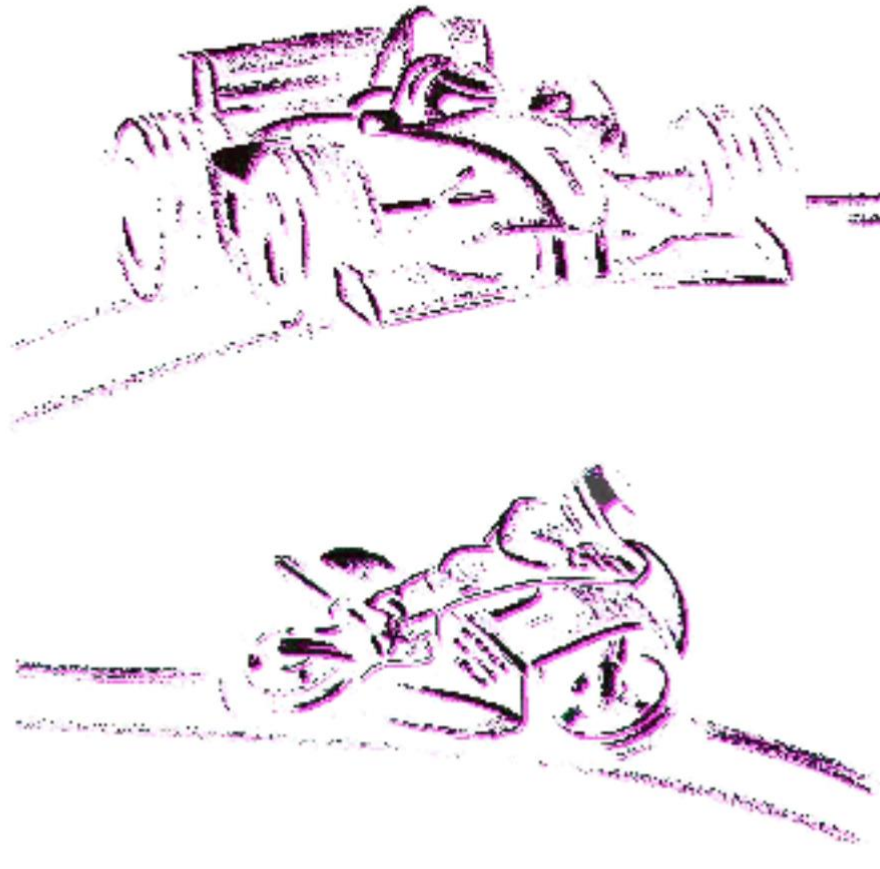


- English -



2D CalcTool

1 Notes and symbols used in this Manual



In the paragraphs highlighted with this symbol, you will find tips and practical advice to work with the 2D-Software.



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



Documentation reference

- A user manual reference number is provided so the user can find additional information

Revision History

Revision	Description	Release Date	Author
0	Initial Release	2008-10-14	
1	Revision 1	2017-03-04	
2	Revision 2	2020-07-21	FS
3	Revision 3	2021-09-10	FS
4	Revision 4	2022-01-11	FS

Revision 0

Initial release of *2D CalcTool* manual

Revision 1

Addition of the latest commands

Revision 2

Fundamental revision and addition of the latest commands

Revision 3

Addition of the commands for easier analysis of WhileTrue-conditions.

- *MinWhileTrue* (6.3.14.4.3)
- *PosMinWhileTrue* (6.3.14.4.4)
- *MaxWhileTrue* (6.3.14.5.3)
- *PosMaxWhileTrue* (6.3.14.5.4)

Also, the including of calculation files was improved because of better file-extension-handling (6.1.2.3.1), multi-stage path-handover (6.1.2.3.2) and new path-placeholders <SysCal> and <UserCal> for include calls (3.3).

Revision 4

Insertion of a separate chapter for the description of the prepared CAL-files (8) and toolchains (9) already delivered by 2D-Datarecording.

Addition of the latest commands for conditional execution of calculation groups

- *IfOrgExists(#CH)* (6.1.3.5)
- *IfNotOrgExists(#CH)* (6.1.3.6)

2 Content

1	Notes and symbols used in this Manual	2
2	Content	4
3	Basics	5
3.1	What is a calculation file?	5
3.2	How to create/edit CAL-files	6
3.3	2D directories	7
3.4	Calculation File Manager	8
3.5	General structure of CAL-file	9
3.6	Creating permanent channels	10
3.7	Replacing channels	10
3.8	Creating constants	10
3.9	Editing assistance	11
3.10	Executing CAL-files	12
4	SpecSheet	14
4.1	Structure	15
4.2	Read access	16
4.3	Write access	16
5	Phases	16
6	Pre- and Postprocessor	17
6.1	Pre-processor functions	17
6.3	Postprocessor-functions	35
7	Table of all calculation functions	101
8	Predefined CAL-files	106
8.1	2D_DistanceAndTimeCH.CAL	106
8.2	2D_LapChannels.CAL	106
8.3	2D_SOD_Reverse.CAL	107
8.4	GearCount2D.CAL	108
8.1	2D_Conditions	110
9	Toolchains	112
9.1	2D_GPSAuto	112
9.2	2D_GPSTracks	112
9.3	2D_ValidateTracks	112
9.4	2D_FilterAndRotate	113
9.5	2D_Comfort	113
9.6	Combining toolchains	114

3 **Basics**

Within *CalcTool* the channels recorded by a 2D data acquisition system can be further processed with the help of text-based calculation files to e.g. visualize data for development purposes in various plots provided by the 2D software or for further calculations.

Further processing includes among others:

- Mathematical calculations
- Trigonometrical calculations
- Filtering
- Bit manipulation operations
- Signal analysis (Min, Max, Average, ...)

3.1 **What is a calculation file?**

A calculation file is a simple text-based file which contains the instructions for a calculation. There are two different types of calculation-files:

Encrypted calculation files	*.CCF	unreadable and non-editable
Unencrypted calculation files	*.CAL	readable and editable

Unencrypted files (.CAL) can be opened by any user, even via the editor, and changed if necessary. However, encrypted calculation files (.CCF) can only be decoded and then modified by 2D-Datarecording members.



- Encryption of a calculation file does not always mean that the file contains secret information. Files are often encrypted to prevent editing by other users.



- Team license managers can get an application of 2D data recording to encrypt files! If required, please send us an e-mail to Info@2D-datarecording.com containing your license name as this application will be linked to your license name.
- These CCF files are used to secure calculation files against changes, read access or modifications by others.

