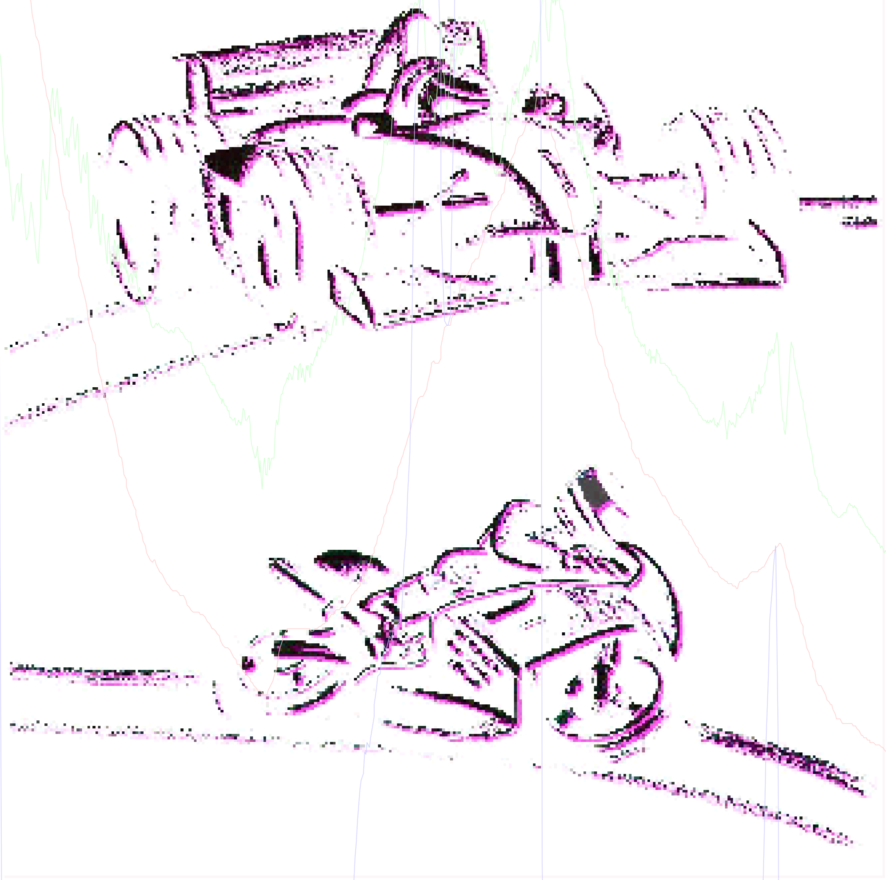
- English -



2D MATLAB interface: MathToolEngine

modified 17/11/17

Content

Symbols used in the text

These paragraphs contain tips and practical advice for working with the 2D system

In the paragraphs highlighted with this symbol, you will find additional information and it is very important that you follow the instructions given.

Documentation reference

- A user manual reference number is provided so the user can seek further assistance

1 Overview

With the MathToolEngine 2D provides an interface between 2D measurements and the multi-paradigm numerical computing environment MATLAB. It enables MATLAB to read 2D channel files and SpecSheets, do any vector based calculation on this data and store the results back in the same 2D channel files.

The calculation either could be done within the analyzer software or started out of MATLAB. In the following we show both ways with an easy example. Later in this documentation we give a description of the architecture of the MathToolEngine.

2 Running MATLAB calculation on 2D measurements

There are two different ways to combine the calculation power of MATLAB with the data structure of 2D measurements. The difference is the starting point of the calculation.

1) It is possible to work with MATLAB and access the data of 2D measurements using the M-files provided by 2D, which are described below.
2) The second possibility is to start the calculation of M-files from within the 2D Analyzer. For this the Analyzer starts MATLAB in the background and passes the M-files to and receives results from MATLAB by inter-process communication.

2.1 *Accessing 2D measurements with MATLAB*

If you are familiar with MATLAB and its user interface you might prefer to define and run your calculations on 2D data from within MATLAB. In the following we explain how to do that on a simple example M-file, which is delivered with the software: Sample_StandAlone_2014.m.

