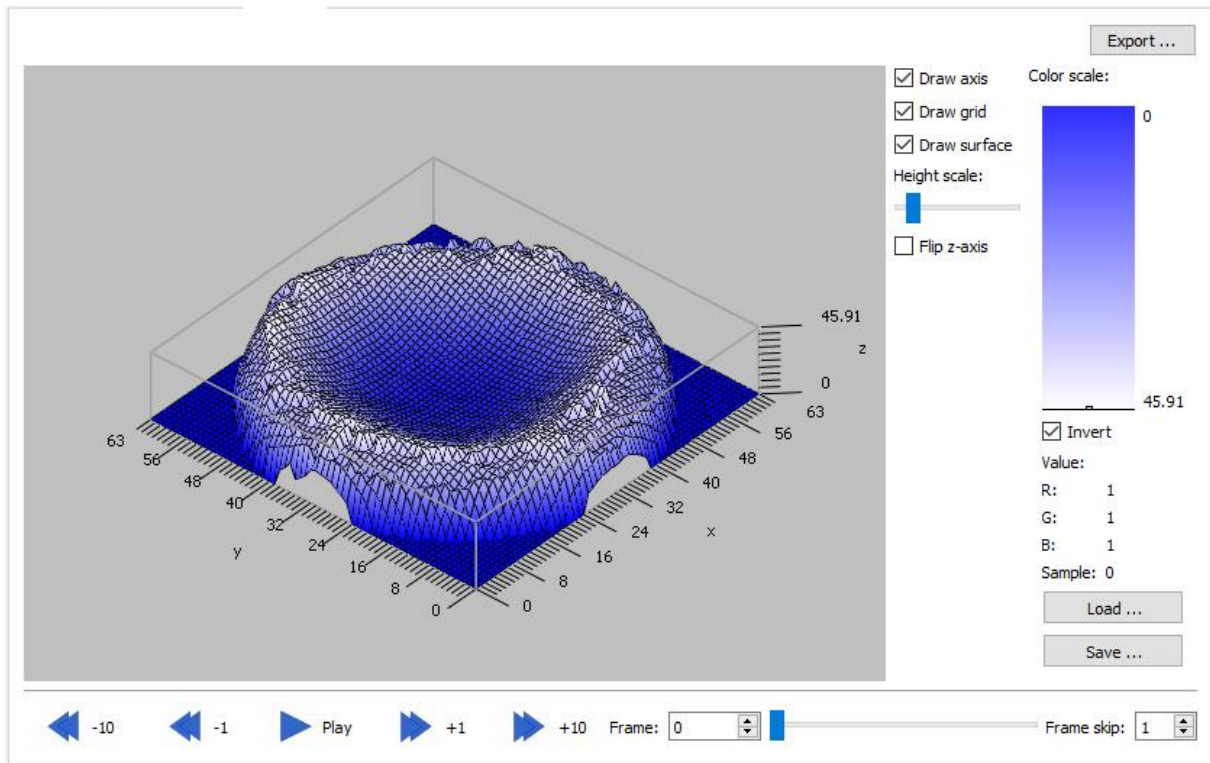


Figure 19: Typical surface plot.

The colour scale widget is described in section 4.8. The surface plot widget itself supports the following functionality:

- Right click and drag: Moves the diagram.
- Left click and drag: Rotates the diagram.
- Mouse wheel: Zooms the diagram.

The control panel on the right next to the colour palette offers the following controls:

- **Draw axis:** Draws the plot axis.
- **Draw grid:** Draws the surface grid.
- **Draw surface:** Draws the coloured plot surface.
- **Height scale:** Scales the plot in the z-direction.
- **Flip z-axis:** Flips all z values.

## 4.9 Plot2D view

The *Plot2D view* shows a two-dimensional plot of the selected node.

The *x-axis* and *y-axis* for the diagram can be selected from the two drop-down menus. Some files contain multiple tables or diagrams. The used data source can be selected using the “*Table*” drop-down box.

With “*Set zoom rectangle...*” a distinct rectangular viewing area can be selected. Click left and drag the mouse in the plot area to zoom in (a rubber band with the new zoom area appears). Click right to restore the last zoom level.

The diagram can be exported to different formats (e.g. as an image or a vector graphics) by clicking “*Export...*”.

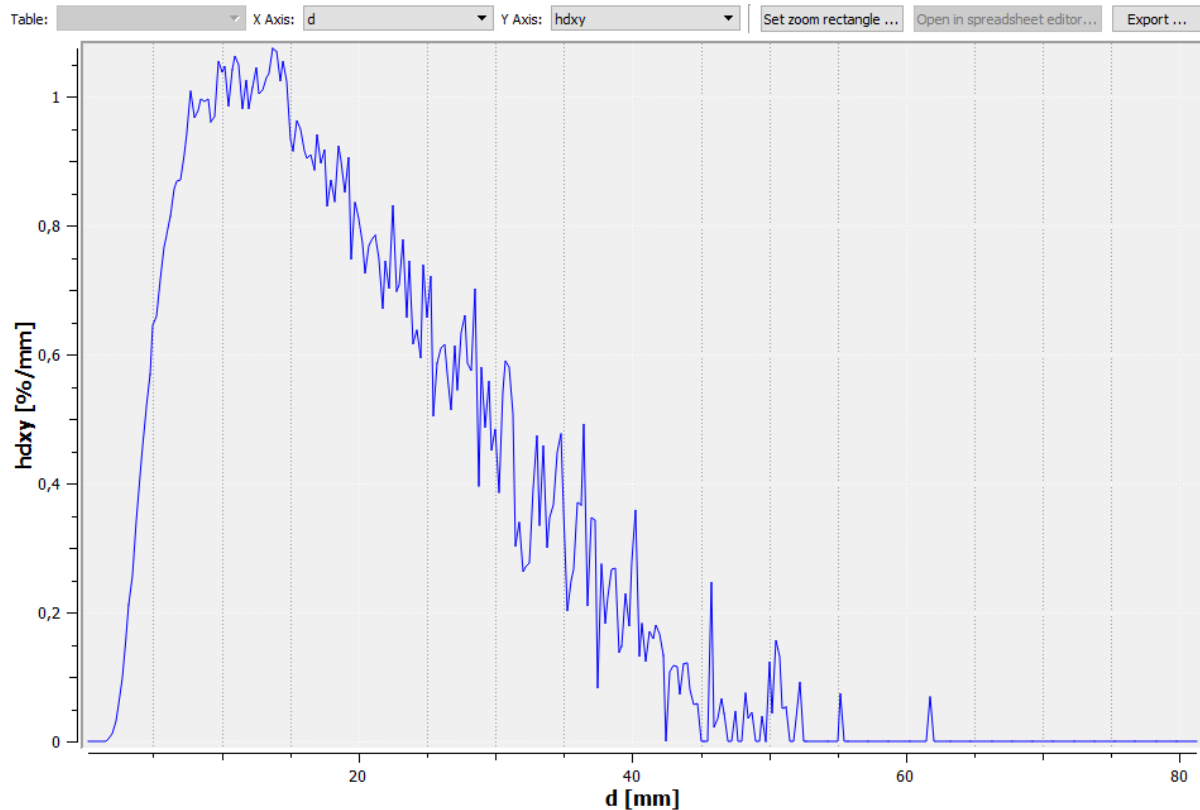


Figure 20: Plot2D view.

#### 4.10 Slice view

The slice view is also a two-dimensional representation of the measurement data. It supports the visualization of horizontal and vertical data slices.

Controls at the top:

- **Geo mask:** selects a geometry mask.
- **Slice scale:** Scales the two slices.
- **Frames:** Sets the number of frames shown in the slice view.
- **Export:** Exports the current view as a MPEG video or a sequence of images. The user can choose between the top, horizontal or vertical view.

Controls at the bottom:

- **-10, -1, +1, +10:** Seeks to next or previous frame.
- **Play:** Plays an animation of the data.
- **Frame:** Sets the currently shown frame.
- **Frame Skip:** Sets the value to skip while playing the data.

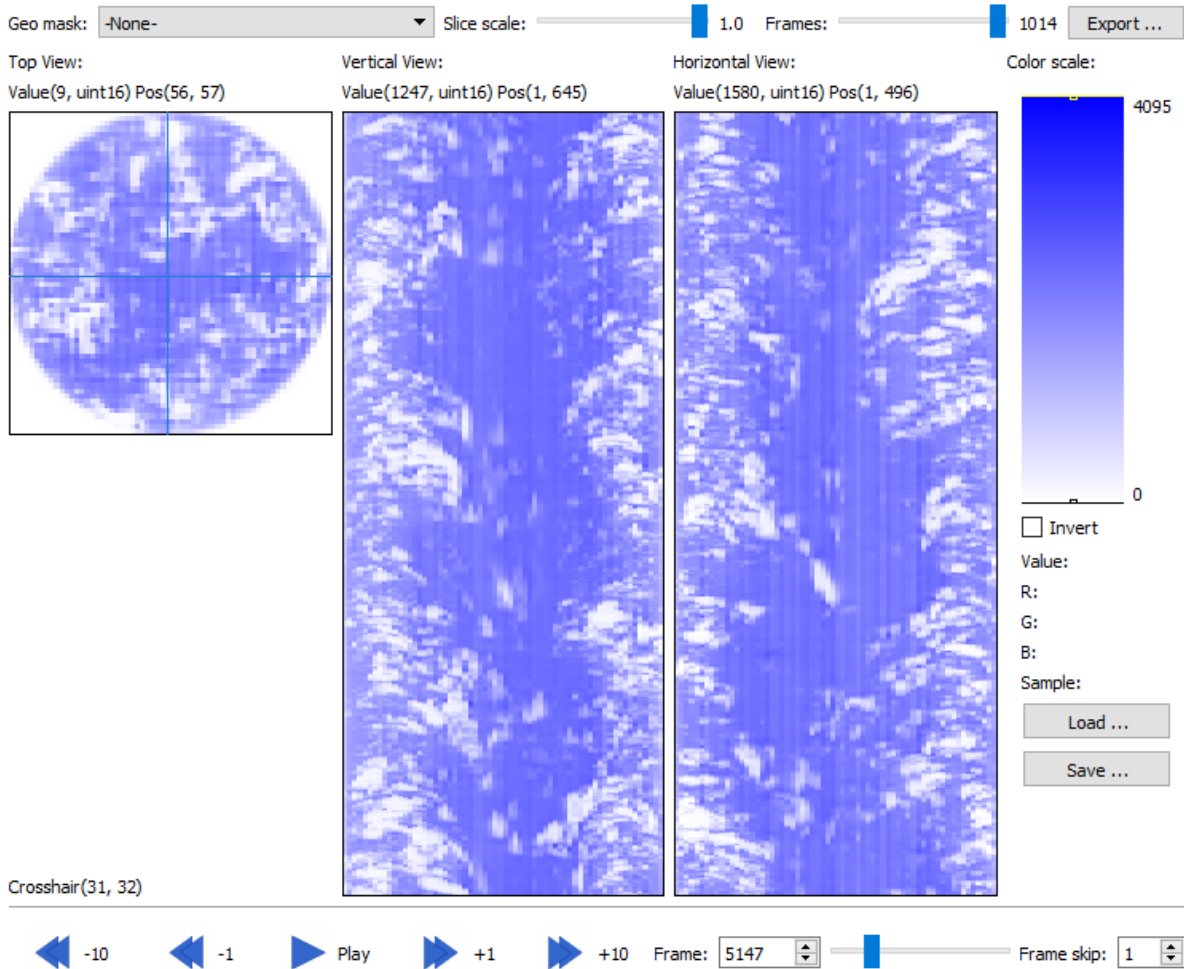


Figure 21: Sliced view.

You can edit colour palettes on the right side of the slice view. See 4.8 for more information.

#### 4.11 3D Voxel view

The 3D view shows an Iso surface representation of the data. The following controls are available:

- **View type:** Iso surface: classical Iso surface lit by a virtual light; Transparent iso surface: Same as iso but the far side is also shown.
- **Invert:** Inverts the data values: The liquid phase becomes the gas phase and vice versa.
- **Transparent:** Shows the volume in transparent mode. Uses Beers Law to calculate the attenuation of light through the volume.
- **Perspective projection:** Changes between an orthographic and a perspective projection.
- **Frame skip:** Shows only the  $n^{\text{th}}$  frame.
- **Threshold:** Sets the threshold (boundary) for the Iso surface.
- **Ray step size:** Step size of the ray-casting algorithm.

- **Slice Min/Max sliders:** Cuts the volume into horizontal or vertical slices very similar to the slice view.
- **Flip X/Y/Z:** Flips the axis of the volume.
- **Flow:** Selects the flow direction. Can be horizontal or vertical.
- **Select background color...:** Background color of the 3D view.
- **Select foreground color...:** Color of the Iso surface.

The “Export...” button supports MPEG video and image sequences, similar to the slice view.