3.2 How to create/edit CAL-files

Since CAL-files are text-based files, there are several ways to edit them:

CalcEdit.exe	Editor
<p><i>CalcEdit.exe</i> is an automatically installed 2D-CAL-file-editing application, which is specially adapted to the editing of CAL-files.</p> <p>This application offers the following advantages: (see 3.9)</p> <ul style="list-style-type: none"> - Syntax checker (syntax examination) - Automatic code completion - Parameter suggestion for instructions - Instruction list (command list) - Coloured distinction <p>All CAL-files are linked to the program. By double-clicking the respective CAL-file is displayed in <i>CalcEdit.exe</i>.</p>	<p>Calling a CAL-file via any word processing computer program</p>

Editing CAL-files via *CalcEdit.exe* is usually done from the *Analyzer*. For this purpose, the *Calculation File Manager* (see 3.4) is opened via the tab *Functions* → *Calculation File Manager*. Alternatively, the key combination <ALT> + <N> can be pressed in the *Analyzer*.



- The following link shows an explanation how to create and execute a CAL-file: <https://www.youtube.com/watch?v=8COo3toU44s>

3.3 2D directories

To generalize file paths, there are certain placeholders in the 2D software which stand for different directories.

The possible placeholders are:

Name of directory	Short-cuts	Placeholder
Event directory	<CTRL> + <ALT> + <E>	<EventDir>
Measurement directory	<CTRL> + <ALT> + <M>	<MesDir>
Race application directory	<CTRL> + <ALT> + <A>	<AppDir>
Race application CAL directory		<RaceDir>\SYSTEM\CAL\ <SysCal>
User data directory	<CTRL> + <ALT> + <U>	<UserDataDir> <RaceDir>
User data CAL directory		<UserDataDir>\CalFiles\ <UserCal>
Application data directory	<CTRL> + <ALT> + <D>	<AppDataDir>
RootDirectory\TMP		<TempDir>

Name of directory	Short-cuts	Placeholder
Name of active measurement		<MesName>
Name of active event		<EventName>

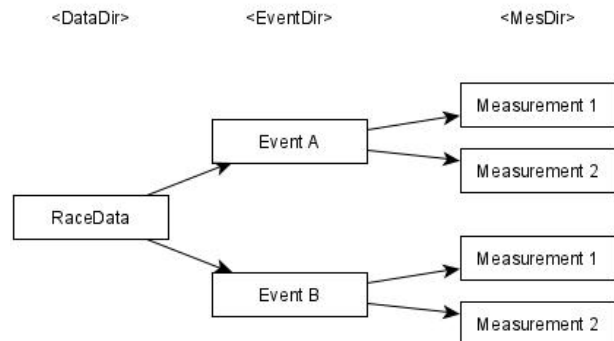


- These placeholders can also be used at file-paths when files are called!
- The respective paths on your PC can be seen at:
WinARace → *Settings* → *Folders -Protocoll*

DataDir, EventDir & MesDir

The definition of the directories is done backwards from the measurement-directory:

The individual measurements are located inside the measurement directories, which in turn are located inside the respective event directories. The event directories are combined in the data directory. This data directory is usually located directly after the root directory, e.g. C:\ and is normally called RaceData, so the path ends up as C:\Racedata\.



RaceDir

Race directory contains among others the 2D sub-applications.

UserDir

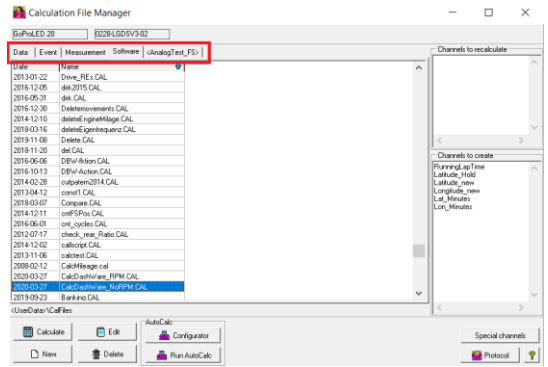
User directory contains various user-related data such as templates, CAL files and scripts.

3.4 Calculation File Manager

The *Calculation File Manager* is opened via the tab *Functions* → *Calculation File Manager*. Alternatively, the key combination <ALT> + <N> can be pressed in the *Analyzer*.

The CAL-files can be stored in different directories on your computer. The red box is highlighting the directories.

It depends on the application of the respective CAL-file where it should be stored.



Directory	Path	WinARace-Shortcut
Data	<...><DataDir>	
Event	<...><DataDir><EventDir>	[CTRL] + [ALT] + [E]
Measurement	<...><DataDir><EventDir><MesDir>	[CTRL] + [ALT] + [M]
Software	<RootDir>\Users\Public\Documents\Race2020\CalFiles	[CTRL] + [ALT] + [U]
<...>	<RootDir>\Users\Public\Documents\Race2020\CalFiles\<...>	

By using the directories correctly and storing the Events and Measurements on a server/data storage, a team-internal CAL-file management structure can be set up:

Directory		Description
Data	Server/Data storage	In this directory calculation files are stored which should be accessible for every other Analyzer user of a team. These calculation files are valid for all Events and Measurements.
Event		In this directory calculation files are stored which should be accessible for every other Analyzer user of a team. These calculation files are only valid for the respective Events and its Measurements.
Measurement		In this directory calculation files are stored which should be accessible for every other Analyzer user of a team. These calculation files are only valid for the respective Measurements.
Software	Users PC	In this directory calculation files are stored which should be accessible only for one, dedicated Analyzer user of a team.
<...>		In this directory calculation files are stored which should be accessible only for one, dedicated Analyzer user of a team. Compared to Software calculation files, these files can be stored in different sub-directories for better organisation.



- **Accessible** means that the calculation files can be executed by any user. It should be noted that if the calculation files are saved as **CAL** files, that any user can also edit them. If a file is to be executable by any user, but not editable by any user, it must be saved as a **CCF** file (see 3.1).
- The calculation files stored in *Software* or *<...>* are usually debugging CAL files or CAL files with confidential calculations that must be actively shared with other Analyzer users to be passed on.
- If the *Calculation File Manager* is opened for a measurement from Event B, the calculation files of Event A are not visible!
- Sub-directories (<...>) can also be created in this Software directory for better organisation, but only one of these subdirectories can be displayed in the *Calculation File Manager* at a time.
- Under the placeholder <...> a user-defined folder inside <UserDataDir>\CalFiles\ can be selected with a right click on this tab

3.4.1 Explanation of the buttons

Calculate	The currently selected CAL-file is applied to the currently selected measurement
New	Create a new CAL-file in the currently selected directory
Edit	Editing the currently selected CAL-file in <i>CalcEdit.exe</i>
Delete	Delete the currently selected CAL-file

3.5 General structure of CAL-file

A CAL-file has the same file structure as a Windows INI-file and can be also seen as a text-file. It contains at least one group whose name is enclosed in square brackets. More than one group is calculated chronologically.