The sample M-files are stored in the subfolder “MathFiles” which is located under the user data folder of the 2D installation. To open a Windows Explorer in that folder press the key combination <CTRL><ALT><U> on the main window of WinARace. Navigate into “MathFiles” and open “Sample_StandAlone_2014.m” by double click. MATLAB shows the file in its main window like the following:

```
% -------------------------------------------------------------------------
% START OF PRE-PROCESSOR PART - DO NOT TOUCH
% -------------------------------------------------------------------------

scriptName = mfilename('file');
scriptName = strcat(scriptName,'.m');

MTEPath = getenv('2d_MTEPath');
addpath(MTEPath);

Matlab2D_preProc;

% -------------------------------------------------------------------------
% END OF PRE-PROCESSOR PART - DO NOT TOUCH
% -------------------------------------------------------------------------
% START OF USER PART - AS WOULD BE WRITTEN IN MATHTOOL FROM ANALYZER
% -------------------------------------------------------------------------

% Demonstration: Matlab StandAlone run
% Needed input channels: RPM
% New channel to be created: "RPM_Matlab"

% Add 1000 to RPM and multiply the result with 1.1

RPM_Matlab = (RPM + 1000) * 1.1;
RPM_Matlab.unit  = RPM.unit;

% -------------------------------------------------------------------------
% END OF USER PART - AS WOULD BE WRITTEN IN MATHTOOL FROM ANALYZER
% -------------------------------------------------------------------------
% START OF POST-PROCESSOR PART - AS PREPARED BY MATHTOOL PARSER
% -------------------------------------------------------------------------

Matlab2D_postProc;

% -------------------------------------------------------------------------
% END OF POST-PROCESSOR PART - AS PREPARED BY MATHTOOL PARSER
% -------------------------------------------------------------------------
```

In the example one can see, that the calculation consists of three main parts which are separated by some comment lines. As the comments in the file suggest, the first and the third part always have to look like shown in the example. These parts handle the access to the 2D data structure.

The first part makes the 2D measurement available for calculation in MATLAB.

The filename of the current M-file is stored in the variable “scriptName”. Since the MATLAB function “mfilename” does not give the file extension back, it is added in the next line. The variable is used from the script “parseScript” (see 3.1.12) which retrieves the channel names used in the M-file that has to be executed.

MATLAB has to know, in which folder the M-files for the 2D access are stored. During installation 2D sets an environment variable of Windows named “2d_MTEPath”. This variable holds the folder of the 2D M-files. This variable is retrieved by MATLAB and stored in the variable “MTEPath”. In the next line this folder is added to the searching path of MATLAB. That makes the 2D scripts available for usage.

The last line of the first part calls the preprocessor “Matlab2D_preProc”. This powerful script does all necessary steps to access a 2D measurement. Called directly from MATLAB, the script calls a measurement selection dialog. After the user selected a .MES folder the script will automatically load the channels required by the script.