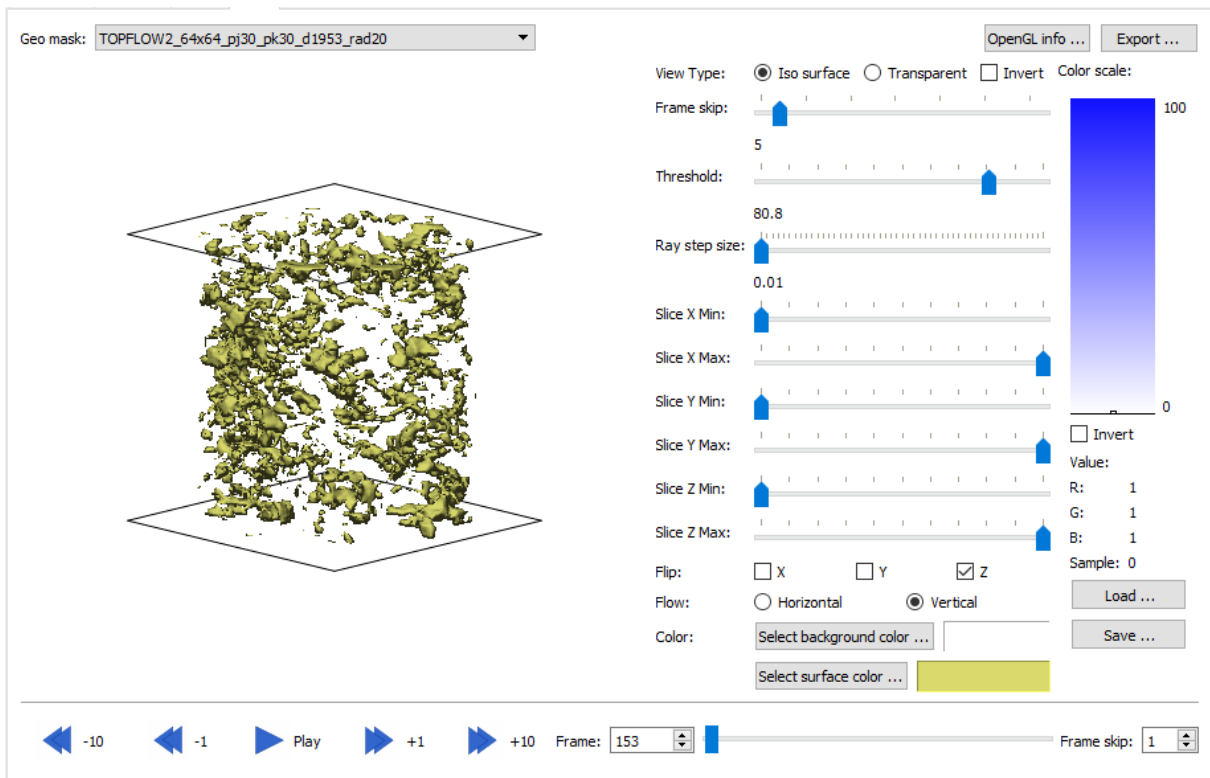


Figure 22: Voxel 3D view.

## 4.12 Table view

Shows a tabular grid of the measurement data values. The shown record can be selected in the drop-down box at the top of the view.

Record: 1

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	588	1024
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	423	847	1369	1604
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	737	1279	1548	1726	1865
7	0	0	0	0	0	0	0	0	0	0	0	500	1196	1608	1857	1924	1999	2074	
8	0	0	0	0	0	0	0	0	0	0	537	1203	1617	1707	1839	1836	1751	1758	
9	0	0	0	0	0	0	0	0	0	706	1406	1826	2003	1924	1825	1791	1860	1908	
10	0	0	0	0	0	0	0	0	0	738	1366	1555	1756	1729	1701	1781	1721	1751	1799
11	0	0	0	0	0	0	0	0	628	1416	1730	1706	1733	1740	1735	1783	1788	1797	1910
12	0	0	0	0	0	0	0	735	1422	1725	1744	1641	1751	1839	1749	1854	1836	1927	2151
13	0	0	0	0	0	0	406	1473	1775	1688	1781	1757	1822	1904	1753	1841	1851	2112	2170
14	0	0	0	0	0	1060	1586	1699	1569	1695	1616	1717	1773	1875	1949	1937	1982	2061	
15	0	0	0	0	1118	1605	1852	1749	1724	1862	1732	1863	1927	1908	2201	2199	2199	2253	
16	0	0	0	468	1697	1769	1738	1788	1805	1953	1847	1945	2233	2083	2240	2249	2263	2305	
17	0	0	0	1138	1804	1610	1679	1756	1727	1857	1727	2090	2148	2021	2140	2108	2123	2185	
18	0	0	434	1508	1738	1596	1675	1721	1682	1774	1926	1988	2038	1951	2061	2022	2041	2117	
19	0	0	1174	1726	1827	1738	1774	1835	1987	2170	2033	2133	2192	2080	2167	2187	2195	2251	
20	0	0	390	1558	1700	1740	1664	1659	1732	1679	2018	1891	2025	2086	1999	2052	2057	2064	2124
21	0	0	1148	1853	1653	1966	1787	1818	1903	1815	2185	2026	2171	2235	2095	2214	2189	2185	2278
22	0	0	1577	1825	1701	1920	1762	1792	1838	1861	2147	1993	2099	2172	2094	2173	2157	2138	2216
23	0	622	1663	1759	1656	1864	1748	1819	1836	1966	2176	2061	2099	2176	2068	2156	2138	2121	2179
24	0	1221	1773	1741	1790	2018	1797	1911	1957	2029	2177	2073	2144	2269	2132	2229	2216	2202	2293

Figure 23: Table View.

### 4.13 Batch Processor

The batch processor allows the user to use a sub-tree of a project as a batch template. The batch dialog is separated into two panels. The left panel consists of a list of files which are used as the input for the node tree on the right side. The tabular file list on the left side also consist of an overview of all changed parameters for each batch item.

Controls at the bottom:

- **Add files...:** Adds a list of files to the batch table.
- **Add all entries to project...:** Adds all batch entries to the project tree. The framework will ask for a new directory where to populates the new file nodes. *This new sub-tree can be huge if the file list is long!*
- **Add current entry to project...:** Adds the currently edited batch entry to the project.
- **Run all...:** Runs the batch processor for all files.
- **Run current:** Runs the currently selected batch entry.

Context menu entries for the batch list (right click on the file row):

- **Copy file entries:** Copies batch items.
- **Paste file entries:** Pastes copied batch entries.
- **Delete file entries:** Deletes the current batch entry.
- **Paste Parameters:** Paste copied parameters.
- **Add files...:** Add more files to the batch list.

Additional context menu entries for the parameter rows (right click on the parameter row):

- **Copy parameter:** Copies the selected parameter.
- **Paste parameter:** Pastes the selected parameter to another batch entry.
- **Delete parameter:** Deletes the entry and restores it to the default value.

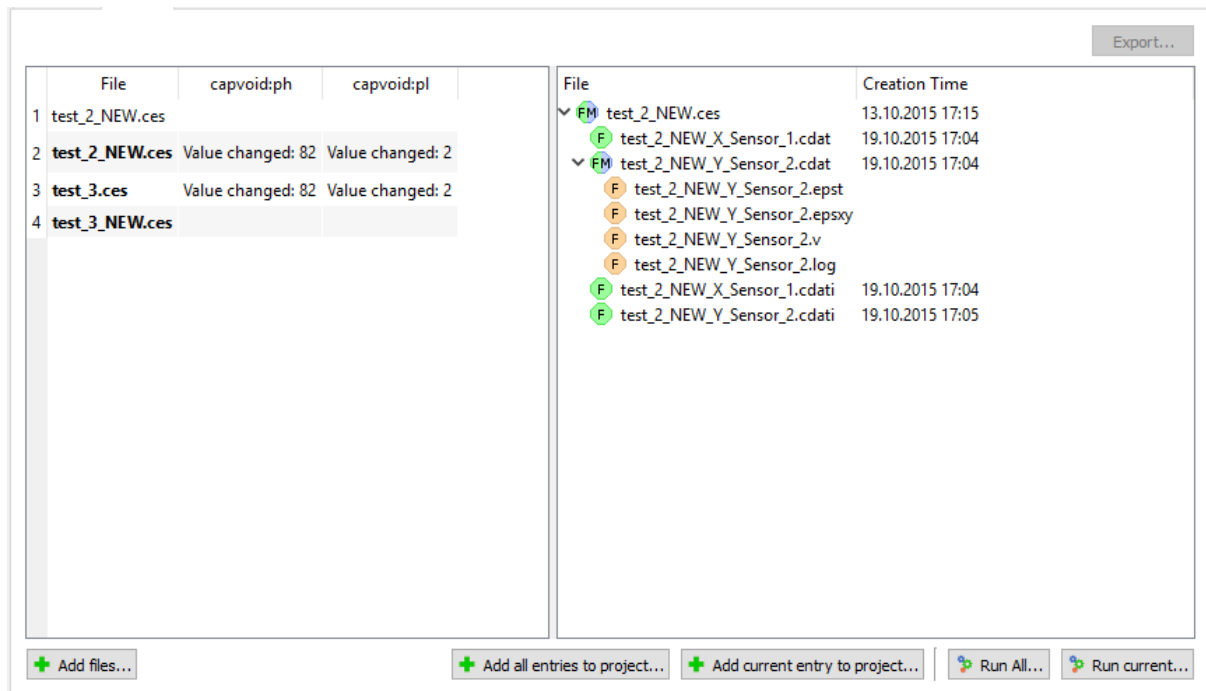


Figure 24: The batch processor view.

The tree view on the right side behaves similar to the project tree in the main window. The following options are available:

- **Apply module...:** Applies a module to a file.
- **Edit module parameters:** Edits the parameter. This will change the parameter table on the right side of the batch view.

## 5 Modules for the Wire Mesh Sensor Framework

### 5.1 General remarks

All modules, explained in the following chapters, are written in *Borland Delphi Professional 7.0* and C++ and compiled for *Microsoft Windows* environment (XP or Server 2003 and higher). The program modules use parameters for data setup and work in batch mode without user interfaces. The *WMS framework* helps to integrate the different modules into one graphical user interface (*GUI*) allowing the user to specify the necessary parameters in a table, pick and drag files into the command line, generate batch files and use wild cards. For each module a \*.minf (module information file) is provided containing all required information about input and output parameters, input and output files and paths. The framework is open for the integration of new user defined modules (written in any compiler language), as long as the conventions for the \*.minf-files are fulfilled.

The following descriptions and the software modules are made for the case of a 64 x 64 wire-mesh sensor with a 64 x 64 electronic device. Most of the routines require configuration files for the wire-mesh sensor geometry.

The names of the generated files consist of the same body like the measurement file and the corresponding extension (see below). All routines described below use a \*.log file in *ASCII*-format to save information about the calculation (e.g. date, time, names of the geometry files, thresholds, bubble numbers) and error messages.

**NOTE:** If the modules are used stand-alone, all parameters start with a hyphen "-" (e.g.: "-fs:C:\Program Files\...").

### 5.2 FileConverter module, converts raw measurement files into single sensor files

The *FileConverter* module is used to convert and extract the packed 12-bit file format of the raw data \*.mes into binary data files for each single sensor \*.dat. The binary \*.dat files contain the measured values for all exported frames of a sensor with the dimension M x N as 16-bit words in the following dimension. The example is given for a 32 x 32 sensor:

Number of data word	Data word description
0	Column 1 of row 1 of frame 1
1	Column 2 of row 1 of frame 1
2	Column 3 of row 1 of frame 1
...	...
31	Column 32 of row 1 of frame 1
32	Column 1 of row 2 of frame 1
33	Column 2 of row 2 of frame 1

...	...
1023	Column 32 of row 32 of frame 1
1024	Column 1 of row 1 of frame 2
1025	Column 2 of row 1 of frame 2
...	...

Therefore, the resulting file size is 2-byte x M x N x number of exported frames.

The *FileConverter* module requires the following parameters:

Parameter	Description
fs	Input file name (including path)
fe	Sensor extraction info file name (*.sei) (including path)
sp	Output path (optional)
o	Output file name (optional)
f1	Frame start (optional)
f2	Frame end (optional)
j1	Point j1 coordinate (not required if *.sei-file is available)
k1	Point k1 coordinate (not required if *.sei-file is available)
j2	Point j2 coordinate (not required if *.sei-file is available)
k2	Point k2 coordinate (not required if *.sei-file is available)

The input file name is the name and path of the raw data file (\*.mes) to be converted. The sensor extraction file name (\*.sei) specifies the SEI-file which contains all necessary information about the single sensors location in the raw data matrix. This file is automatically generated by the latest software version of the WMS200 device. In case the SEI-file is not present it can be generated by the WMS *framework*. The SEI-file defines the dimension of the raw data matrix (e. g. 64 x 128 for two 64 x 64 sensors sampled in parallel mode) and specifies the start and end coordinates of the single sensor. The following example of a SEI-file for a set of two 64 x 64 sensors illustrates the file structure:

[MATRIX]	<i>section for raw data matrix</i>
INFO=dual sensor vertical pipe DN200	<i>free user defined text</i>
HEIGHT=64	<i>height of the raw data matrix</i>
WIDTH=128	<i>width of the raw data matrix</i>
[OUTPUTFILENAMELIST]	<i>section for output files _</i>
OUTPUTFILENAME1=_Sensor_1	<i>appendix for the first sensor output file name</i>
OUTPUTFILENAME2=_Sensor_2	<i>appendix for the second sensor output file name</i>
[SENSOR1]	<i>section describing first sensor</i>
ID=00	<i>additional ID number (no longer used)</i>
NAME=Sensor_1	<i>name of first sensor (same as file name extension)</i>
J1=0	<i>x-coordinate start point first sensor</i>
K1=0	<i>y-coordinate start point first sensor</i>
J2=63	<i>x-coordinate end point first sensor</i>
K2=63	<i>y-coordinate end point first sensor</i>
TYPE=X	<i>"X" defines the first sensor of a set of two</i>
RES1=	<i>not yet defined</i>
SERIAL=1071	<i>hardware serial number of the sensor</i>
INFO=Lower sensor	<i>free user defined text (e. g. sensor position)</i>
[SENSOR2]	<i>section describing second sensor</i>



ID=00	additional ID number (no longer used)
NAME=Sensor_2	name of the sensor (same as file name extension)
J1=64	x-coordinate start point second sensor
K1=0	y-coordinate start point second sensor
J2=127	x-coordinate end point second sensor
K2=63	y-coordinate end point second sensor
TYPE=Y	"Y" defines the second sensor of a set of two
RES1=	not yet defined
SERIAL=1072	hardware serial number of the sensor
INFO=upper sensor	free user defined text (e. g. sensor position)

The output path is optional. If it is not defined by the user, the same path as the input file path is used. The other parameters correspond to the SEI-file. They are unused if no SEI-file was selected. After the run of the *FileConverter* module, the number of \*.dat files specified in the SEI-files are generated.

### 5.3 Geo & GeoAdv module, generation of geometry configuration files

The geometry of the used sensor is important for later calculations of void fraction profiles, velocities etc. The geometry module calculates a set of geometry files for use in other modules.