Then follow the calculation instructions, with each instruction starting with the channel to be created.



- Group names must be unique!



- It is very important to use groups, because after each group the respectively grabbed memory is released again when the respective calculation file is executed. If no or too few groups are used, the required memory may not be sufficient for calculating long measurements. Beside using many groups, include files (chapter 6.1.2) can be used to save memory!

Since not every channel has to be saved, a distinction is made between two types of channels:

Temporary channels

Start with a C followed by a number (e.g. C1). These channels serve as auxiliary channels within a group and are not saved. Therefore, they are only valid in the respective group.



- It is very important not to use too many temporary variables, because depending on the length of the measurement, there may not be enough memory available for the calculation. Temporary variables can also be overwritten within the group and thus used again (see example *Rear wheel spin of a bike*).

Permanent channels

Channels that do not consist of a C followed by a number (e.g. V_rear). In addition to the original recorded channels, channels created in previous groups or other CAL-files are permanent channels



- For debugging purposes, command *SaveTemporaryChannels* can be used. All temporary channels will be equipped with an underscore; thus, they are permanent channels, and will be saved (#C1 → #_C1). Use *Delete(#_C*)* for removing them.
- Within a calculation, both types of channels are called with #.
- Temporary and permanent channels can be used together in calculations as well as in calculations with constants or numbers!

Example: Rear wheel spin of a bike

Syntax	Meaning
[Spin]	; Name of group
Speed_Diff=#V_rear, #V_front)	; Subtraction of two permanent channels
C1=#Speed_Diff, #V_front)	; Division of temporary and permanent channels
C1=#C1, 100)	; Multiplication of temporary channel and a number
Result=Word(#C1, -32.768, 32.767)	

3.6 Creating permanent channels

As mentioned in the previous section, the channel to be generated permanently is at the beginning of each instruction.

A distinction is made between direct and indirect channel generation:

- Direct The channel name of the permanent channel to be created is at the beginning of an instruction
- Indirect The channel name of the permanent channel to be created is the group name in which the instruction is currently executed

Example: Rear wheel spin of a bike

Syntax	Meaning
[Spin]	; Name of group
Speed_Diff=#V_rear, #V_front)	; Direct creation of permanent channel #Speed_Diff
C1=#Speed_Diff, #V_front)	; Direct creation of temporary channel #C1
C1=#C1, 100)	; Overwriting of temporary channel #C1
Result=Word(#C1, -32.768, 32.767)	; Indirect creation of permanent channel #Spin

3.7 Replacing channels

Channels originally recorded by a 2D measuring system can only be overwritten with a special command "NewResult". For this purpose, the group in which the original channel is to be overwritten must have the name of the original channel. At the end, the special command "NewResult" must be used instead of "Result".

Example: Filtering original channel #Speed

Syntax:	Meaning
[Speed]	; Original channel to be overwritten
NewResult=F(#Speed, Med(7))	; Overwriting original channel #Speed



- The name of the group must have the name of the channel, which should be replaced.
- The original channel is still present but not visible for the user. It is possible to restore the originally recorded channel at any time (see 6.3.18.3)
- *NewResult* is only able to overwrite originally recorded channels but not CALC channels!

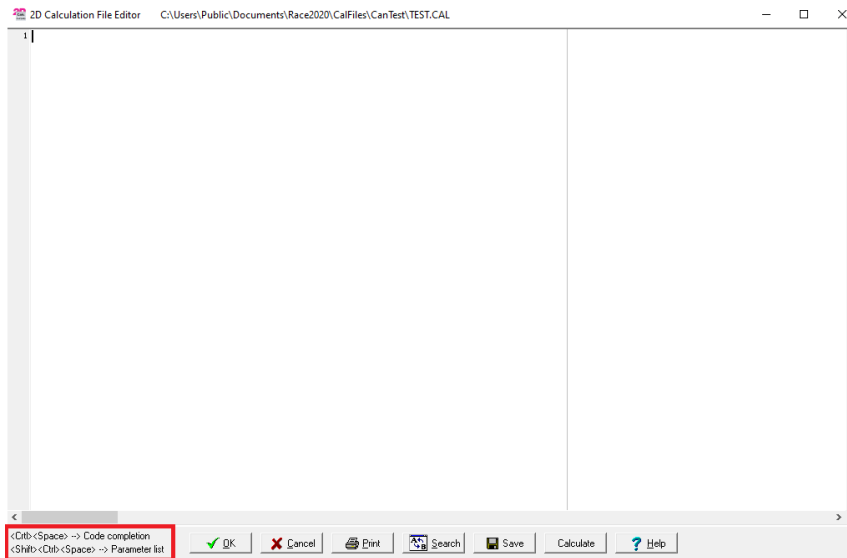
3.8 Creating constants

In addition to channels whose values usually change continuously, channels with constant values, so-called constants, can also be created with the Const function.

Syntax	Meaning
C1=Const(Value, Rate)	Creates a channel with constant value. Rate defines the sampling rate at which channel C1 is saved.

3.9 Editing assistance

3.9.1 Code completion and parameter list



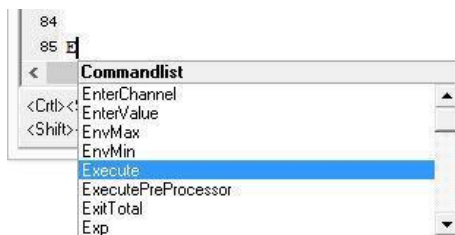
On the lower left side of the 2D-Calculation File Editor there are hints for *Code completion* and *Parameter list*.

Example: Execute-Instruction

Code completion

<CTRL><SPACE>

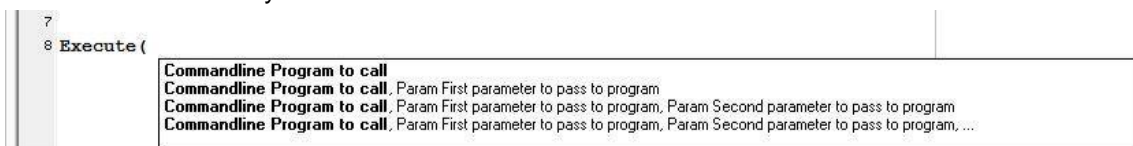
- ➔ The software will display a list of suggestion (“command list”)
- ➔ Selecting the *Execute* instruction



Parameter list

<SHIFT><CTRL> <SPACE>

- ➔ A function call is always followed by a “(” character.
- ➔ A list of parameters for the instruction *Execute* is suggested after entering the parenthesis after a short delay



3.9.2 Comments

After a command line a comment can be added with a semicolon.