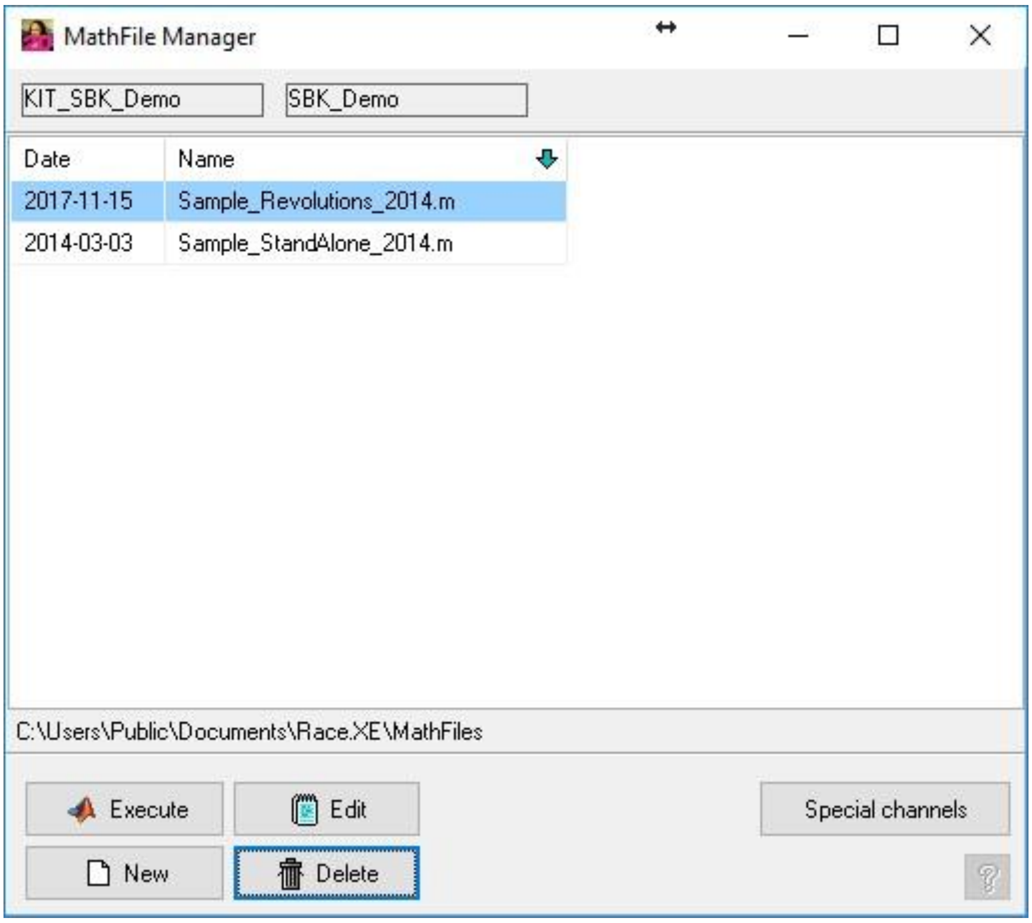
The second part of the script is the user part. Here the user can define all needed calculations. A new channel can be created by using a name for a new variable that is not already existing as channel name. By assigning an existing 2D channel the new channel is defined. In the example the new channel “RPM_Matlab’ is created by loading the existing channel “RPM”, adding 1000 to each value and multiply the result by 1.1. In the next line the dimension of the new channel is set to the dimension of the input channel.

The third and last part of the sample file only calls the postprocessor. That writes all changed or added channels and SpecSheet values back to the 2D measurement.

2.2 Run M-files in the Analyzer

It is possible to define and run M-files directly in the 2D Analyzer. For the calculation of these files the Analyzer runs MATLAB in the background.

To define or execute a MATLAB calculation out of the Analyzer you have to choose “Functions” – “Math files” from the main menu. That starts the MathFile Manager:



Date	Name
2017-11-15	Sample_Revolutions_2014.m
2014-03-03	Sample_StandAlone_2014.m