The module geo.exe requires the following parameters:

Parameter	Description
cs	WMS cross-section, (circular=circ, rectangular=rect)
id	Sensor ID, user specified string (max. 12 character, optional)
nj	Number of wires, direction j
nk	Number of wires, direction k
pj	Distance of wires in mm, direction j
pk	Distance of wires in mm, direction k
ds	Inner diameter of the wire-mesh sensor in mm (circular sensors only)
xx	The distance without influence. <i>This parameter is only available in the GeoAdv module.</i>
dj	Width of the sensor, direction j (rectangular only)
dk	Height of the sensor, direction k (rectangular only)
nr	Number of ring shaped domains (see Fig. 1.4)
sp	Output path for the calculated geometry files (optional)

Output files of the geometry module:

File	Type	Description
*.geo	ASCII (Matrix file)	Geometry file for the measurement cross-section, e.g. ID01_64x64_mj26_mk26_d1953_rad80.geo description: 64 x 64 wires, 2.6 mm lattice spacing, 195.3 mm inner diameter of the sensor; the file contains 64 x 64 values: 0 – means that the measurement area around a crossing point is completely outside the measuring area; value > 0: the part of the active measurement area related to the general area of the sensor (cp. Figure 5)

*.grd	ASCII (multiple Matrices)	Geometry file for the ring-shaped domains, e. g. ID01_64x64_mj026_mk026_d1953_rad80.grd description: the same as before + 80 ring-shaped domains; content: 80 tables with the same structure as before including the weight coefficients for the single ring-shaped domains (cp. Figure 5).
*.gpl	ASCII	Log file of the geometry module, type ASCII, containing all information about the used parameters and the resulting files.

## 5.4 Void module, calibration and void fraction distributions

The module *Void* requires the following parameter:

Parameter	Description
fs	Measurement data file including path (*.dat, *.v)
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
sp	Output path (not required, if blank = path of the measurement file)
fc	Calibration file, either water calibration file (*.dat) or calibration file from previous run (*.uw)
hc	Type of calibration ("y" - histogram calibration, "" - requires calibration file)
mf	Measure frequency, default 2500 Hz

The void-module is able to handle uncalibrated measurement files (\*.dat) or calibrated void-files (\*.v) as input files (fs). For the later the calibration procedure is skipped and only void fraction profiles are calculated. The geometry file name (including path) has to be specified as parameter fg. The module expects three geometry files (see 5.3). Output path (sp:) is optional. For calibration of the measurement data a calibration file has to be specified. This can be either a \*.dat-file for pure water flow or calibration file \*.uw from a previous run of the module. The \*.uw is generated by the first run of the module from a water measurement file. The use of the \*.uw file is recommended since it is a short condensed file containing only averaged water values for each crossing point. Instead of using a calibration file the user can specify the parameter hc. If this parameter is set to "y" (yes) the program runs through the measurement file and performs the histogram calibration routine (cp. 3.1).

**ATTENTION:** Histogram calibration has to be used very carefully! It can only be applied, if the user is absolutely sure that there is really each crossing point covered with water for many times during the measurement. Therefore, histogram calibration cannot be applied for horizontal stratified or annular flow situations for instance! It is advised to have a look to the \*.uwrاد\_\* files, which are generated after a histogram calibration. If the calibration values vary too much, the histogram calibration has failed (cp. 3.1).

The program uses a noise filter (cp. 3.3) with a threshold. Either this limiting value is set to a fixed value (e.g. 10%), which is the case if histogram calibration was used or it

is taken as a result from the assessment of the frames of a calibration file (\*.dat) if file calibration has been activated.

The module first generates a binary file with e. g. 64 x 64 x 25000 void fraction values (where 25000 is the number of measured frames), called *v*-file, stored in byte-format. In this file, the void fraction values for each crossing point are stored as a number 0 to 100, corresponding to 0% and 100% respectively. Matrix points with the value 255 are outside of the sensor area. Furthermore, the void module generates the following averaged void fraction distributions:

File	Type	Description
*.epsxy	ASCII (Matrix)	The files contain the time averaged volumetric gas fractions in percent as matrix for the cross-section (e. g. 64 x 64 values). The points outside of the pipe cross-section have the value 0% (Table 1).
*.epst	ASCII (Table)	It contains the cross-sectional averaged gas fractions over time steps presented by two columns: The left column indicates the time step in s while in the right column the cross-sectional void fraction in percent is stored (Table 2).
*.epsrad_80	ASCII (List of Matrixes)	T time and azimuthally (inside of e. g. 80 ring-shaped domains) averaged gas fraction; Column 1 contains the centre radii of the respective ring in mm (see fig. 4). The right column contains the appropriate gas fraction in percent (Table 3).
eps_all.asc	ASCII (Log)	Additionally, the module adds one row for each run in the eps_all.asc file which includes the date and time of the calculation + the name of the measuring file + the time and cross-section averaged void fraction.
*.uw	ASCII (Matrix)	If the program was started with a calibration file (fc:) (*.dat) a *.uw-file with the same name as the calibration file, containing the values of the calibration matrix in dimensions x, y is generated. This file can be used as input file for evaluation of measurement data with the same calibration conditions (e. g. jw, tw...). If no calibration file has been specified but the parameter (hc:) has been set to "y" a *.uw-file with the name of the measurement file (fs:) is generated respectively (Table 4).
*.uwrاد_*	ASCII (Table)	If histogram calibration is performed a *.uwrاد_* file (e. g. *.uwrاد_80 for 80 rings) is generated, containing the azimuthally averaged calibration values from the *.uw file. This is very useful to check the quality of histogram calibration (cp. 1.1, Table 5).

The contents of the \*.epsxy file:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0.72	
0	0	0	0	0	0	0	0	0	0	0	0	1.64	5.50	
0	0	0	0	0	0	0	0	0	0	0	1.89	7.96	14.98	
0	0	0	0	0	0	0	0	0	0	3.21	10.29	16.23	19.37	
0	0	0	0	0	0	0	0	0	3.36	10.96	16.53	19.79	22.58	
0	0	0	0	0	0	0	0	3.06	10.31	16.22	19.83	21.98	24.56	
0	0	0	0	0	0	2.43	10.75	16.10	21.03	23.77	24.79	27.33		

0	0	0	0	0	0	0.64	8.40	15.66	20.19	23.17	26.07	28.06	29.02
0	0	0	0	0	0.10	7.42	15.18	20.52	23.23	25.85	28.05	29.74	30.76
0	0	0	0	0	5.60	14.37	19.71	23.72	26.08	27.78	30.12	32.35	33.23
0	0	0	0	2.80	12.20	18.78	23.12	25.53	28.71	30.65	31.53	34.36	33.29
0	0	0	0	7.11	15.93	20.69	24.06	28.32	29.65	32.58	33.63	35.15	35.39
0	0	0	3.03	13.68	19.19	23.29	27.39	30.63	31.34	33.41	34.47	36.25	37.33
0	0	0	8.57	18.04	21.84	25.67	29.48	31.95	33.86	35.02	36.34	37.60	39.05
0	0	3.61	14.25	21.12	26.01	28.41	31.09	32.86	35.34	36.78	38.41	38.73	40.35
0	0	7.43	18.02	24.67	27.91	29.51	32.74	33.86	36.45	38.01	39.03	38.98	41.17
0	1.93	11.77	20.31	25.91	29.55	30.74	33.69	35.55	37.08	37.29	39.58	41.10	41.33
0	4.43	15.19	22.52	26.78	29.13	31.32	34.86	35.92	38.10	38.94	39.15	40.66	40.18
0	8.20	18.29	23.26	27.12	30.03	32.36	35.57	35.98	38.31	39.15	40.46	41.13	40.92
0.53	11.17	20.14	24.69	28.03	30.29	33.01	36.54	36.15	38.33	39.93	39.64	40.67	41.12
2.93	12.80	21.76	26.42	29.93	31.73	33.68	35.76	36.78	38.02	38.96	39.83	40.68	43.07
4.96	15.00	21.77	27.13	31.22	32.95	34.64	37.44	38.07	38.06	39.51	40.67	41.61	42.90
7.33	16.06	23.12	28.44	31.97	33.95	35.89	36.59	37.47	38.11	39.19	40.84	41.45	41.37
9.04	17.42	23.93	28.37	31.82	33.62	35.70	37.70	37.65	38.20	39.35	41.14	43.38	43.27
10.21	18.41	24.72	27.19	30.58	33.17	35.88	37.14	37.54	38.92	39.92	41.87	44.15	43.09
11.26	19.42	24.76	27.60	30.63	32.71	35.78	37.24	37.77	39.13	40.00	42.07	42.89	43.71
11.82	19.83	24.52	27.07	30.50	32.88	35.10	35.86	36.71	38.12	40.98	42.84	43.83	44.20
....													

Table 1: Visualisation of time averaged gas fractions over a quarter of the measurement cross-section from a \*.epsxy file.

t	eps(t)
s	%
0.00040	34.80
0.00080	34.93
0.00120	35.07
0.00160	35.26
0.00200	35.42
0.00240	35.56
.....	

Table 2: Part of a \*.epst file.

r	eps(r)
mm	%
0.6	12.173
1.8	12.173
3.1	12.313
4.3	12.612
5.5	12.748
6.7	13.074
.....	

Table 3: Part of a \*.epsrad\_80 file.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.905
0	0	0	0	0	0	0	0	0	0	0	0	0.972	0.972	
0	0	0	0	0	0	0	0	0	0	0	0.517	0.528	0.820	
0	0	0	0	0	0	0	0	0	0	0.705	0.784	1.221	0.755	
0	0	0	0	0	0	0	0	0	0.882	0.665	0.814	1.129	0.778	
0	0	0	0	0	0	0	0.938	0.827	0.750	0.795	1.167	0.963		
0	0	0	0	0	0	0.669	1.235	1.071	1.010	0.808	1.029	0.820		
0	0	0	0	0	0	0.847	1.019	1.117	0.897	0.854	0.972	1.050		
0	0	0	0	0	1.500	1.129	0.991	0.963	0.955	1.029	1.129	1.141		
0	0	0	0	0.868	1.019	1.479	0.719	1.082	1.029	0.991	1.346	1.105		
0	0	0	0	0.766	0.972	0.905	0.861	0.669	1.180	1.193	0.808	1.071		
0	0	0	0.565	0.745	1.019	0.905	0.536	1.419	1.382	1.382	1.250	1.296		
0	0	0	0.571	1.071	1.221	0.461	0.477	0.861	1.117	1.207	1.029	1.141		
0	0	0.963	0.597	0.739	1.280	1.094	0.991	1.029	1.094	1.061	1.071	1.117		
0	0	0.946	1.019	0.981	0.972	1.250	1.154	0.991	1.167	1.071	1.082	0.882		
0	0	0.991	0.981	0.854	0.991	1.000	0.905	1.019	0.861	0.921	1.061	1.040		
0	1.346	0.745	0.972	0.875	0.905	0.929	0.868	0.972	0.897	1.207	0.890	0.963		
0	0.861	0.471	1.019	0.972	1.029	0.905	0.882	0.854	0.847	1.193	0.946	0.938		
0	0.897	0.938	0.972	0.875	1.000	1.061	1.000	1.117	1.296	1.207	0.946	0.827		
0.000	0.991	1.050	1.040	0.946	0.921	0.827	0.929	1.419	1.313	1.221	1.193	0.789		
0.525	1.615	1.050	0.882	0.905	0.882	0.802	1.382	1.400	1.193	1.207	0.802	1.419		
0.533	0.525	1.019	0.929	0.847	0.913	0.827	0.820	1.296	1.094	1.250	1.250	1.382		
0.541	0.607	0.734	0.729	0.808	0.761	0.854	0.827	0.750	0.677	0.691	0.686	1.296		
1.479	0.719	0.814	0.625	0.827	1.667	0.772	1.094	0.714	1.082	1.000	0.868	1.458		
0.682	1.438	0.854	1.000	0.833	0.827	1.180	0.686	0.691	1.141	1.105	1.117	1.329		
1.193	0.745	1.382	1.061	0.778	0.574	0.745	0.724	0.847	0.921	1.591	1.129	0.897		
...														

Table 4: Example of the calibration values for a part of the measurement cross-section from a \*.uw file.

0.6	2076.00	0.6	2.581
1.8	2076.00	1.8	2.581
3.1	2081.17	3.1	2.581
4.3	2090.73	4.3	2.581
5.5	2088.69	5.5	2.581

6.7	2083.10
7.9	2087.58
9.2	2095.03
10.4	2107.30
11.6	2113.51
12.8	2111.34
14.0	2108.42
15.3	2107.56

.....

Table 5: Examples for azimuthally averaged calibration values (\*.uwrad\_80).

6.7	2.647
7.9	2.647
9.2	2.647
10.4	2.581
11.6	2.581
12.8	2.581
14.0	2.647
15.3	2.647

.....

Table 6: Examples for azimuthally averaged gas velocities (\*.vel).

## 5.5 CapVoid Module

The *CapVoid* module is used to calculate the mixing ratios of a two-phase flow for capacitive measurements. The *CapVoid* module supports only the parallel model.

The module parameters are described in the following table:

Parameter	Description
fs	Input file name.
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
pl	Low permittivity value. E.g. air has a permittivity of around 1.
ph	High permittivity value. E.g. water at 20°C has a permittivity of around 80.
fl	Low permittivity file.
fh	High permittivity file.
mf	Measure frequency. Default is 2500 Hz.
rp	Radial profile output, use only for radial symmetric flow situations!
sp	Output file path.

This module outputs several files listed in the table below:

File	Type	Description
*.epst	ASCII (table)	Mixing ratio over the whole measurement time for each frame.
*.epsxy	ASCII (single matrix)	Averaged mixing ratio over the whole time.
*.epsrad	ASCII (table)	Radial mixing ratios as defined in the *.grd file.
*.v	Binary (8-bit unsigned integer)	The mixing ratio file calculated by the module. Values 0-100. 255 means invalid value.

## 5.6 CapVoid2 Module

The *CapVoid* & *CapVoid2* modules are used to calculate the mixing ratios of a two-phase flow for capacitive measurements. The new *CapVoid2* module let the user decide which mixing model he want to use (specified by the *pm* parameter). In contrast, the classical *CapVoid* module supports only the parallel model.