3.9.3 Case-Sensitivity

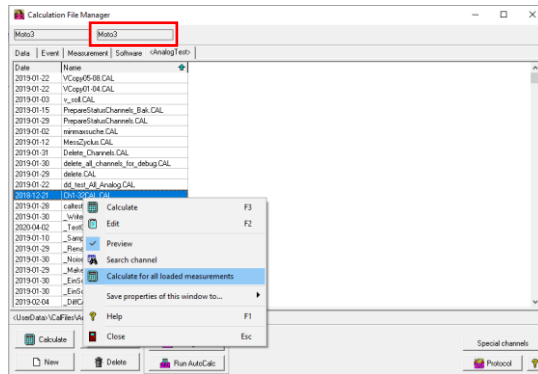
Instructions in the command lines are not case sensitive.

3.9.4 Deactivating codelines

Single lines can be deactivated by using semicolons like for comments, but for larger code blocks it is more common to use curly brackets

3.10 Executing CAL-files

3.10.1 From Calculation File Manager

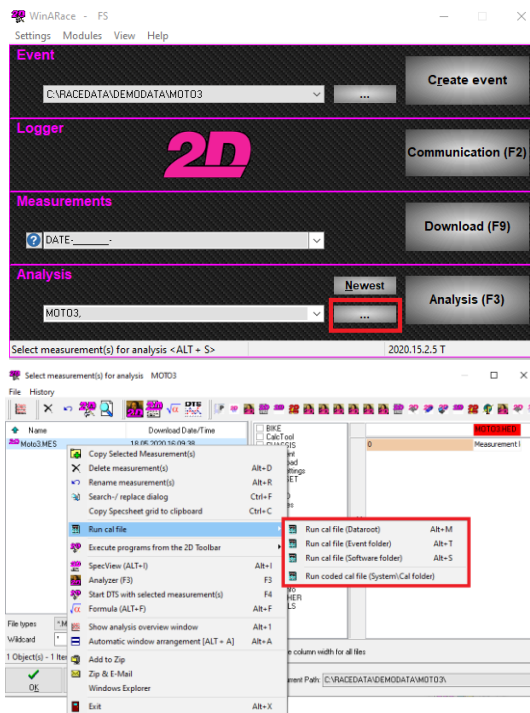


With a right click on the corresponding CAL-file it will be executed for all loaded measurements.



- If the CAL-File is executed via the button *Calculate*, it is only executed for the currently selected measurement (Currently selected measurement can be seen in red rectangular)

3.10.2 From WinARace



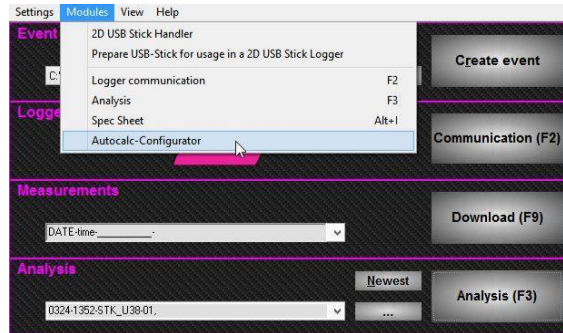
Run cal file from the Dataroot	<ALT> + <M>		
Run cal file from the current event folder	<ALT> + <T>	<EventDir>	
Run cal file from the software folder	<ALT> + <S>	<UserDataDir>\CAL	
Run cal file from "System\Cal folder".		<RaceDir>\SYSTEM\CAL	only crypted files are selectable



- Assignment of the CAL directories see 3.3
- The CAL-file is executed for all selected measurements

3.10.3 Automatic processing of the measurement data

With *AutoCalc-Configurator* you can define which CAL-files will be executed automatically after downloading the data from logger. Inside the *Autocalc-Configurator* the user defines which CAL-file(s) should be executed automatically after the download:



1st step: Selected 2D factory calculation files

Choose from a predefined list of delivered 2D factory calculation files. Their file names always start with "2D_". After "2D_", the next 3 characters are defining a group. You can only select one file within a group. For example, you can select either "2D_GPS8Hz.ccf" or "2D_GPS4Hz.ccf" or "2D_GPSKit.ccf", but not 2 or 3 of them.

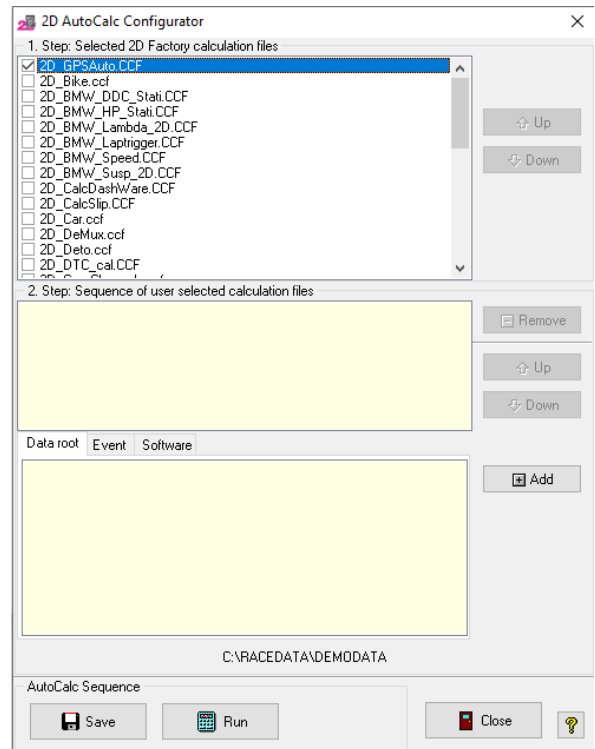


- Only one CCF can be selected within a group!
- The factory calculation files are encrypted and can't be modified or read by the user (see 3.1)!

2nd step: Sequence of user selected calculation files

User calculation files can be added from three different directories (see 3.3):

- Data root
- Event
- Software



By clicking the button **<Add>**, the CAL-file is added to the calculation sequence. **<Up>** and **<Down>** can be used to change the order of the user calculation sequence.

Entries of the user calculation sequence can be removed by clicking on the **<Remove>** button.

If you want to execute the list of automatically calculated CAL-files independent from downloading data, you can push the button **<Run>** inside the *AutoCalc-Configurator*.



- Save the sequence and optionally run it to check proper calculation
- 2D AutoCalc Configurator can also be called from the *Calculation File Manager*

4 SpecSheet

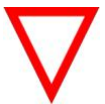
The *SpecSheet* is a text-based HED-file, which can be edited by an Editor.

Users can create own *SpecSheets* by using an Editor and thus can pre-define a specified *SpecSheet*. Especially in motorsports, *SpecSheets* are used to store various information about vehicle setup, weather, or drivers for example during a race weekend or testing. These files are generated for every download (=reading the data from the data logger) in measurement folder and in logger communication mode it can be chosen in which *SpecSheet* the information should be written at measurement download automatically. Therefore, there is a separate *SpecSheet* with measurement related information for each measurement in measurement folder. The user of the program – the engineer, the data recording man, or the mechanic – is in charge of the maintenance and integrity of the data.