C:\Users\Public\Documents\Race.XE\MathFiles

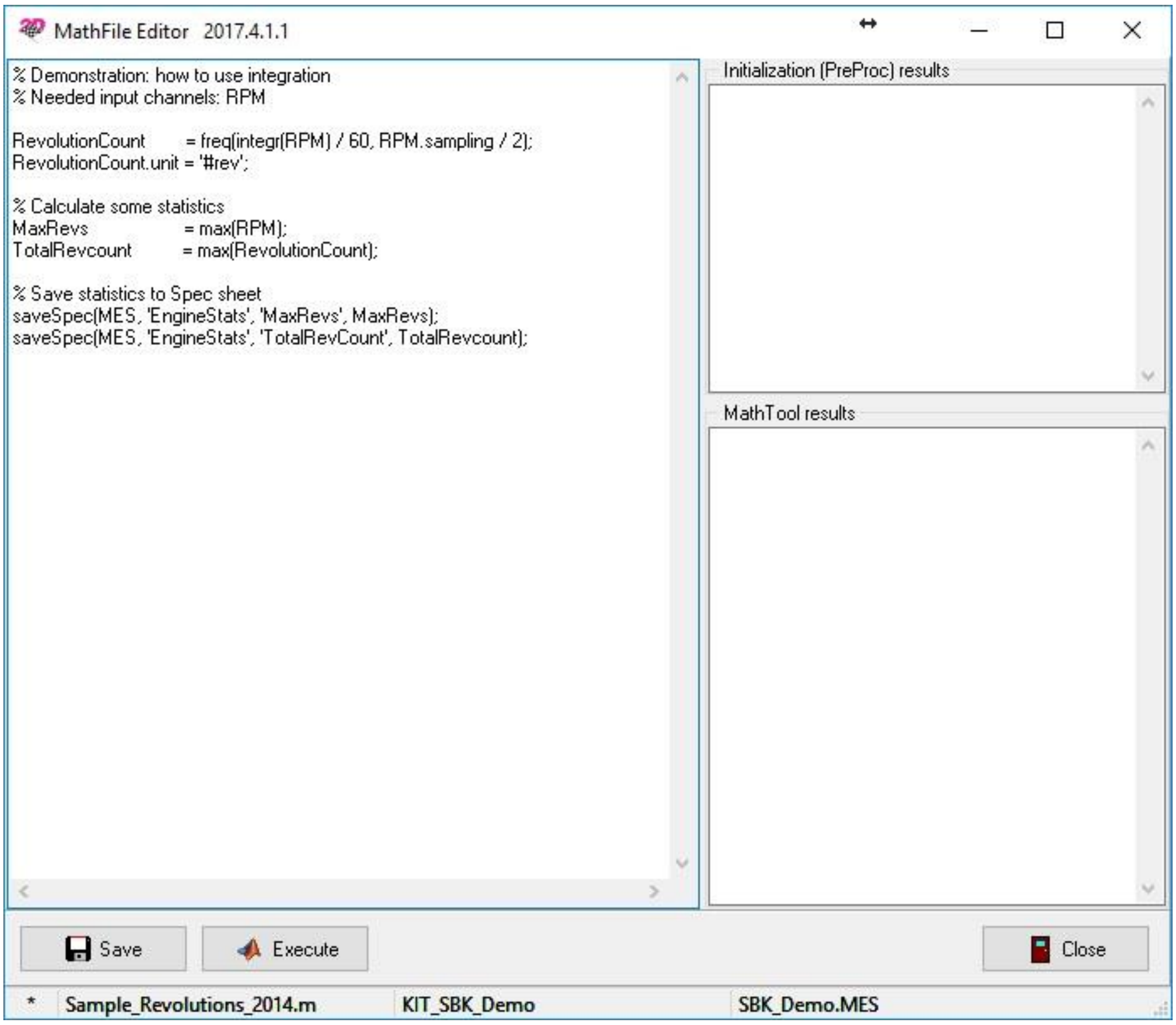
In the manager you see all existing user M-files in a list with the date of the last modification and the filename of each file.

Under that list you find all buttons needed to manage your files:

- **Execute**: This starts the calculation with the file currently selected in the list. Everything is done in the background. When the calculation finished the Analyzer automatically reloads the current measurement.
- **Edit**: That button starts the MathFile Editor with the currently selected file. A short description of the Editor follows below (2.2.1).
- **New**: After asking for a name for the new file the MathFile Editor is called with an empty file
- **Delete**: Deletes the currently selected file.
- **Special channels**: Open the dialog for the definition of special channels. (See more in Analyzer help)

2.2.1 MathFile Editor

When an existing file is changed or a new file is created the Analyzer calls the MathFile Editor. Here with the sample file loaded:



```
% Demonstration: how to use integration
% Needed input channels: RPM

RevolutionCount      = freq(integr(RPM) / 60, RPM.sampling / 2);
RevolutionCount.unit = '#rev';

% Calculate some statistics
MaxRevs              = max(RPM);
TotalRevcount        = max(RevolutionCount);

% Save statistics to Spec sheet
saveSpec(MES, 'EngineStats', 'MaxRevs', MaxRevs);
saveSpec(MES, 'EngineStats', 'TotalRevCount', TotalRevcount);
```

* Sample_Revolutions_2014.m KIT_SBK_Demo SBK_Demo.MES

The editor is divided in three parts. The large box on the left is the editor, in which the M-file is written. The upper box on the right shows any error messages coming from MATLAB when it runs the preprocessor part (3.1.13). In the lower box all errors of MATLAB are shown which occurred during the rest of the calculation.