The module parameters are described in the following table:

Parameter	Description
-----------	-------------

fs	Input file name.
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
pm	Permittivity model to use. <i>This parameter is only available in the CapVoid2 module!</i> The parameter must be of one of the following values: <ul style="list-style-type: none"> <li>• <i>parallel</i></li> <li>• <i>series</i></li> <li>• <i>logarithmic</i></li> <li>• <i>maxwell-garnett-epsilon-low-is-dispersed</i></li> <li>• <i>maxwell-garnett-epsilon-low-is-continuous</i></li> </ul>
p	Write permittivity output files. Can be "y" or "n".
pl	Low permittivity value. E.g. air has a permittivity of around 1.
ph	High permittivity value. E.g. water at 20°C has a permittivity of around 80.
fl	Low permittivity file.
fh	High permittivity file.
rp	Radial profile output, use only for radial symmetric flow situations!
vf	Use unclamped floating-point values. The calculated alpha value of the mixture model is not clamped between 0 and 1. This option also writes floating-point values to a *.fv file. Holdup values can therefore be > 100% and < 0.0%.
th	Threshold value. Any hold-up >= this threshold is set to 100%.
sp	Output file path.

This module outputs several files listed in the table below:

File	Type	Description
*.epst	ASCII (table)	Mixing ratio over the whole measurement time for each frame.
*.epsxy	ASCII (single matrix)	Averaged mixing ratio over the whole time.
*.epsrad	ASCII (table)	Radial mixing ratios as defined in the *.grd file.
*.v of *.fv	Binary (8-bit unsigned integer or 32-bit floating point value)	For 8-bit values: The mixing ratio file calculated by the module. Values 0-100. 255 means invalid value. For 32-bit floating point outputs: NaN means invalid/undefined value.
*.pxy	ASCII (table)	Averaged permittivity over the whole measurement time.
*.p	Binary (16-bit unsigned integer)	Contains the permittivity values calculated by the module. The values are scaled by 100. This means that a value of 4576 is a permittivity of 45.76.

## 5.7 Velocity Module, calculation of the averaged local gas velocities

For the calculation of local gas phase velocities, the module velocity.exe can be applied. It uses the *Fast Fourier Transformation* to make a fast *cross-correlation* between the data of a set of two sensors in flow direction. The module requires the following parameters:

Parameter	Description
fs	First sensor void file (*.v) (including path)
dc	Distance between measurement planes in mm
fc	Second sensor void file (*.v) (including path)
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
sp	Output path (optional)
mf	Measurement frequency (optional, if not set 2500 Hz is used as default)

After calculation the velocity module delivers the following two files:

File	Type	Description
*.vel	ASCII (Table)	Contains a table with two columns. The left shows the centre radii of the ring-shaped domains in mm and the right presents the associated azimuthally averaged local gas velocities in m/s (Table 6),
*.velxy	ASCII (Matrix)	Matrix in sensor dimension $x, y$ , containing the calculated time averaged velocity value of each individual mesh point ( $x, y$ ) in m/s. The quality of the local velocities may be insufficient due to the bad statistics. Therefore, it is recommended to check these files very carefully before using for scientific data evaluation. Points outside of the pipe cross-section are marked by 0 whereas velocities with an absolute value of zero are denoted by 0.000 (Table 8).

## 5.8 BubIdent Module, identification and labelling of single bubbles

The *BubIdent* module is used to identify and separate single bubbles from the three dimensional void fraction files (\*.v). The method of bubble identification has been described in detail in chapter 3.4. The module requires the following parameters:

Parameter	Description
fs	Source void file for bubble identification (*.v) (including path)
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
dl	Threshold level for bubble search algorithm (optional 0..30, default 10)

Changing the threshold level (dl) makes significant changes to bubble sizes and is only recommended for experienced users! Standard value of 10% gives reliable results for water gas two phase flows!

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.921
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.905	1.141
0	0	0	0	0	0	0	0	0	0	0	0	0.972	0.972	0.600	
0	0	0	0	0	0	0	0	0	0	0	0	0.517	0.528	0.820	0.660
0	0	0	0	0	0	0	0	0	0	0.705	0.784	1.221	0.755	0.761	
0	0	0	0	0	0	0	0	0	0.882	0.665	0.814	1.129	0.778	0.772	
0	0	0	0	0	0	0	0.938	0.827	0.750	0.795	1.167	0.963	1.180		
0	0	0	0	0	0	0.669	1.235	1.071	1.010	0.808	1.029	0.820	0.913		
0	0	0	0	0	0	0.847	1.019	1.117	0.897	0.854	0.972	1.050	1.221		
0	0	0	0	0	1.500	1.129	0.991	0.963	0.955	1.029	1.129	1.141	1.313		
0	0	0	0	0.868	1.019	1.479	0.719	1.082	1.029	0.991	1.346	1.105	0.972		
0	0	0	0	0.766	0.972	0.905	0.861	0.669	1.180	1.193	0.808	1.071	1.061		
0	0	0	0.565	0.745	1.019	0.905	0.536	1.419	1.382	1.382	1.250	1.296	1.207		

0	0	0	0.571	1.071	1.221	0.461	0.477	0.861	1.117	1.207	1.029	1.141	1.235
0	0	0.963	0.597	0.739	1.280	1.094	0.991	1.029	1.094	1.061	1.071	1.117	1.154
0	0	0.946	1.019	0.981	0.972	1.250	1.154	0.991	1.167	1.071	1.082	0.882	0.938
0	0	0.991	0.981	0.854	0.991	1.000	0.905	1.019	0.861	0.921	1.061	1.040	1.180
0	1.346	0.745	0.972	0.875	0.905	0.929	0.868	0.972	0.897	1.207	0.890	0.963	1.154
0	0.861	0.471	1.019	0.972	1.029	0.905	0.882	0.854	0.847	1.193	0.946	0.938	1.094
0	0.897	0.938	0.972	0.875	1.000	1.061	1.000	1.117	1.296	1.207	0.946	0.827	1.265
0.000	0.991	1.050	1.040	0.946	0.921	0.827	0.929	1.419	1.313	1.221	1.193	0.789	1.280
0.525	1.615	1.050	0.882	0.905	0.882	0.802	1.382	1.400	1.193	1.207	0.802	1.419	1.280
0.533	0.525	1.019	0.929	0.847	0.913	0.827	0.820	1.296	1.094	1.250	1.250	1.382	1.329
0.541	0.607	0.734	0.729	0.808	0.761	0.854	0.827	0.750	0.677	0.691	0.686	1.296	1.154
1.479	0.719	0.814	0.625	0.827	1.667	0.772	1.094	0.714	1.082	1.000	0.868	1.458	1.313
0.682	1.438	0.854	1.000	0.833	0.827	1.180	0.686	0.691	1.141	1.105	1.117	1.329	1.167
1.193	0.745	1.382	1.061	0.778	0.574	0.745	0.724	0.847	0.921	1.591	1.129	0.897	0.882
...													

Table 7: Example of the gas velocity values for a part of the measurement cross-section from a \*.velxy file.

As an output file the module generates a binary file of the same dimension as the void file containing biunique identification numbers (32-bit signed integer) for all voxels belonging to the same bubbles:

File	Type	Description
*.b	Binary (32-bit unsigned integer)	Three dimensional matrix of bubble identifiers.

Table 8: Example of the gas velocity values for a part of the measurement cross-section from a \*.velxy file.

## 5.9 BubProp Module, bubble property analysis

The next step in the data evaluation is the definition of bubble properties. For this, it is necessary to finish all aforementioned operations, i.e. to have the \*.v and \*.b files. The *BubProp* module needs the \*.v file as well as the \*.b file for calculation. Only one of these has to be specified. The module expects the related file with the same name in the same folder, if one of the files is missed an error message is generated in the \*.log-file. Additionally, the geometry files have to be specified (fg). The algorithms for property determination and calculation are described in detail in chapter 3.5. The module requires the following parameters:

Parameter	Description
fs	Name and path of source files for bubble properties (*.v, *.b)
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
mf	Measurement frequency (optional, if not set 2500 Hz is used as default)

The program delivers the results in form of a table which contains one row with the properties for each bubble after the head.

File	Type	Description
------	------	-------------



<b>*.a</b>	ASCII (Table)	Table with bubble properties for each single detected bubble with the bubble number "bb" with the following content (each parameter describes is saved in one column of the *.a-file)
------------	---------------	---

The data are stored in an ASCII file (\*.a) containing this table (Table 9 exemplifies the beginning of a \*.a file):

bb	im	jm	km	Ifront	jfront	kfront	iback	jback	kback	rmi	rmj	rmk	rmxy	max	v	rv	n	deps	rxymax
[-]	[ms]	[mm]	[mm]	[ms]	[mm]	[mm]	[ms]	[mm]	[mm]	[ms]	[mm]	[mm]	[mm]	[%]	[ms*mm <sup>2</sup> ]	[s3(ms*mm <sup>2</sup> )]	[-]	[%]	[mm]
1	3.6	18.7	132.2	0.4	15.0	120.0	9.6	21.0	135.0	5.4	13.1	13.0	18.4	100	1348.93	6.85	888	0.0004527	9.031
2	18.0	098.2	103.1	0.4	21.0	102.0	52.0	36.0	135.0	23.9	97.0	72.5	121.1	100	304569.44	41.74	135812	0.1022072	55.256
3	6.0	165.9	44.1	0.4	147.0	51.0	15.6	177.0	48.0	9.2	15.6	13.4	20.6	100	3991.92	9.84	2134	0.0013396	12.017
4	8.5	27.5	66.7	0.4	18.0	66.0	16.0	42.0	72.0	8.7	16.3	12.9	20.8	100	3754.53	9.64	1964	0.0012599	10.822
5	14.993	6	25.1	9.2	93.0	24.0	19.2	99.0	24.0	5.7	10.2	8.4	13.2	100	1180.37	6.56	667	0.0003961	7.183
6																			

Table 9: Part of the table with bubble properties in a \*.a-file.

The abbreviation in the table head stands for:

Parameter	Unit	Description
<b>bb</b>	-	Bubble identification number,
<b>im</b>	ms	Coordinates of the centre of the bubble in i – flow direction and j, k – measurement cross-section,
<b>jm</b>	mm	
<b>km</b>	mm	
<b>ifront</b>	ms	Coordinates of the bubble front,
<b>jfront</b>	mm	
<b>kfront</b>	mm	
<b>iback</b>	ms	Coordinates of the bubble back,
<b>jback</b>	mm	
<b>kback</b>	mm	
<b>rmi</b>	ms	Moments of the bubble in i – flow direction and j, k – measurement cross-section,
<b>rmj</b>	mm	
<b>rmk</b>	mm	
<b>rmxy</b>	mm	Radial moment of the bubble in measurement cross-section plane,
<b>max</b>	%	Maximum void fraction of the bubble,
<b>v</b>	ms*mm <sup>2</sup>	Bubble volume,
<b>rv</b>	f(ms*mm <sup>2</sup> )	Radius of a volume equivalent sphere,
<b>n</b>	-	Number of volume elements occupied by the bubble,
<b>deps</b>	%	Part of the void fraction per bubble referring to the total flow volume,
<b>rxymax</b>	mm	Maximal circle equivalent radius of the bubble in the measurement plane.

Table 10: Description of the single bubble parameters in the \*.a-file.

In the second line of Table 9 the dimension units for the bubble properties are given. It is indicated once again that with the bubble characteristics contrary to the gas fraction calculation, the index *i* refers to the serial number of the frames (time axis or *z*-direction) and the indices *j* and *k* apply to the measurement cross-section, respectively. All characteristics in direction of the time axis are given in *ms*. The conversion of the time dimension in a geometrical dimension (*mm*) is possible using

the velocity information defined with the velocity module (see 5.5) from the datasets of two wire-mesh sensors series installed.

## 5.10 BubSizeDis Module, Calculation of Bubble size distributions

If it is necessary to analyse bubble size distributions, this can be done with the unit `bubsize.exe`. It requires the `*.v`, `*.b` and `*.a` files. One of these files has to be specified as first parameter (`fs`). The module expects the related files with the same name in the same folder, if one of the files is missing an error message is generated in the `*.log`-file. Additionally, the geometry files have to be specified (`fg`). Optionally, a velocity file (`*.vel`) can be specified as additional parameter (`fc`). In this case, the required geometrical data in the bubble property table (`*.a` file) is converted into real 3D information, e. g. the volume equivalent diameter of the bubble is converted from  $\sqrt[3]{ms * mm^2}$  into `mm`. If there is no `*.vel`-file specified, the superficial gas velocity (`vg`) can be entered in `m/s`. The overall measurement time is calculated from measurement frequency (`mf`;) which can be defined by the user (default 2500 Hz) and the total frame number. Furthermore, the inner diameter of the sensor and the number of ring-shaped domains is determined from the geometry files.

After finishing calculations, the program returns a `*.his_lin` and a `*.his_log` file, which contain the linear and the logarithmic bubble size distributions. Both files are *ASCII* files. exemplifies a logarithmic bubble size distribution.

d	hdxy	hdrelxy	hdnxy	hdv	hdrelv	hdnv
[mm]	[%/mm]	[1/mm]	[1/mm/s]	[%/mm]	[1/mm]	[1/mm/s]
...						
2.50	0.0696	0.3416	2882.00	0.0288	0.1412	1971.00
2.60	0.0664	0.3261	2503.00	0.0382	0.1875	2331.00
2.70	0.0848	0.4165	3009.00	0.0461	0.2264	2549.00
2.80	0.0940	0.4612	3055.00	0.0545	0.2674	2716.00
2.90	0.1056	0.5183	3241.00	0.0627	0.3077	2825.00
3.00	0.1212	0.5949	3533.62	0.0730	0.3586	2951.55
3.11	0.1288	0.6324	3376.85	0.0849	0.4166	3103.95
3.22	0.1276	0.6265	2992.83	0.0976	0.4789	3203.98
3.33	0.1402	0.6880	2968.25	0.1098	0.5388	3234.49
3.45	0.1568	0.7696	3042.70	0.1197	0.5876	3194.02
3.57	0.1563	0.7674	2786.43	0.1380	0.6773	3306.48
3.70	0.1726	0.8473	2829.40	0.1506	0.7395	3257.98
3.83	0.1688	0.8285	2546.78	0.1624	0.7972	3157.65
3.97	0.1919	0.9422	2660.76	0.1723	0.8459	3037.73
4.11	0.2001	0.9824	2567.87	0.1905	0.9349	3022.71

Table 11: Detail of a bubble size distribution for logarithmic bubble class widths.