Mostly this *SpecSheet* is only used for documentation, but it can also be used in connection with the *CalcTool* because the *CalcTool* can read from information as constant from the *SpecSheet* and write information back into it.



- For more information about *SpecSheet* please see the manual “SpecView” on the following link:
<http://2d-datarecording.com/downloads/manuals/>



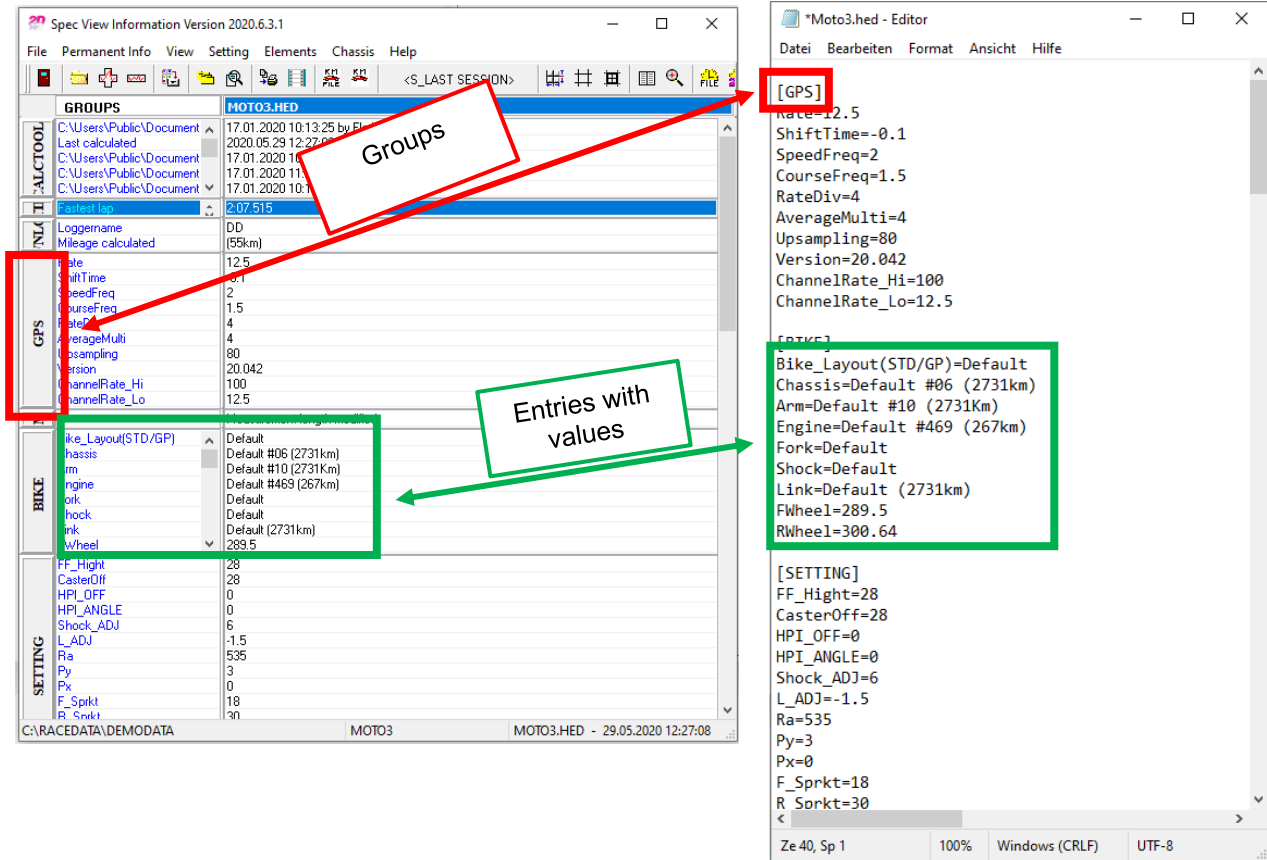
- From CAL-files, HED-files can be read and write accessed!



- The respective HED-file can always be found in the respective measurement folder
- Via *WinARace* and *Analyzer* the *SpecSheet* can be opened in the *SpecView* with short-cut <ALT>+<i>
- The *SpecSheet* is linked automatically to CAL-files
- Beside *SpecSheet* other HED-files can be linked to calculation files too

4.1 Structure

Display of the same *SpecSheet* in *SpecView* (left) and in *Editor* (right)



SpecView Information Version 2020.6.3.1

GROUPS	MOTO3.HED
C:\Users\Public\Document	17.01.2020 10:13:25 by E...
Last calculated	2020.05.29 12:27:08
C:\Users\Public\Document	17.01.2020 10:13:25
C:\Users\Public\Document	17.01.2020 11:00:00
C:\Users\Public\Document	17.01.2020 10:13:25
Fastest lap	2:07.515
Loggename	DD
Mileage-calculated	(56km)
Rate	12.5
ShiftTime	-0.1
SpeedFreq	2
CourseFreq	1.5
RateDiv	4
AverageMulti	4
Upsampling	80
Version	20.042
ChannelRate_Hi	100
ChannelRate_Lo	12.5
Bike_Layout(STD/GP)	Default
Chassis	Default #06 (2731km)
Arm	Default #10 (2731km)
Engine	Default #469 (267km)
Fork	Default
Shock	Default
Link	Default (2731km)
FWheel	289.5
RWheel	300.64
FF_Hight	28
CasterOff	28
HPI_OFF	0
HPI_ANGLE	0
Shock_ADJ	6
L_ADJ	-1.5
Ra	535
Py	3
Px	0
F_Sprkt	18
R_Sprkt	30

***Moto3.hed - Editor**

```

[GPS]
Rate=12.5
ShiftTime=-0.1
SpeedFreq=2
CourseFreq=1.5
RateDiv=4
AverageMulti=4
Upsampling=80
Version=20.042
ChannelRate_Hi=100
ChannelRate_Lo=12.5

[BIKE]
Bike_Layout(STD/GP)=Default
Chassis=Default #06 (2731km)
Arm=Default #10 (2731km)
Engine=Default #469 (267km)
Fork=Default
Shock=Default
Link=Default (2731km)
FWheel=289.5
RWheel=300.64

[SETTING]
FF_Hight=28
CasterOff=28
HPI_OFF=0
HPI_ANGLE=0
Shock_ADJ=6
L_ADJ=-1.5
Ra=535
Py=3
Px=0
F_Sprkt=18
R_Sprkt=30
    
```

4.2 Read access

CalcTool can read from information from *SpecSheet* as constants.