If you compare this example script with the one of the chapter before, you can see that the pre- and postprocessor commands are missing. The analyzer adds these automatically, so the user can focus on the actual calculations..

2.2.1.1 Example

The sample file for the calculation out of the Analyzer shown in the picture above does the following:

In the first line that is no comment and not empty a new channel “RevolutionCount” is defined. One can see that it is possible to nest one MATLAB function in another. Here the inner function “integr(RPM)” creates internal a new channel as integration of the existing input channel “RPM”. The result of this is divided by 60, which changes revolutions per minute to revolutions per second. Now this is the input for the outer function “freq()” which creates a new channel by changing the sampling rate to half of the rate of the input channel “RPM”.

The next line sets the dimension of the new channel to “#rev”.

Two statistic values are calculated by using the function “max” on the input channel “RPM” and the new channel “RevolutionCount”.

At the end of the file these two values are written back into the SpecSheet of the 2D measurement.

3 Architecture of the MathToolEngine

MATLAB provides an interface to access data and calculations from other software systems. These have to provide a dynamic link library (DLL) with all necessary functions to access the data. In the 2D software suite two versions of this DLL type are available, one for 32 bit and one for 64 bit version of MATLAB: MesAccess.DLL and MesAccess64.DLL.

To pass information to MATLAB about the functions available in this DLL, a header file has to be written, containing the functions of the DLL and the types of parameters passed to the functions. In the 2D MathToolEngine there are two files with that description: MesAccess.h and MesAccess64.h. Both files have the same content, but one is used for the 32-bit version of MATLAB and the other for 64 bit.

These header files were imported by MATLAB with a special import function. With this function MATLAB creates a special DLL through which it accesses the user DLL. MESAccess_thunk_pcwin64.DLL is the one created to access with 32-bit version of MATLAB and MESAccess64_thunk_pcwin64.DLL is for 64 bit.

The selection of a 2D measurement is a function called from the MesAccess.DLL or MesAccess64.DLL depending on the architecture of MATLAB. But these DLLs call other external DLLs for the selection: FileSel_M.DLL and FileSel_M64.DLL. Again one file for 32 and one for 64 bit MATLAB.

The rest of the MathToolEngine is a set of M-files. These are simple text files containing commands, which are interpreted and executed by MATLAB directly.

3.1 *M-Scripts in the MathToolEngine*

The 2D MathToolEngine does contain several predefined MATLAB scripts, which are necessary to read data from 2D measurements and to write changed or added data back. As you have seen in the examples above you normally do not need to know much about them. The most important are the pre- and the postprocessor. These combine the usage of the others in a way that they read all needed channels and SpecSheet values and write them back at the end of the calculation. In the following we give a short description of each file and its functions.

3.1.1 chann2d.m

This MATLAB script is the largest of the provided scripts. It defines the datatype “chann2d” and a set of basic calculations possible with that datatype. This file must not be changed.

3.1.1.1 The datatype chann2d

The most important part of that M-file is the declaration of the class chan2d. The class contains tree properties:

- The first property it the dimension of the channel and is called “unit”.
- The second is the sampling rate of the channel and has the name “sampling”.
- The third and last property is “data”, it represents a vector of float numbers. These numbers are the value of the channel at each sampling point.

The class chan2d does contain three methods:

- chan2d.timearr returns a vector containing the time in [units] value to each channel value.
- The method chan2d.length returns the number of values stored in that channel. That is the length of the vector chan2d.data.
- The sampling rate in Hz of a channel can be retrieved as double precision float with the method chan2d.sampl.