The single columns in Table 11 include the following information:

Parameter	Unit
d	mm
hdxy	%/mm
hdrelxy	1/mm
hdnxy	1/mm/s
hdv	%/mm
hdrelv	1/mm
hdnv	1/mm/s

- ***d***: Bubble diameter in mm.
- ***hd***: Gas fraction of the bubbles of this class referring to bubble class width in %/mm
- ***hdrel***: *hd* related to the total gas fraction in 1/mm
- ***hdn***: Number of bubbles in the respective class referring to the class width and the total

In both files these three columns (distributions) are available related to the area equivalent bubble diameter for the largest cross-section area of the bubble in the measurement plane (*xy*) and basing on the volume equivalent bubble diameter (*v*), respectively. In the first column of Table 11 shows the arrangement of the bubble classes. Independent of the logarithmic distributions, for bubble diameters less than 3 mm a linear width of 0.1 mm is used. Only on bubble diameters larger than 3 mm the logarithmic increase of the bubble class width takes effect.

The linear distributions contained in the *\*.his\_lin* files have the same structure apart from the difference of linear increase of the bubble class width (normally HZDR use 0.25 mm) over the complete interval of bubble classes.

## 5.11 MixCalib Module, calibrate mixing measurements

The *MixCalib* module calculates the calibration matrices *m* and *n* from a set of given calibration files and their corresponding conductivity values. See chapter 3.7 for more detailed explanation what this module does.

The module parameters are described in the following table:

Parameter	Description
<b>fs</b>	First input file
<b>c0</b>	Conductivity of the first file in $\mu\text{S/m}$ .
<b>fg</b>	Name and path of a geometry file (*.geo, *.grd, *.gpl)
<b>f1</b>	Second calibration file.
<b>c1</b>	Conductivity of the second calibration file.
<b>f2 ... f7</b>	3 <sup>rd</sup> , 4 <sup>th</sup> , 5 <sup>th</sup> , .. calibration files.
<b>c2 ... c7</b>	3 <sup>rd</sup> , 4 <sup>th</sup> , 5 <sup>th</sup> , .. conductivity values in $\mu\text{S/m}$ .
<b>sp</b>	Output path to use.
<b>o</b>	Output file name to use.

Output files:

File	Type	Description
<b>*.mixm</b>	ASCII (Matrix)	The <i>m</i> matrix of the calibration.
<b>*.mixn</b>	ASCII (Matrix)	The <i>n</i> matrix of the calibration.
<b>*.mixcd</b>	ASCII (Matrix)	Contains the coefficient of determination for each crossing point of the calibration.

<b>*.mixtab</b>	ASCII (Table)	Contains a graph for each measured ADC value and its conductivity.
-----------------	------------------	--

## 5.12 MixCond Module, calculation the conductivity of a mixture

The *MixCond* module uses the matrices *m* and *n* from the *MixCalib* module to calculate the conductivity of a measurement. You can find a detailed explanation of this module in chapter 3.7.

The module parameters are described in the following table:

Parameter	Description
<b>fs</b>	Input file
<b>fg</b>	Name and path of a geometry file (*.geo, *.grd, *.gpl)
<b>fm</b>	The <i>m</i> matrix file.
<b>fn</b>	The <i>n</i> matrix file.
<b>sp</b>	Output path to use.
<b>o</b>	Output file name to use.

Output files:

File	Type	Description
<b>*.cond</b>	Binary (16-bit unsigned integer)	The conductivity in $\mu\text{S}/\text{m}$ .

## 5.13 MixRatio Module, calculating mixing ratios

The *MixRatio* module calculates mixing ratios from an input file, a base reference file and a tracer fluid conductivity to determine the mixing the matrices *m* and *n* from the *MixCalib* module to calculate the conductivity of a measurement. See chapter 3.7 for more in depth information.

The module parameters are described in the following table:

Parameter	Description
fs	Input file
fg	Name and path of a geometry file (*.geo, *.grd, *.gpl)
fb	Base reference conductivity file (*.cond).
cb	Base reference conductivity value in $\mu\text{S/m}$ if fb is not set.
ft	Tracer conductivity file (*.cond).
ct	Tracer conductivity value in $\mu\text{S/m}$ if ft is not set.
am	Averaging method to use. This option is only used for the base and the tracer file. Can be either: average_frames_and_crossing_points: Averages all frames and crossing points to a single value used for mixing ratio calculation. or average_frames: Averages all frames but not the crossing points to calculate the mixing ratio calculation.
sp	Output path to use.
o	Output file name to use.

Output files:

File	Type	Description
*.mrat	Binary (8-bit unsigned integer)	The calculated mixing ratios in percent (0-100%). 255 means invalid value.

## 5.14 Transform module, applying various transform operations

The *Transform* module is used for applying various transformations. It supports rotate, mirror and transpose operations. Furthermore, an interpolation method can be used for the rotate operation. You can choose between *nearest* and *bilinear* interpolation. The selected angle of rotation is always *counter-clockwise* and defined in degrees between  $0^\circ$  and  $360^\circ$ .

The module parameters are described in the following table:

Parameter	Description
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
s	Start frame is the first frame to process.
e	End frame is the last frame to process.
op	Transform operation. The input data can be transposes, mirrors or rotated.
m	Interpolation method. Can be <i>nearest</i> or <i>bilinear</i> .
mf	Measurement frequency (optional, if not set 2500 Hz is used as default)

Output files:

File	Type	Description
*.dat	Binary (16-bit)	Transformed data file.

	unsigned integer)	
--	-------------------	--

**Warning:** Be careful with this module! Especially the rotate option is a destructive operation. It's recommend to use this module only for visualization purposes and not for any further data processing.

### 5.15 Trim Module, trimming the size of files

This module was developed to trim wrongly extracted data files. It extracts a rectangular subsection from the input data files and writes it to a newly generated output file. A start and an end frame can also be chosen.

The module parameters are described in the following table:

Parameter	Description
s	Start frame is the first frame to process.
e	End frame is the last frame to process.
x1	Start x is the x position of the upper left trim coordinate.
y1	Start y is the y position of the upper left trim coordinate.
x2	End x is the x position of the lower right end coordinate.
y2	End y is the y position of the lower right trim coordinate.
sp	Output directory (optional)
o	Output file name

Output files:

File	Type	Description
*.dat	Binary (16-bit unsigned integer)	Trimmed data file.

**Warning:** Only use this module as a last resort! It was mainly written for data forensics. E.g. when you have a set of wrongly extracted data files and the original measurement files were lost or hard to reclaim. Always use the original measurement files to regenerate your data files using the FileConverter module.

### 5.16 FixUp Module, fixing faulty pixels

The *FixUp* module is another module that can be used for data forensics. It fixes single pixels as well as rows and columns of faulty pixels. The information about faulty pixels can be supplied by an error mask file or directly by specifying the module parameters.

The error mask file is a simple *ASCII* text file very similar to the content of the geometry files. The file contains a matrix with only 3 possible values:

- A 0 means that the pixel is not part of the measuring area and is therefore ignored by the fixing algorithm.

- A value of 2 means that the pixel is inside the measuring area and a non-faulty pixel.
- A value of 3 indicates that the pixel is faulty and can be replaced by the fixing algorithm.

A typical error mask for an 8 x 8 sensor could look like that:

```
0 0 0 1 1 0 0 0
0 0 1 1 1 2 0 0
0 2 2 2 2 2 2 0
1 1 1 1 1 2 1 1
1 1 1 1 1 2 1 1
0 1 1 1 1 2 1 0
0 0 1 1 1 2 0 0
0 0 0 1 1 0 0 0
```

Where column 5 (start counting from 0) and row 2 were fixed by the module. If you don't have an error matrix file, you can manually enter the rows, columns and single pixels using the *hi*, *vi* and *pi* parameters.

A complete list of module parameters is described in the following table:

Parameter	Description
<i>fg</i>	Name and path of geometry files (*.geo, *.grd, *.gpl)
<i>s</i>	Start frame is the first frame to process. If nothing was entered the whole file is processed.
<i>e</i>	End frame is the last frame to process. If nothing was entered the whole file is processed.
<i>ee</i>	The error mask file as described above.
<i>hi</i>	Fix horizontal pixel data is used to fix rows of faulty pixels. Rows are separated with semicolons. The value is a zero-based index. If you want to fix horizontal line 4 and 8, the <i>hi</i> parameter is “4;8”
<i>vi</i>	Fix vertical pixel data is used to fix columns of faulty pixels. Columns are separated with semicolons. The value is a zero-based index. If you want to fix vertical line 7, 9 and 11, the <i>vi</i> parameter is “7;9;11”
<i>pi</i>	Fix single pixels. Single pixels are separated with semicolons. Coordinates are separated with commas. If you want to fix a pixel at position 4,5 and another one at position 6,7, the <i>pi</i> parameter is “4,5;6,7”
<i>m</i>	Interpolation method. Can be <i>median</i> or <i>mean</i> .
<i>fs</i>	Output directory to use.
<i>o</i>	Output file name to use.

Output files:

File	Type	Description
*.dat	Binary (16-bit unsigned integer)	The fixed data file.

**Warning:** Only use this module as a last resort! It was mainly written for data forensics. E.g. when you have a set of wrongly extracted data files and the original measurement files were lost or hard to reclaim. Always use the original measurement files to regenerate your data files using the *FileConverter* module.

## 5.17 NRemoval Module

The Noise Removal module interpolates/fixes faulty or noisy pixels using a noise reference file. It's very similar to the FixUp module which uses an error mask (can be parameter defined or a selected mask file) for all frames.

The parameters used by the module are described in the following table:

Parameter	Description
fg	Geometry file
fe	Noise file (can be a *.dat, *.cdat or *.v file)
m	Method to interpolate neighbours (can be 'all_26', 'local_8' or 'time_2')
th	Threshold. Every value in the Noise file $\geq$ threshold will be fixed.
s	Start frame is the first frame to process.
e	End frame is the last frame to process.
sp	The output directory (optional).
o	The output file name to be used.

The module currently supports 3 different interpolation methods. These methods are description in the following table:

Method	Description
all_26	All 26 neighbours of a single pixel including previous and next frame.
local_8	Only the 8 neighbour pixels of the current frame. No interpolation in the time domain is done.
time_2	No local interpolation. Only interpolate the next and the previous pixel in the time domain.

The output files of the module:

File	Type	Description
*.dat, *.cdat or *.v	Binary (16-bit unsigned integer or 8-bit value)	Filtered output file.

## 5.18 Reduce Module, reduces multiple frames to a single frame

This module reduces \*.dat, \*.cdat and void files. It writes the median or mean for each cross section within a chosen window size to an output file.

The module parameters are described in the following table:

Parameter	Description
s	Start frame is the first frame to process.
e	End frame is the last frame to process.
op	Filter operation to be used. Can be "mean" or "median".



<b>f</b>	Window size in frames.
<b>sp</b>	The output directory (optional).
<b>o</b>	The output file name to be used.

Output files:

File	Type	Description
*.dat, *.cdat	Binary (16-bit unsigned integer)	Reduced data file.
*.v	Binary (8-bit unsigned integer)	Reduced void file.

## 5.19 ImageMaker module

The *Imagemaker* module has been designed to export frames from uncalibrated sensor measurement files (\*.dat) as well as calibrated void fraction data (\*.v) as bitmap series. The user can specify the files to be exported (fs:) and the output path (sp:). The start and end frame for the bitmap export (f1:) and (f2:) and an integer zoom factor (z:). If a \*.dati-file with the same name as the \*.v or \*.dat file exists in the same folder the sensor dimension is extracted from this file, otherwise the user must specify the width and height of the sensor with the parameters (j1:) and (k1:) respectively. The file names of the created bitmaps consist of the original file name the frame number in the original data set and the extension (\*.bmp) optional the original file name can be changed also into a user defined string (o:).

## 5.20 OptFlow module

The optical flow module calculates axial and tangential velocities from \*.dat files and saves them in several output files.

The complete list of module parameters is described in the following table:

Parameter	Description
<b>fs</b>	Input file name
<b>fg</b>	Name and path of geometry files (*.geo, *.grd, *.gpl)
<b>mf</b>	Measurement frequency. Default is 2500 Hz.
<b>fc</b>	Calibration file. If not set parameter fu is used.
<b>fu</b>	UW-file
<b>al</b>	Optical flow alpha value. Default is 0.01
<b>oi</b>	Number of optical flow iterations.
<b>fl</b>	Filter type: none, max or ac.
<b>ai</b>	Iterations for ac filter.
<b>sp</b>	Output directory.
<b>o</b>	Output file name

The optical flow module generates several files containing the calculated velocities of the input data. These files are:

File	Type	Description
*.va	Binary (signed 16 bit integer)	Absolute velocity calculated from vx and vy.
*.vx	Binary (signed 16 bit integer)	Velocity in x direction.
*.vy	Binary (signed 16 bit integer)	Velocity in y direction.
*.vt	Binary (signed 16 bit integer)	Tangential velocity
*.vr	Binary (signed 16 bit integer)	Radial velocity
*.vei	ASCII (Ini)	Info file for velocity files. Contains the size and number of frames.

## 5.21 Histogram Module

The *Histogram* module was designed to be used in conjunction with the *OptFlow* module. This module may be extended to support more file types in the future.

The complete list of parameters is described in the following table:

Parameter	Description
fs	Input file name. Currently only the output files of the OptFlow module are accepted (*.va, *.vx, *.vy, *.vr, *.vt).
fg	Name and path of geometry files (*.geo, *.grd, *.gpl)
s	Start frame
e	End frame
bi	Minimum bin interval.
ba	Maximum bin interval.
b	Number of bins within this interval.
fl	Filter type: none, max or ac.
sp	Output directory.
o	Output file name.