Syntax	Meaning
Group.Entry	Retrieves the complete value of the entry "Entry" in the group "Group"
Group.Entry.%N	Retrieves the Nth part of the value of the entry "Entry" in the group "Group". The part must be separated by a space character.

Example:

It is assumed that the value of the entry "FWheel" in the group "BIKE" is "X-Z 439"

Call	Result
C1=Const(Bike.FWheel , Rate)	X-Z 439 Not usable as CalcTool-constant (CalcTool ERROR)
C1=Const(Bike.FWheel.%1, Rate)	X-Z Not usable as CalcTool-constant (CalcTool ERROR)
C1=Const(Bike.FWheel.%2, Rate)	439 Usable as CalcTool-constant



- Be careful with dimensions when reading values from *SpecSheet*!

4.3 Write access

Besides read-access, values can also be written back to the *SpecSheet*. This functionality can be used for documentation and testing purposes. Since a *SpecSheet* entry consists of only one value, only certain commands can be used to write to the *SpecSheet*:

Syntax	Meaning
Group.Entry =LastValue(#CH)	Last value of channel (6.3.14.7)
Group.Entry =FirstValue(#CH)	First value of channel (6.3.14.6)
Group.Entry =MaxValue(#CH)	Maximal value of channel (0)
Group.Entry =MinValue(#CH)	Minimal value of channel (6.3.14.4.1)
Group.Entry =AvgValue(#CH)	Average value of channel (6.3.14.8)



- By adding p_File or pFile to Syntax, values can also be written to respective permanent *SpecSheet* file of the logger which was used for recording the current measurement.

→ pFile.Group.Entry=MaxValue(#CH)

5 Phases

In the analyzing software *2D Analyzer* it is necessary to reduce the volume of the data to make special analysis (for example MinMax-tables, XY-plots, etc.). Often not all data are interesting, but the fastest lap, a special corner or when the values of a channel are in a special range. The reduction of the data is defined by so called phase-conditions. It is possible to combine several phase-conditions. Phases are measurement parts where the phase-conditions meet.



- For more information about Phases please see the manual "Phases" on the following link:

<http://2d-datarecording.com/downloads/manuals/>

6 Pre- and Postprocessor

Besides the calculation instructions CalcTool provides a set of functions which are performed before or at the main calculation. These functions are executed either by the pre-processor (function before the main calculation) or the postprocessor (at the main calculation).

This section describes the use of the pre- and postprocessor.

6.1 Pre-processor functions

6.1.1	ExecutePreprocessor	17
6.1.2	Include files	18
6.1.3	Conditional execution of CAL-file groups	23
6.1.4	Substitution of place holders	27
6.1.5	Mileage	32
6.1.6	Format of numeric parameters	32
6.1.7	User interaction	33

6.1.1 ExecutePreprocessor

The pre-processor functions are executed before the calculation of the channels.

The execution of the Pre-processor before executing the calculation file runs automatically. In the code, you can use the command ExecutePreprocessor to force execution of the Pre-processor

Syntax	Meaning
ExecutePreprocessor	Forcing Pre-Processor executing at respective point in code.



- Forcing Pre-processor execution is necessary if e.g. a SpecSheet entry was written in a previous group of the same calculation file is used later again

6.1.2 Include files

In the CalcTool there are several possibilities to include other files in a calculation.

It is possible to include the following types of files:

- Parameter files (.HED).
- Variable files (.Var)
- Calculation files (.CAL/.CCF)



- Variable files are included at the beginning of a calculation file!
- Calculation files are included within groups!



- In chapter 3.3 an overview over different path-placeholders can be found which can replace ... in the path calls like ...\FileName.VAR

6.1.2.1 Parameter files

Only the measurement related parameter file (.HED), which are located in the respective measurement directory can be accessed from calculation files!



- For more information about parameter files (.HED) please see chapter 4.

6.1.2.2 Variable files

For including variable files (.VAR), the following call is valid.

Syntax	Meaning
{\$V ...\FileName.VAR}	Including variable file



- If there is no path defined at a VAR-File include beginning of the CAL-file, the VAR-File is searched inside Race-installation folder!
- For more information about VAR-file see 6.1.4.4.2.



- The VAR-file include call must be placed at the beginning of a calculation file!
- If more than one VAR-files are included at the beginning of a calculation file, only the first include call is considered!

6.1.2.3 Calculation files

Include calculation files are very suitable for outsourcing frequently used routines and thus making the calculation file clearer.

Since parameters and channels can also be passed to the include files, generally valid functions for several purposes can also be used as include files. Include calls can also be linked to a condition.

6.1.2.3.1 File extension

Syntax	Meaning
{\$! ...\FileName.CAL}	Including encrypted calculation file
{\$! ...\FileName.CCF}	Including crypted calculation file

- If no extension, e.g. ...\FileName, is available in the include call, first a CAL file is searched for
- If no CAL could be found in respective directory, it is searched for a CCF
- If no CCF is found in respective directory, an error message appears



- This approach applies to both unconditional and conditional inclusions!

6.1.2.3.2 Path definitions



- This approach applies to both crypted and encrypted calculation files

If there is no path defined at a calculation file include call, e.g. {\$! FileName.CAL}, the calculation file to be included is searched for in the directory, the calculation file with include call is located in! If a calculation file from another directory must be included, the file-path must be defined at include call.

Only since *Race2021* the path of third stage include call (CAL3 → see following **Example: Race2021**) no longer needs to be defined separately, because the path will of CAL3 is traced back to the include call of CAL2.



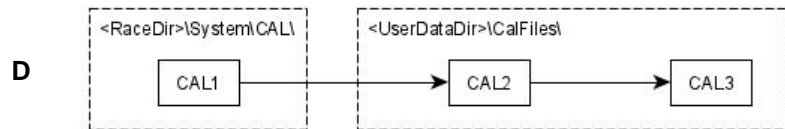
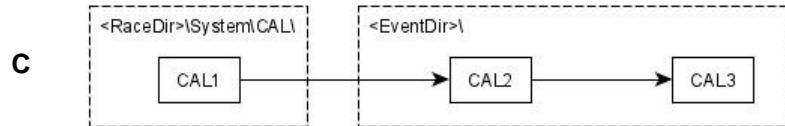
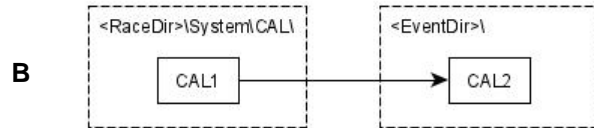
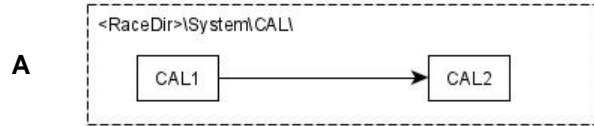
- The advantage of this is that when moving CAL2 and CAL3 to another directory, only the include path of CAL2 in include call at CAL1 needs to be changed. (**Example: Race2021 C & D**)

In older Race-versions, the path of CAL3 must also be defined for the include call in CAL2, because if no path is defined for the include call in CAL2, the system will search in the directory of CAL1 for CAL3 (**Example: Race2020 and older versions A**).