3.1.1.2 Basic functions with chan2d

Besides the class definition of chan2d there are 42 functions defined, which can be used with the datatype chan2d. The following table gives an overview to these functions

Function	Possible parameters	Description
+	(chan2d,chan2d)	Addition of two channels or a channel and a given number
	(chan2d,number)	
	(number,chan2d)	
-	(chan2d,chan2d)	Subtraction of two channels, channel minus number or number minus channel
	(chan2d,number)	
	(number,chan2d)	
*	(chan2d,chan2d)	Multiplication of two channels or a channel with a number
	(chan2d,number)	
	(number,chan2d)	
/	(chan2d,chan2d)	Division of two channels, channel divides by number or number divides by channel
	(chan2d,number)	
	(number,chan2d)	
Unary -	chan2d	Change sign of all values of the channel (same as multiplication with -1)
Unary +	chan2d	Result is a new channel as copy of input
^	(chan2d,chan2d)	Result is one channel power the other, a number power a channel or a channel power a number
	(chan2d,number)	
	(number,chan2d)	
sqrt	chan2d	Square root of a channel
sin	chan2d	Sine of a channel (radiant)
dsin	chan2d	Sine of a channel (degree)
sind	chan2d	Sine of a channel (degree)
cos	chan2d	Cosine of a channel (radiant)
dcos	chan2d	Cosine of a channel (degree)
cosd	chan2d	Cosine of a channel (degree)
tan	chan2d	Tangent of a channel (radiant)
dtan	chan2d	Tangent of a channel (degree)
tand	chan2d	Tangent of a channel (degree)
asin	chan2d	Invers sine of a channel (radiant)
arcsin	chan2d	Invers sine of a channel (radiant)
asind	chan2d	Invers sine of a channel (degree)
acos	chan2d	Inverse cosine of a channel (radiant)
arccos	chan2d	Inverse cosine of a channel (radiant)
acosd	chan2d	Inverse cosine of a channel (degree)
atan	chan2d	Invers tangent of a channel (radiant)
arctan	chan2d	Invers tangent of a channel (radiant)
dtand	chan2d	Invers tangent of a channel (degree)
atan2	(chan2d,chan2d)	Four-quadrant inverse tangent of each point
	(chan2d,number)	
	(number,chan2d)	
abs	chan2d	Change all values to absolute values
lt	(chan2d,chan2d)	Less than comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	

Function	Possible parameters	Description
gt	(chan2d,chan2d)	Greater than comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	
le	(chan2d,chan2d)	Less or equal comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	
ge	(chan2d,chan2d)	Greater or equal comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	
ne	(chan2d,chan2d)	Not equal comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	
eq	(chan2d,chan2d)	Equal comparison of two channels, a number and a channel or a channel and a number
	(chan2d,number)	
	(number,chan2d)	
not	chan2d	Logical inverse of a channel
freq	(chan2d,number)	Create a new channel as copy of input channel, but with new sampling rate by linear interpolation
F	(chan2d,number)	Moving average filter over the given number of values
shift	(chan2d,number)	Shift a channel to the right by the given number of seconds, negative time to shift to the left
deriv	chan2d	Derivate a channel
integr	chan2d	Integrate a channel
min	chan2d	Find the minimum value of a channel
max	chan2d	Find the maximum value of a channel

3.1.2 readMeas.m

This MATLAB script provides the needed function to read a 2D measurement. The function "readMeas" can be called with either of these three different parameters:

- None

  Without parameter the function runs in interactive mode and the user will be asked to select a 2D measurement in a selection dialog.

- Path to DDD File

  Open 2D measurement directly by providing the path to the DDD file.

- Path to .MES Folder

  Open 2D measurement directly by providing the path to the .MES folder.

The result of the function is an instance of the class MES. That class has the following properties:

Property	Meaning
MES.EventName	Name of the current event (Folder in which .MES folders are located)
MES.MESName	Name of the measurement (Last part of measurement folder including .MES)
MES.MESDir	Complete folder of the measurement (same as EventName + MESName)
MES.DDDName	Filename of the DDD file without folder but with file extension .DDD
MES.DDDFile	Complete filename of the DDD file (Same as MESDir + \ + DDDName

3.1.3 expandMES.m

This script consists only of one function, which has the same name as the file. The function does not accept any parameters. It adds the list of channels to a MES structure, if any available, otherwise an error is shown. After the function succeeded the MES structure has the property CHAN set, which is an array of all channels of the MATLAB type chan2d.