The optical flow module generates a files containing the calculated histogram of the input data:

File	Type	Description
*.hist	ASCII (tabular file)	Histogram generated from the input data.



## 6 Editors of the Wire Mesh Sensor Framework

The framework comes with

### 6.1 SEI Editor

This editor allows the user to create SEI-files (sensor extraction information files). The SEI-file specifies the location of one or more sensors inside the measurement domain matrix.

You can start the SEI-Editor by clicking *Tools* → *SEI Editor*. This will run the editor as a standalone tool. No files were added to the project tree! It's also possible to start the SEI-Editor by right clicking on a project folder or the main project node itself. This will add a newly generated \*.sei file to the project. Select *Create file(s)* → *SEI Editor* in the context menu.

The editor window consists of two panels. The left contains the definition of the measurement domain and a list of all sensor definitions inside the matrix. The right panel contains a graphical representation of the whole measurement matrix.

The info field in the left panel is a user defined string to identify the experiment/sensor arrangement. The matrix width and height is the dimension of the measured \*.mes file. This is the width and height of the whole sensor matrix.

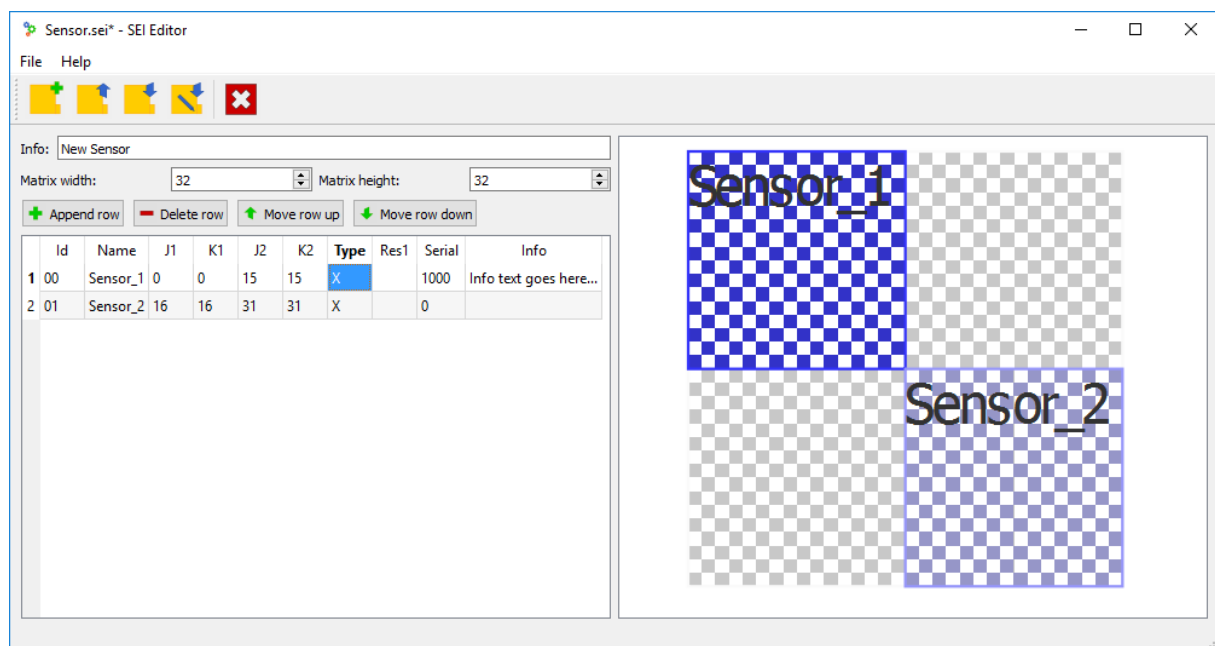


Table 12: The SEI Editor of the Wire Mesh Sensor Framework.

In the sensor table a user defined *ID* (12 characters), a sensor *Name* (added to the file name), the *Serial* number and a free *Info* text can be added and edited. The *Type* is used to specify the first and second sensor of a double sensor arrangement (e.g. velocity calculations). Typically, "X" for the first and "Y" for the second sensor is used respectively. The parameters *J1*, *K1*, *J2* and *K2* specify the start and end coordinates of

a single sensor within the whole measurement matrix. By clicking of one of the lines in the table, the highlighted sensor is shown in the right graphical visualization area.

The tool bar at the top of the window consists of the following buttons:

- *New*: Creates a new SEI-file. If the current one was modified, it asks to save the current SEI file.
- *Open*: Opens an existing SEI-file.
- *Save*: Saves the current SEI-File.
- *Save as...:* Saves the current SEI-file under a new file name.
- *Exit*: Closes the editor. The editor asks to save the current SEI-file if it was modified.

These buttons can also be found in the *File* menu at the top of the window.

## 6.2 Mask Editor

This editor allows the user to edit complex mask files for the wire mesh sensor. In contrast to the standard *Geometry* module it allows much more control over the generated geometry mask files.

You can start the Mask-Editor by clicking *Tools* → *Mask Editor*. This will run the editor as a standalone tool. Like in the SEI-Editor no files were added to the project tree! You can also start the editor by right clicking on a project folder or the root node itself. This will add a newly generated *\*.wmmask* file to the project.

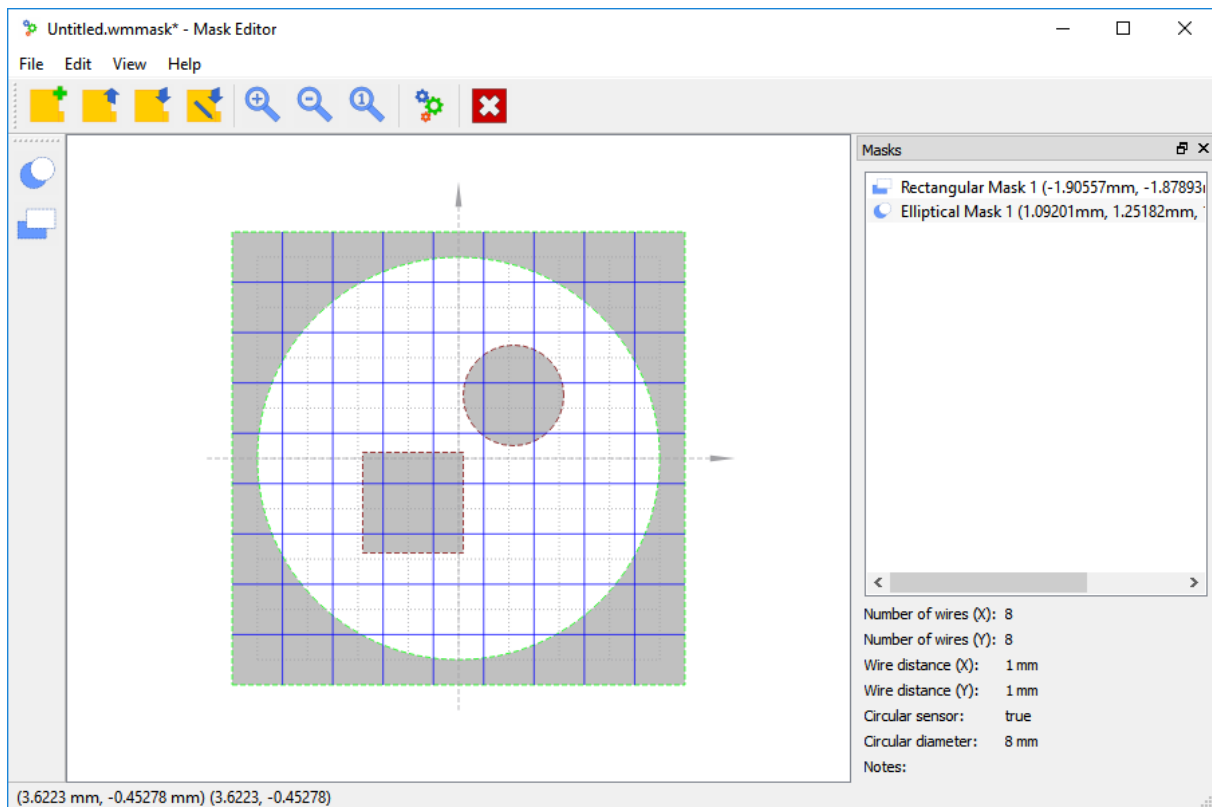


Table 13: Mask Editor of the Wire Mesh Sensor Framework.

The tool bar at the top of the window consists of the following buttons:

- *New*: Creates a new mask file. If the current one was modified, the editor asks you to save the file.
- *Open*: Opens an existing mask file. Asks when the current file was modified.
- *Save*: Saves the current mask file.
- *Save as...*: Saves the current mask file under a new file name.
- *Zoom in*: Zooms the current mask in.
- *Zoom out*: Zooms the current mask out.
- *Reset Zoom*: Resets the zoom level.
- *Configure*: Configures the current sensor layout. It's possible to set the number of wires in  $x$  and  $y$  direction and the distance between the wires.
- *Exit*: Closes the editor. The editor asks to save the current mask if the file was modified.

To create a new mask file, click on the *New* button in the toolbar or *File* → *New ...*. A layout dialog is shown (see

Figure 25). You can enter the number of wires in  $x$  and  $y$  direction. Also the distance between a pair for wires can be entered. The circular sensor check box lets you define whether the sensor is a *circular* or a *rectangular* one. Selecting the *circular* sensor will

show a circular mask in the sensor view. The note field can be used for short user defined notes.

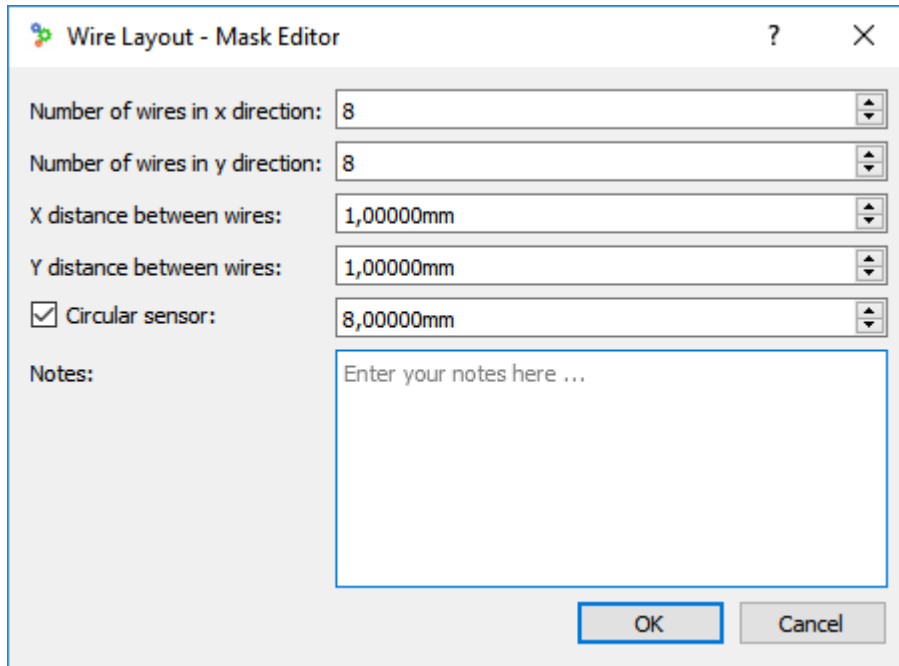


Figure 25: New layout dialog.

The tool bar at the left side of the window consists of a set of different masking primitives:

- *Elliptical mask*: Creates an elliptical mask.
- *Rectangular mask*: Creates a rectangular mask.

By clicking on them a dialog is shown. Here you can enter the initial parameters for a new mask primitive. You can select between millimetres and wire units (multiples of wire distances). Also the  $x$ ,  $y$  position and the size of the mask element can be entered. Subtractive masks are used to cut out regions from non-subtractive masks.

The docking window on the right contains a list of all mask primitives inside the measurement area. They can be edited by right clicking on them.

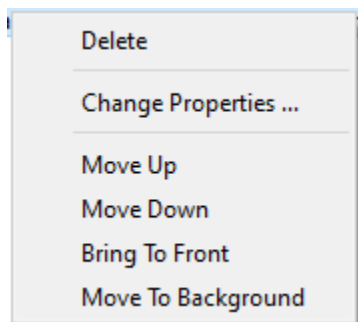


Figure 26: Context menu for a mask primitive.

The context menu exposed is shown in

Figure 26 and exposes the following menu entries:

- *Delete*: Deletes the mask element from the measurement area.
- *Change Properties...*: Changes the properties of the mask element.
- *Move Up*: Moves the element one level up in the Z direction.
- *Move Down*: Moves the element one level up in the Z direction.
- *Bring to Front*: Brings the element to the front. All other mask object are now behind this object.
- *Move to Background*: Moves the element to the background. All other mask elements are now behind this one.

The same menu is exposed when right clicking on the object in the mask view.



## 7 Internal Data Formats

This chapter describes the internal data formats used by the framework. These are configuration file formats like Module information files, Editor information files, Template configuration files and other files the framework is able to read and show. See the individual chapters for more information about the used formats.

### 7.1 Module Information Files

Module information files are used to describe the command line interface of an external module. They are automatically read at the start-up of the framework. The detailed structure is described in the following sections.

#### General Structure

MINF-files are ordinary INI-files with some extensions. They are divided in several sections and the general syntax/structure looks like that:

```
[Section 1]
Key1=Value1
Key2=Value2

[Section 2]
Key1=Value3
Key2=...

[Section 3]
Key1=Value4
Key2=...
...
```

The individual sections of MINF-file are described below.

#### Module Section

Contains general information about a module. The keys in the Module section are described in the following table:

Key	Description
<b>Name</b>	Name of the module.
<b>Info</b>	Short information text of the module.
<b>Icon</b>	Icon of the module. Currently not in use.
<b>Executable</b>	The name of the executable to start. Note: There is no need to add an *.exe suffix to the file name. It's automatically added.
<b>HelpFile</b>	Additional help file of the module.