- Keep in mind that include-call of CAL3 in CAL2 must be changed if CAL2 and CAL3 are moved to another directory!

Example: Race2021



Include-call in CAL1

`{ $I CAL2, P () }`

`{ $I <EventDir>\CAL2, P () }`

`{ $I <EventDir>\CAL2, P () }`

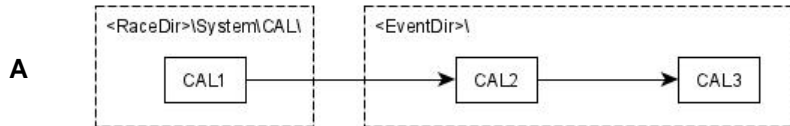
`{ $I <UserDataDir>\CalFiles\CAL3, P () }`

Include-call in CAL2

`{ $I CAL3, P () }`

`{ $I CAL3, P () }`

Example: Race2020 and older



Include-call in CAL1

`{ $I <EventDir>\CAL2, P () }`

Include-call in CAL2

`{ $I <EventDir>\CAL3, P () }`


6.1.2.3.1 Unconditional include

For including calculation files without any conditions, the following calls are valid:

Syntax	Meaning
{\$! ...\FileName.CAL}	Including encrypted calculation file
{\$! ...\FileName.CCF}	Including crypted calculation file

6.1.2.3.2 Conditional include

For including a calculation files only if one conditions is fulfilled, the following calls are valid:

-  - Conditional include commands are only executed if the respective condition is fulfilled!
- Conditional include is possible for crypted and encrypted calculation files!

The following conditional includes are possible:	
{\$IfExists(#CH) ...\FileName.CAL, P(P1, P2, ...)}	Only include if channel #CH available
{\$IfNotExists(#CH) ...\FileName.CAL, P(P1, P2, ...)}	Only include if channel #CH not available
{\$IfLaptimesExist ...\FileName.CAL, P(P1, P2, ...)}	Only include if laptimes exists
{\$IfNotLaptimesExist ...\FileName.CAL, P(P1, P2, ...)}	Only include if laptimes exists
{\$IfLicenceNameContains('Name') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry INFO_2D.LicenseName contains entry 'Name'
{\$IfNotLicenceNameContains('Name') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry INFO_2D.LicenseName contains entry 'Name'
{\$IfSpecValueContains(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry not contains entry 'Name'
{\$IfNotSpecValueContains(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry not contains 'Value'
{\$IfSpecIs(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry is 'Value'
{\$IfNotSpecIs(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry not is 'Value'
{\$IfSpecValueExists(Group.Entry) ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry exists
{\$IfNotSpecValueExists(Group.Entry) ...\FileName.CAL, P(P1, P2, ...)}	Only include if Spec-Entry Group.Entry not exists

6.1.2.3.3 Parameter handover

To be able to call a CAL file several times with different parameters, the parameter handover of the include command is used

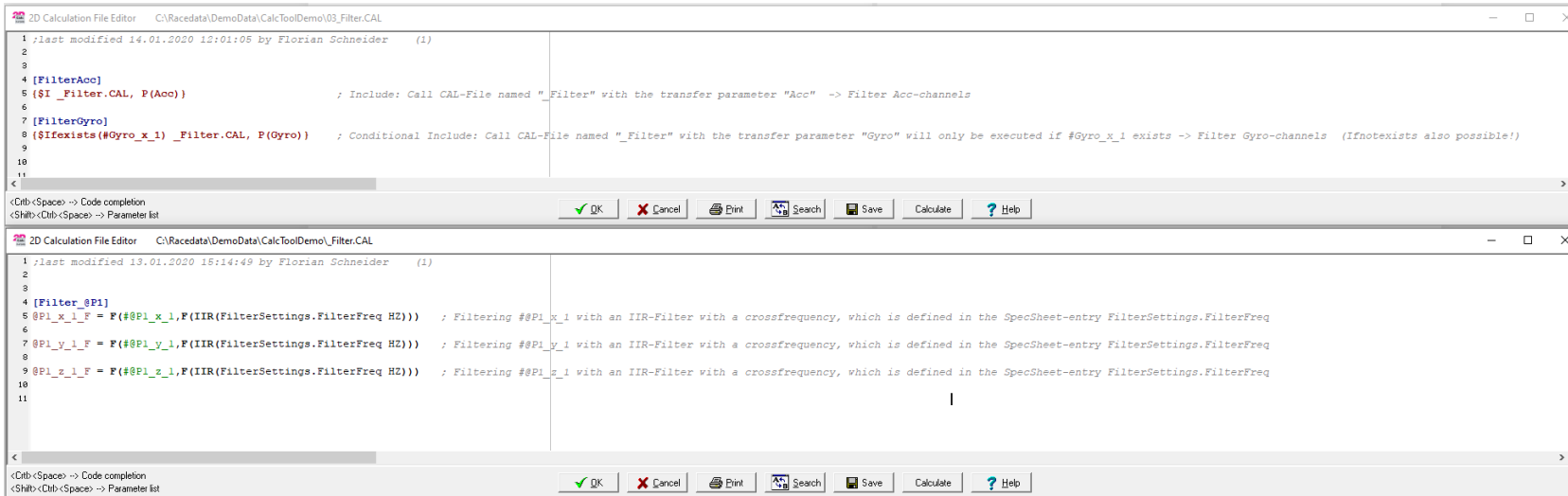
- Conditional and unconditional includes both also provide the possibility for parameter handover!

For including calculation files with parameter handover, the following calls are valid.

Syntax	Meaning
{\$! ... \FileName.CAL, P(p1, p2,...)}	Unconditional include of encrypted calculation file with parameter handover
{\$IfExists(#CH) ... \FileName.CAL, P(p1, p2,...)}	Conditional include of encrypted calculation file with parameter handover
{\$! ... \FileName.CCF, P(p1, p2,...)}	Unconditional include of crypted calculation file with parameter handover
{\$IfExists(#CH) ... \FileName.CCF, P(p1, p2,...)}	Conditional include of crypted calculation file with parameter handover

- In the included calculation file the link to the handover parameters of the include file is established via the calls @P1, @P2, @...
- Parameter-handover is also possible with all other conditions!