3.1.4 isValidMES.m

With this function a MES structure can be checked if it does contain all necessary properties. Again, this file does only contain one function: "isValidMES(MES)". It returns 1 if the MES is usable otherwise a 0.

3.1.5 readChanList.m

This script is used to retrieve the list of channels stored in a 2D measurement. The function "readChanList" needs an instance of the class MES as parameter, which has been created by the function "readMeas". The result of the function is a string containing entries for each channel separated by a ",". Each entry does contain three values separated by a ';'.

The three values are

- Name: name of the channel
- Dimension: dimension of the channel
- Sampling ratio: divisor of the main sampling rate to get the channel rate

3.1.6 readChan.m

To be able to do any calculation on 2D channel data, these have to be loaded into the memory of MATLAB. This is done by this script. It provides the function "readChan". The function can be called with either one or two parameters. The first parameter again has to be an instance of the class MES. As second parameter a list of a channel names to read can be given. With only one parameter all channels are loaded. After the function the channels can be accessed by MES.CHAN, which is an array of chan2d.

3.1.7 saveChan.m

Calculations with 2D channels may result in new or changed channel data. This script contains the function "saveChan". It writes channel data back to the 2D measurement. To do this a MES structure and a chan2d has to be passed as parameter to that function.

3.1.8 readSpec.m

A 2D measurement always does contain a so called SpecSheet. This is a text file containing information about the specifications of the measurement. It is organized in groups, which contain identifiers. For each identifier a value, numbers or text, can be stored. In some cases, it might be useful to use values from this specification file in a MATLAB calculation. This script can be used to read values from the SpecSheet for further calculation.

The first parameter has to be the MES structure retrieved with "readMeas". The name of the group is passed as second parameter and the identifier as third.

3.1.9 readSpecGroups.m

A list of all available groups in the SpecSheet can be retrieved using the function "readSpecGroups". The only parameter for that function is a MES structure.

3.1.10 readSpecEntries.m

Each group of a SpecSheet does contain several identifiers. That list can be read using the function "readSpecEntries". Again, the first parameter has to be a MES structure. The second has to be the name of the group, from which the identifiers should be listed.

3.1.11 saveSpec.m

The access to the SpecSheet is not limited to reading the values. It is also possible to change or add values in that SpecSheet. That is done with the function "saveSpec". Pass a MES structure as first parameter, followed by the name of the group, the identifier and the new value, that should be written.

3.1.12 parseScript.m

If a given M-file calculates with 2D channels as input parameter, these channels have to be loaded by MATLAB. The function "parseScript" in this M-file gets a list of channel names and the name of the script to execute as parameter. The list of channel names normally is the result of the function "readChanList" (see 3.1.5). The function tries to find channel names in the given script. If matches are found these channels are loaded automatically.

MATLAB will show a warning "No channels available" if no match was found.

It is very important to use case sensitive channel names. If your 2D measurement contains a channel "RPM" but you try to use the channel "rpm" in a calculation, that will not work, because these are different names for MATLAB.

3.1.13 Matlab2D_preProc.m

This script has to be used every time MATLAB should have access to 2D measurements. There is only one function contained, which does not accept any parameters: "Matlab2D_preProc". It calls the dialog to select a 2D measurement, but only when called out of MATLAB, if called from Analyzer, the measurement is given by the active measurement of the Analyzer.

After the selection all channels are available for calculation inside MATLAB. This script uses functions which are defined in separate MATLAB scripts described above.

After the script was executed, all needed 2D channels are loaded and all SpecSheet values are available.

3.1.14 Matlab2D_postProc.m

The counterpart to the file above is Matlab2D_postProc.m. The function "Matlab2D_PostProc" in this script writes all changes made by MATLAB back to the 2D measurement. For that it has to be the last command of a calculation, every time channels are added or changed.

No parameters are accepted.

This script combines some of the functions above to do all necessary to write changes back to the 2D measurement.