A typical Module section looks like that:

```
[Module]
Name=Transform Module
Info=Applies various transformations on *.dat files.
Icon=
Executable=transform
HelpFile=
```

## Parameters Section

This section contains a list of all parameters in the module with a short name/description string. Usually this section looks like that:

```
[Parameters]
fs=Input file name
sp=Output path
pa=My module parameter
```

## ParameterTypes Section

In the new framework parameters are typed. This section is used to define a type for each individual parameter:

Type	Description
file	Just a simple file name without a path.
filedir	This is a full file path with directory and file name.
dir	Just a directory without a file name.
int	Integer. The framework represents them internally as 64 bit integers.
double	The parameter is a floating point parameter. The framework represents them as 64 bit floating point values (IEEE 754).
string	The parameter is a string.
enum	Defines an enumeration type. This is used to define a selection between different options. The framework uses a drop-down box in the module dialog for this type.

Typical application:

```
[ParameterTypes]
fs=filedir
fe=int
sp=dir
```

## ParameterEnabled

This section defines if a parameter is either in use or not by default. Its represented by an *boolean* value. Typically, this section looks like that:

```
[ParameterEnabled]
fs=true
fe=false
sp=true
```

## ParameterDefaults

Defines the parameters default values.

```
[ParameterDefaults]
f="myfile"
op=false
t=1.245
```

## ParameterConstraints

This section describes the constraint system for a module. Multiple constraints can be combined with a semicolon. An example usage is shown below:

```
[ParameterConstraint]
f=parameter_is_required();file_must_exist()
op=in_range(-20, 20);
t=any_of("do", "re", "mi");includes_parameter("ip")
ip=relates_to_parameter(GreaterEqual, "op");includes_parameter("t")
```

A list of possible constraints is listed in the table below. The nomenclature of the function signatures is described in chapter 7.5.

Constraint	Compatible types	Description
<code>parameter_is_required()</code>	all	The parameter is required. It's not possible to unselected it in the module dialog. Parameters: <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>file_must_exist()</code>	filedir	The selected file path must exist. Parameters: <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>file_must_by_of_type(&lt;ext1&gt;:string, [&lt;ext2&gt;:string,...])</code>	file, filedir	The file must have a certain file extension. Parameters: <ul style="list-style-type: none"> <li>• ext1: First file extension to check</li> <li>• ext2: Second file extensions to check</li> <li>• ...</li> </ul> Examples: <ul style="list-style-type: none"> <li>• <code>file_must_by_of_type("bmp", "png")</code> → File must be of type bmp or png.</li> </ul>
<code>dir_must_exist()</code>	dir	The selected directory must exist. Parameters: <ul style="list-style-type: none"> <li>• None</li> </ul>
<code>in_range(&lt;min&gt;:(int, double), &lt;max&gt;:(int, double))</code>	integer, double	The selected value must be in range min and max. Parameters can be of type double or integer. Parameters: <ul style="list-style-type: none"> <li>• min: Minimal value.</li> <li>• max: Maximal value.</li> <li>• ...</li> </ul> Examples: <ul style="list-style-type: none"> <li>• <code>in_range(0, 10)</code> → Accepts only values between 0 and 10.</li> </ul>
<code>any_of(&lt;e1&gt;:enum, [&lt;e2&gt;:enum, ...])</code>	enum	The enumeration must be of one of the passed values. Parameters: <ul style="list-style-type: none"> <li>• e1: First enumeration value</li> <li>• e2: Second enumeration value</li> <li>• ...</li> </ul> Examples: <ul style="list-style-type: none"> <li>• <code>any_of("linear", "bilinear")</code> → The enumeration must be either of linear or bilinear.</li> </ul>

<pre>strlen( &lt;min&gt;:(int), &lt;max&gt;:(int) )</pre>	string	<p>String must be at least min characters long but not longer than max characters.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• min: Minimum length of string.</li> <li>• max: Maximal length of string.</li> <li>• ...</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• strlen(4, 6) → Strings like “hello”, and “sensor” are accepted, but not “car” and “framework”.</li> </ul>
<pre>excludes_parameter( &lt;param1&gt;:string, [&lt;param2&gt;:string ...] )</pre>	all	<p>Excludes a list of given parameters if the actual parameter is selected.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• param1: First parameter to exclude.</li> <li>• param2: Second parameter to exclude.</li> <li>• ...</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• excludes_parameter(“a”, “b”) → This parameter cannot be selected with parameter a and b.</li> </ul>
<pre>includes_parameter( &lt;param1&gt;:string [&lt;param2&gt;:string, ...] )</pre>	all	<p>Includes a list of given parameters if the actual parameter is selected.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ext1: First file extension to check</li> <li>• ext2: Second file extensions to check</li> <li>• ...</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• includes_parameter(“a”, “b”) → This parameter must be selected with parameter a and b.</li> </ul>
<pre>relates_to_parameter( &lt;op&gt;:(Greater, Less, GreaterEqual, LessEqual), &lt;param&gt;:string )</pre>	integer, double	<p>Defines a relation between two parameters.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• op: Operation: op can be of Greater, Less, GreaterEqual or LessEqual.</li> <li>• param: Parameter name.</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• relates_to_parameter(GreaterEqual, “a”) → The actual parameter must be greater or equal than parameter a.</li> </ul>

## OutputFiles

This section defines the output files of module. The structure is very similar to the constraint section. Every key in this section contains a list of files to generate.

```
[OutputFiles]
<File1>=<expression>
<File2>=<expression>
```

```
<File3>=...
```

The names of the file keys are user defined. The expression placeholder contains a string generation expression. If the expression generates an empty string, no file will be populated. See Command Reference in chapter 7.5 for more information about string generation expression. All expressions returning a `string[]` or `string` can be used here.

## 7.2 Editor Information files

The framework does not only allow you to integrate command line style applications like modules. It also allows you to integrate graphical applications to edit certain files in your project (see chapter 5.21 for the build in editors). The format is described in this chapter.

Key	Description
Name	Name of the module.
Info	Short information text of the module.
Icon	Icon of the module. Currently not in use.
Executable	The name of the executable to start. Note: There is no need to add an *.exe suffix to the file name. It's automatically added.
HelpFile	Additional help file of the module.
InputFileTypes	A list of files supported by the editor.

A typical editor information file looks like that:

```
[Editor]
Name=Experimental Mask Editor
Info=Edits Mask Files (currently in experimental state)
Icon=
Executable=maskeditor
HelpFile=
InputFileTypes=wmmask
```

All editor information files are automatically read from the *editor* sub-directory during the start-up of the framework.

## 7.3 Template information files

The *Wire Mesh Sensor Framework* supports user defined project templates. A template can be used as basic skeleton for a new project. All needed files are saved in the *templates* sub-directory. Each template consists of two files: a configuration file and a script file.

- `<name>.tinf`
- `<name>.tscript`

The placeholder `<name>` can be any identifier. The template information file describes the input parameters and some general meta information about the template (name of the template, needed files, descriptions, ...). The script file on the other hand contains

a list of commands to execute after the user enters the parameters and finishes the wizard dialog.

### Template Section

Templates are also defined using the well-known INI syntax. The template sections typically look like that:

```
[Template]
Name=Empty project.
Info=An empty project template. Contains no files.
```

Key	Description
Name	Name of the templates. This is shown in the list box of the first wizard page.
Info	Short description text of the template. This text is shown below the template selection when you click on a template.
Script	Name of the script file without extension

### Parameters Section

This section contains a list of all parameters in the template with a short name/description. Usually this section looks like that:

```
[Parameters]
SeiFile=SEI file name
GeoFiles=Geometry files
MesFiles=MES files
ParameterDescriptions
[ParameterDescriptions]
SeiFile=A sensor information file
GeoFiles=A set of geometry files
MesFiles=One or more measurement files
ParameterTypes
[ParameterTypes]
SeiFile=FileOrEditorOutput
GeoFiles=FilesOrModuleOutput
MesFiles=MultiFilePaths
```

Type	Description
FileOrEditorOutput	The wizard asks for a file or an editor to create this file.
FilesOrModuleOutput	The wizard will ask for a set of files or a module to generate these files.
MultiFilePaths	Asks for one or more files.

### ParameterValues

Contains default values for a file parameter. Useful in combination with an editor.

```
[ParameterValues]
SeiFile=Sensor.sei
```

### ParameterExtensions

Comma separated file extensions. This section describes the supported file types for each file parameter.

```
[ParameterExtensions]
SeiFile=sei
```

```
GeoFiles=geo,grd,ggd,gpl
MesFiles=mes,ces
```

## ParameterModules and ParameterEditors

These two sections allow you to select the editor or module you want to use for a file parameter. The key is the parameter identifier and the key value is the module identifier you want to use. If you want to know more about integrating modules to the framework see section 7.1.

```
[ParameterModules]
GeoFiles=geo_adv

[ParameterEditors]
SeiFile=sei
```

## 7.4 Template Script Files

The template director contains another file called the template script file. It contains a list of commands to execute when a new project is generated.

A simple script file looks like that:

```
add_folders("", "SEI File");
add_folders("", "Geometry Files");
add_folders("", "Measurement Files");
add_folders("", "Batch Files");

add_files("SEI File", value("SeiFile"));
add_files("Geometry Files", value("GeoFiles"));
add_files("Measurement Files", value("MesFiles"));
```

Template Command	Description
<pre>add_folders(   &lt;dest_dir&gt;:(string, string[]),   &lt;new_dir&gt;:(string, string[]), )</pre>	<p>Adds a list of new directories to an existing directory.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>dest_dir</code>: Existing directory where to create a new directory. The root directory is an empty string "".</li> <li>• <code>new_dir</code>: Name of the new directory.</li> </ul>
<pre>add_files(   &lt;dir&gt;:string,   &lt;file&gt;:(string,string[]))</pre>	<p>Adds a list of files to a project directory.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>dir</code>: Existing directory where to add the files.</li> <li>• <code>file</code>: List of files to add.</li> </ul>
<pre>value(   &lt;identifier&gt;:string)</pre>	<p>The value of the parameter entered by the user in the wizard dialog.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• <code>identifier</code>: The identifier is the key in the parameters section of the editor information file.</li> </ul>

### Examples

Adds two subfolders (*sub01* and *sub02*) to the „MESFolder/MES01“ folder:

```
add_folders(strlist("MESFolder", "MES01"), strlist("sub01", "sub02"));
```

Adds a new sub-folder named *sub folder* to the root directory.

```
add_folders("", "sub folder");
```

Adds two files (*01.mes* and *02.mes*) to the folder "*MESFolder/my mes files*":

```
add_files(strlist("MESFolder", "my mes files"), strlist("01.mes", "02.mes"));
```

Adds the file (*file.mes*) to the root directory:

```
add_files("", "file.mes");
```

Adds the files from the template parameter *mes\_files* to the "*MES files*" directory:

```
add_files("MES files", value("mes_files"));
```

## 7.5 Command Reference

This section describes all commands that can be used in the *Constraints* and the *OutputFiles* section of the module information file.

Some notes on the nomenclature used here: functions have return values and a set of parameters. These functions are strongly typed and the used types are described in the following table:

Type	Description
<b>int</b>	This is a positive or negative integer. Internally this is a 64 bit number.
<b>double</b>	This is a floating point number. It's internally represented as a 64 bit number.
<b>int[]</b>	This is a list of integers.
<b>double[]</b>	This is a list of doubles.
<b>string</b>	A string.
<b>string[]</b>	List of strings.
<b>bool</b>	Boolean value. Can be true or false.

All commands are described by their signature:

```
name(<value> : <type>[, ...]) : <return_type>
```

The name is the identifier of the function. Value the name of the function parameter and type is the parameter type described in the table above.

Some functions support more than one parameter type. For these parameters the following syntax is used:

```
value : (<type1>, <type2>)
```

This means that a function parameter value can be of type *type1* or *type2*.

Optional parameters are marked with "[" and "]" braces.

Function	Description
<b>and</b> ( <a1>:bool [, <a2>:bool, ...]	Return the disjunction and conjunction of boolean values. Parameters:



<pre> ):bool  or( &lt;a1&gt;:bool [, &lt;a2&gt;:bool, ...] ):bool </pre>	<ul style="list-style-type: none"> <li>• a1, a2, ...: List of booleans</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• boolean</li> </ul> <p>Examples:</p> <ul style="list-style-type: none"> <li>• and(true, true) → true</li> <li>• or(true, false) → true</li> </ul>
<pre> buildpath( &lt;dir&gt;:(string, string[]), &lt;file&gt;:(string, string[]), &lt;ext&gt;:(string, string[]), ):(string, string[]) </pre>	<p>Builds a complete path from a directory, a file name and an extension string using the native path separator.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• dir: Directory</li> <li>• file: File base name</li> <li>• ext: File extension.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• string: if all parameters are strings</li> <li>• string[]: if one parameter is a list of strings.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• buildpath("C:\dir", "myfile", ".png") → "C:\dir\myfile.png"</li> </ul>
<pre> concat( &lt;s1&gt;:(string, string[]), [&lt;s2&gt;], ):(string, string[]) </pre>	<p>Concatenates strings and lists of strings.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• s1, ...: A list of strings.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• string, when s1 ... sN are strings.</li> <li>• string[], when one of the parameters is a list of strings.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• concat("img_", range(1, 10), ".bmp") → ["img_1.bmp", "img_2.bmp", ... "img_10.bmp"]</li> </ul>
<pre> compare( &lt;op&gt;:(Equal, Greater, Less, LessEqual, GreaterEqual), &lt;v1&gt;:(int, double, String), &lt;v2&gt;:(int, double, String) ):bool </pre>	<p>Compares two integer or double values. If the two parameters are strings this function does a lexicographical compare.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• op: Compare operation. Must be on of Equal, Greater, Less, LessEqual, GreaterEqual.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• boolean</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• compare(Less, param("i"), param(j))</li> </ul>
<pre> digits( n:int ):int </pre>	<p>Returns the number of digits of a given integer number. This is useful in combination with the format function (see width parameter).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• n: Integer</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• Number of digits as an integer</li> </ul>

	<p>Example:</p> <ul style="list-style-type: none"> <li>• <code>digits(1) → 1</code></li> <li>• <code>digits(100) → 3</code></li> </ul>
<pre>filebase(   f:string ):string</pre>	<p>Returns the base name of a file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>f</code>: Full file path.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• <code>string</code>: Base file name.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>filebase("c:\test\file.txt") → "file"</code></li> </ul>
<pre>fileext(   f:string ):string</pre>	<p>Returns the file extension of a file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>f</code>: File name.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• <code>string</code>: File extension.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>fileext("c:\test\file.txt") → "txt"</code></li> </ul>
<pre>filedir(   f:string ):string</pre>	<p>Returns the directory of a given file path.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>f</code>: File name</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• <code>string</code>: Directory name</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>filedir("c:\test\file.txt") → "c:\test"</code></li> </ul>
<pre>format(   &lt;value&gt;:(int, int[]),   [&lt;width&gt;:int,   &lt;fillchar&gt;:string] ):(string, string[])  format(   &lt;value&gt;:(double, double[]),   &lt;format_type&gt;:string,   &lt;prec&gt;:int] ):(string, string[])</pre>	<p>Formats an integer and double values. Also arrays of integers and arrays of doubles are supported.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• <code>value</code>: Numerical value to format.</li> <li>• <code>width</code>: Width of the integer to format. Using the <code>digits()</code> function here is recommended in the most cases.</li> <li>• <code>fillchar</code>: Filling character.</li> <li>• <code>format_type</code>: Formatting style for double values: 'f' (decimal) or 'e' (exponential).</li> <li>• <code>Prec</code>: Precision describes the number of decimals.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• Returns a string or a list of string depending on the input parameters.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>format(10, 3, "0") → „010“.</code></li> <li>• <code>format(13.5931, 'f', 2) → 13.59</code></li> </ul>
<pre>ini(   &lt;filename&gt;:string,   &lt;section&gt;:string,   &lt;key&gt;:string,</pre>	<p>Reads a key from an INI file.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• <code>filename</code>: Complete path to the INI file.</li> <li>• <code>section</code>: Section name</li> </ul>

<pre>&lt;type&gt;:(Int, String, Double) ):(int, double, string)</pre>	<ul style="list-style-type: none"> <li>• key: Key name.</li> <li>• type: Defines how to read the value from the INI file. Parameters are: Int, String and Double.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• The value of the key in the defined section.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• Reads an integer value from the file "tst.ini": inifile("test.ini", "section", "key", Int)</li> </ul>
<pre>max(   &lt;a&gt;:(int, double),   &lt;b&gt;:(int, double) ) :(int, double)</pre>	<p>Return the maximum of two values.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• a, b: The two values to compare. Can be a double or an integer value.</li> </ul> <p>Return values:</p> <ul style="list-style-type: none"> <li>• an integer or double depending on the parameter type.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• max(39, 55) → 55</li> </ul>
<pre>min(   &lt;a&gt;:(int, double),   &lt;b&gt;:(int, double) ) :(int, double)</pre>	<p>Return the minimum of two values.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• a, b: The two values to compare. Can be a double or an integer value.</li> </ul> <p>Return values:</p> <ul style="list-style-type: none"> <li>• An integer or double depending on the parameter type.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• min(-4, 2) → -4</li> </ul>
<pre>param(   &lt;name&gt;:string ) :(int, double, string)</pre>	<p>Returns the value of the parameter. Can be an integer, a double or a string.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• name: Name of the parameter.</li> </ul> <p>Return values:</p> <ul style="list-style-type: none"> <li>• an integer, double or string depending on the parameter type.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• param("j")</li> </ul>
<pre>range(   &lt;start&gt;:int,   &lt;end&gt;:int ) :int[]</pre>	<p>Generates a list of values within a given range.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• start: Starting value.</li> <li>• end: End value.</li> </ul> <p>Return values:</p> <ul style="list-style-type: none"> <li>• int[]: Integer list from start to end.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• range(0, 10) → [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]</li> </ul>
<pre>strlist(   &lt;s1&gt;:string,   [&lt;s2&gt;:string, ...]</pre>	<p>Generates a list of strings from the parameters.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• s1: First string</li> </ul>

):string[]	<ul style="list-style-type: none"> <li>• s2, ...: Second string, ...</li> </ul> Return values: <ul style="list-style-type: none"> <li>• string[]: String list from start to end</li> </ul> Example: <ul style="list-style-type: none"> <li>• strlist("a", "b", "c")</li> <li>• Returns ["a", "b", "c"]</li> </ul>
<pre> gplfile(   &lt;file_name&gt;,   &lt;section&gt;   [, &lt;type&gt;:(Int, String,   Double)] ):(int, string, double)] </pre>	Reads the parameter section from a gpl file. Parameters: <ul style="list-style-type: none"> <li>• file_name: Full path to the *.gpl file name</li> <li>• section: Name of the value in the parameter section</li> <li>• type: Can be Int, String or Double.</li> </ul> Return values: <ul style="list-style-type: none"> <li>• Parameter value from the gpl file.</li> </ul> Example: <ul style="list-style-type: none"> <li>• gplfile("param", Int)</li> </ul>
<pre> inuse(   &lt;param&gt;:string ):boolean </pre>	Checks if a parameter is marked active. Parameter: <ul style="list-style-type: none"> <li>• param: Parameter name</li> </ul> Return values: <ul style="list-style-type: none"> <li>• Is true if parameter is check or false if parameter is not checked.</li> </ul> Example: <ul style="list-style-type: none"> <li>• inuse("o")</li> </ul>
<pre> not(   &lt;b&gt;:bool ):boolean </pre>	Negates the input boolean value. Parameter: <ul style="list-style-type: none"> <li>• b: Boolean value to negate</li> </ul> Return values: <ul style="list-style-type: none"> <li>• negation of b</li> </ul> Example: <ul style="list-style-type: none"> <li>• not(true) → false</li> </ul>
<pre> outfile():string </pre>	Returns the output file name. If the o parameter of the module is set it return the value of this parameter. If it's not set it returns the input files (fs parameter of the module) base name. Parameter: <ul style="list-style-type: none"> <li>• None</li> </ul> Return values: <ul style="list-style-type: none"> <li>• File name as string</li> </ul> Example: <ul style="list-style-type: none"> <li>• outfile() → Returns "file" for example.</li> </ul>
<pre> outdir():string </pre>	Returns the output directory. Returns the value of the o parameter of the module. If this value is not set it reads the directory from the fs parameter of the module. Parameter: <ul style="list-style-type: none"> <li>• None</li> </ul>

	<p>Return values:</p> <ul style="list-style-type: none"> <li>• Output directory string.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>outdir()</code> → Returns “C:\myprojects” for example.</li> </ul>
<code>pathsep():string</code>	<p>Returns the platforms native path separator: This can be a backslash “\” on Windows or a slash “/” on UNIX like platforms.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• None</li> </ul> <p>Return values:</p> <ul style="list-style-type: none"> <li>• string:</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>pathsep()</code> → “\”</li> </ul>
<code>seifile(   &lt;file_name&gt;:string ) :string[]</code>	<p>Returns a list of sensor file names from a SEI file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• <code>file_name</code>: Path of the SEI file name.</li> </ul> <p>Returns:</p> <ul style="list-style-type: none"> <li>• A list of output files.</li> </ul>
<code>if(   &lt;cond&gt;:bool,   &lt;true_cond&gt;,   &lt;false_cond&gt;)</code>	<p>Parameter:</p> <ul style="list-style-type: none"> <li>• <code>cond</code>: Boolean condition</li> <li>• <code>true_cond</code>: Expression to evaluate when <code>cond</code> is true.</li> <li>• <code>false_cond</code>: Expression to evaluate when <code>cond</code> is false.</li> </ul> <p>Return value:</p> <ul style="list-style-type: none"> <li>• The return value of either <code>true_cond</code> or <code>false_cond</code>.</li> </ul> <p>Example:</p> <ul style="list-style-type: none"> <li>• <code>if(compare(Greater, 50, 10), 42, 24)</code> → 42</li> </ul>

## 7.6 Examples

### Creating a list of images

This expression creates a list of image file names:

```
concat("img_", range(param("s"), param("e")), ".bmp")
```

The result is (assuming that `s=1` and `e=4`):

```
["img_1.bmp", "img_2.bmp", "img_3.bmp", "img_4.bmp"]
```

Alternatively, with leading zeros as fill characters:

```
concat("img_", format(range(param("s"), param("e")), digits(param("e")), "0"), ".bmp")
```

This will generate the following list of bitmap file names (assuming that `s=8` and `e=12`):

```
["img_08.bmp", "img_09.bmp", "img_10.bmp", "img_11.bmp", "img_12.bmp"]
```

## Testing flags

Tests if a certain parameter was set. If the parameter `gv` is in use a file named *file.ve* will be generated, otherwise no file will be generated:

```
if(inuse("gv"), "file.ve", "")
```

## 7.7 Readable File Formats

The framework supports the following file types:

- ces, mes, dat, cdat, v, sei, dati, inf, tab, epst, epsr, epsrad, uwrad, vel
- txt, log
- gpl, geo, grd, ggd
- uw, epsxy, velxy,
- hist
- bmp
- va, vx, vy, vt, vr, vi, ev

### Table Files

Table files are plain *ASCII* files. Each line represents a tabular record. The first two lines are reserved for the row name and the used unit. Columns are whitespace separated. Floating point numbers are in international format (decimal points instead of commas). This is equivalent to the ISO C standard for the "C" locale.

To describe a two column table with a time stamp in seconds and a holdup measurement in %, a typical tabular file could look like that (here an *\*.epst* file):

```
t eps(t)
s      %
0 90.4379
0.0004 90.8505
...     ...
```

File types which use the tabular file format are:

- \*.epst
- \*.epsr
- \*.epsrad
- \*.uwrad
- \*.vel

The framework uses the *2D plot* view to visualize tabular files.

### Matrix Files

Matrix files are ordinary *ASCII* files very similar to tabular files. Each line represents a row in a matrix. Columns are whitespace separated. Floating point numbers are encoded with the ISO "C" locale standard.

```

0 0 0 0 0 0 0 0 ...
0 0 0 0 0 0 1 1 ...
0 0 0 0 0 1 4 6 ...
0 0 0 0 1 2 1 1 ...
0 0 0 1 3 2 3 1 ...
0 0 1 2 7 6 1 4 ...
0 1 2 4 5 6 1 1 ...
...

```

File types which use the matrix file format are:

- \*.geo
- \*.ggd
- \*.uw
- \*.epsxy
- \*.velxy
- \*.err

It's also possible to save multiple matrices in a file. Multiple matrices are separated by an empty line.

File types which are in the multi matrix file format are:

- \*.grd

The *Wire Mesh Sensor Framework* uses the *3D Plot view* and the *Table view* to visualize tabular files.

## Binary Files

Binary files contain a byte stream of continuous frames. The data is saved in *Little-Endian* Format. Each data field can be one of the types listed in Table 14 depending on the file type:

Type	Size (bytes)	Description	File types
byte/unsigned char	1	Single byte value. Values range from 0 to 255.	*.v
uint16/unsigned short	2	Two bytes unsigned integer value. Values range from 0 to $2^{16}-1$ .	*.cdat; *.dat
int16/short	2	Two bytes signed integer. Values range from $-2^{15}$ to $2^{15}-1$ .	
uint32/unsigned int	4	Four bytes unsigned integer. Values range from 0 to $2^{32}-1$ .	*.b
int32/int	4	Four bytes signed integer. Values range from $-2^{31}$ to $2^{31}-1$ .	
float	4	Four bytes single precision floating point value (IEEE 754).	*.vx; *.vy, ...

Table 14: Used data formats in the framework.

The following table shows the structure of a binary file. The real data offset in bytes must be calculated by multiplying the offset with the size in Table 14.

Data Offset (multiple of data words!)	Frame	Column	Row	Description
0	0	0	0	Start of the first line of the first frame.
1	0	1	0	
...	0 ...	2 ...	0 ...	
width	0	0	1	Start of the second line of the first frame.
width + 1	0	1	1	
...	0 ...	2 ...	1 ...	
width x height	1	0	0	Start of the first line of the first frame.
width x height + 1	1	1	0	
...	1 ...	2 ...	0 ...	
2 x width x height	2	0	0	
...	3 ...	1 ...	0	
3 x width x height			0	
...	...	...	...	
num_frames x width * height - 1	num_frames - 1	width - 1	height - 1	Last data value in stream.

Usually these binary files are bundled with an information file.



## 8 Summery

The software algorithms described in this document have been implemented for the purpose of data evaluation of wire mesh sensor data. The single executables have been originally designed for stand-alone execution from the DOS command line or via a batch script. Recently the wire mesh sensor frame work has been launched to integrate these single modules under a graphical user interface, to allow the user to specify the parameters and file settings in a usual WINDOWS® software surface. Over more, the framework allows the users to implement their own software code and integrate the new modules into it due to a "module information file" specifying the required parameters, input and output files.

This is the first step to user-friendly homogeneous wire mesh sensor data evaluation software. We hope that this package will enable our scientific partners and customers to proceed their wire mesh sensor data unaffiliated.

## 9 References

- [1] M. J. Da Silva, „Impedance Sensors for Fast Multiphase Flow Measurement and Imaging,“ TUD Press, Dresden, 2008.
- [2] J. Huhn und J. Wolf, Zweiphasenströmung gasförmig/flüssig, Leipzig: Fachbuchverlag Leipzig, 1975.
- [3] H.-M. Prasser, E. Krepper und D. Lucas, „Evolution of the two-phase flow in a vertical tube - decomposition of gas fraction profiles according to bubble size classes using wire-mesh sensors,“ *International Journal of Thermal Sciences* 41, pp. 17-28, 2002.
- [4] H.-M. Prasser, D. Scholz und C. Zippe, „Bubble size measurement using wire-mesh sensors,“ in *Flow Measurement and Instrumentation* 12, 2001, pp. 299-312.
- [5] H.-M. Prasser und M. Beyer, „Bubble recognition algorithms for the processing of wire-mesh sensor data,“ in *paper:S7\_Thu\_B\_50*, Leipzig, 09.-13.07.2007.
- [6] H.-M. Prasser, M. Beyer, H. Carl, A. Manera, H. Pietruske und P. Schütz, „Experiments on upwards gas/liquid flow in vertical pipes,“ FZD-482, Dresden, 2007.