Example:



The screenshot shows two instances of the '2D Calculation File Editor' window. The top window displays a CAL file with the following code:

```

1 ;last modified 14.01.2020 12:01:05 by Florian Schneider (1)
2
3
4 [FilterAcc]
5 {$I _Filter.CAL, P(Acc)} ; Include: Call CAL-File named "_Filter" with the transfer parameter "Acc" -> Filter Acc-channels
6
7 [FilterGyro]
8 {$IfExists(#Gyro_x_1) _Filter.CAL, P(Gyro)} ; Conditional Include: Call CAL-File named "_Filter" with the transfer parameter "Gyro" will only be executed if #Gyro_x_1 exists -> Filter Gyro-channels (Ifnotexists also possible!)
9
10
11
    
```

The bottom window displays a CAL file with the following code:

```

1 ;last modified 13.01.2020 15:14:49 by Florian Schneider (1)
2
3
4 [Filter_@P1]
5 @P1_x_1_F = F(##P1_x_1,F(IIR(FilterSettings.FilterFreq HZ))) ; Filtering ##P1_x_1 with an IIR-Filter with a crossfrequency, which is defined in the SpecSheet-entry FilterSettings.FilterFreq
6
7 @P1_y_1_F = F(##P1_y_1,F(IIR(FilterSettings.FilterFreq HZ))) ; Filtering ##P1_y_1 with an IIR-Filter with a crossfrequency, which is defined in the SpecSheet-entry FilterSettings.FilterFreq
8
9 @P1_z_1_F = F(##P1_z_1,F(IIR(FilterSettings.FilterFreq HZ))) ; Filtering ##P1_z_1 with an IIR-Filter with a crossfrequency, which is defined in the SpecSheet-entry FilterSettings.FilterFreq
10
11
    
```

6.1.3 Conditional execution of CAL-file groups

In some cases, it is useful to execute a group in calculation file only if a certain channel or *SpecSheet* value exists (or if it doesn't exist). For this reason, *CalcTool* provides the conditional execution of calculation groups.



- It is possible to use more than one condition for a group, but the condition lines must be one after the other at the start of the group:

```
[Group1]
IfExists(#CH1)
IfNotExists(#CH2)
```

6.1.3.1 IfLaptimesExist

With this condition it is possible to calculate a single group only if the measurement contains lap times.

Syntax	Meaning
[MyCalc]	Start of group
IfLaptimesExist	Condition
Result = ...	This calculation is only executed if the measurement contains lap times.

6.1.3.2 IfNotLaptimesExist

This condition is the opposite of the one above. A single calculation group is calculated only if the measurement does not contain lap times.

Syntax	Meaning
[MyCalc]	Start of group
IfNotLaptimesExist	Condition
Result = ...	This calculation is only executed if the measurement does not contain lap times

6.1.3.3 IfExists

With this condition it is possible to calculate a single group only if a respective channel exists.

Syntax	Meaning
[MyCalc]	Start of group
IfExists(#CH)	Condition
Result = ...	This calculation is only executed if #CH exists.

6.1.3.4 IfNotExists

This condition is the opposite of the one above. A single calculation group is performed only if a respective channel doesn't exist.

Syntax	Meaning
[MyCalc]	Start of group
IfNotExists(#CH)	Condition
Result = ...	This calculation is only done if #CH does not exist.

6.1.3.5 IfOrgExists

With this condition it is possible to calculate a single group only if a respective **originally recorded** channel exists.

Syntax	Meaning
['Ch']	Start of group
IfOrgExists(#CH)	Condition
NewResult = ...	This calculation is only executed if #CH exists as originally recorded channel.



- *IfOrgExists* will most likely be used in combination with *NewResult* (3.7) to check if an originally recorded channel is available. *NewResult* is only able to overwrite originally recorded channels with the group name ['Ch'] but not CALC channels!

6.1.3.6 IfNotOrgExists

This condition is the opposite of the one above. A single calculation group is performed only if a respective **originally recorded** channel doesn't exist.

Syntax	Meaning
[MyCalc]	Start of group
IfNotOrgExists(#CH)	Condition
'Ch' = ...	This calculation is only executed if #CH does not exist as originally recorded channel.



- *IfNotOrgExists* will most likely be used in combination with *creating new permanent channels* (3.6) when no **originally recorded** channel is available.
- Please see the following example why it is important to use 'Ch'

Example IfOrgExists/IfNotOrgExists

Especially GPS channels *Altitude* and *Course* are at different 2D-GPS-Systems sometimes available as originally recorded channels or generated CALC channels. To handle the different options in calculation files the *IfOrgExists/IfNotOrgExists* were created

[Course]	[Course]
IfOrgExists (#Course)	IfOrgExists (#Course)
NewResult=Noop(#Course)	NewResult=Noop(#Course)
[Course]	[Course_NotExists]
IfNotOrgExists (#Course)	IfNotOrgExists (#Course)
Result=Noop(#Course)	Course=Noop(#Course)
With NewResult/Result an error will occur because same group names [Course] are used!	To avoid the adjacent problem, the second IfNotOrgExists-group will be called [Course_NotExists] to avoid same channel name and instead of creating channel by result, the channel Course is created in line with its name.

6.1.3.7 IfSpecValueExists

With this condition it is possible to calculate a single group only if the *SpecSheet* of the measurement contains a certain entry.

Syntax	Meaning
[MyCalc]	Start of group
IfSpecValueExists(Group.Entry)	Condition
Result = ...	This calculation is only done if the <i>SpecSheet</i> of the measurement contains group "Group" with the entry "Entry".

6.1.3.8 IfNotSpecValueExists

This condition is the opposite of the one above. A single calculation group is calculated only if the *SpecSheet* of the measurement does not contain the entry "Entry" in the group "Group".

Syntax	Meaning
[MyCalc]	Start of group
IfNotSpecValueExists(Group.Entry)	Condition
Result = ...	This calculation is only done if the Spec Sheet of the measurement does not contain an entry "Entry" in the group

6.1.3.9 IfSpecValueContains

With this condition the user has possibility to calculate a single group only if the *SpecSheet* of the measurement contains a certain entry and if the entry contains a certain string (SearchStr).

Syntax	Meaning
[MyCalc]	Start of group
IfSpecValueContains(Group.Entry, SearchStr)	Condition
Result = ...	This calculation is only executed if the <i>SpecSheet</i> of the measurement contains group "Group" with the entry "Entry" and the entry contains the searched string.

6.1.3.10 IfNotSpecValueContains

This condition is the opposite of the one above. A single calculation group is only calculated if the *SpecSheet* of the measurement does not contain the entry "Entry" in the group "Group" or if the entry exists but does not contain a certain string (SearchStr).

Syntax	Meaning
[MyCalc]	Start of group
IfNotSpecValueContains(Group.Entry, SearchStr)	Condition
Result = ...	This calculation is only executed if the <i>SpecSheet</i> of the measurement does not contain an entry "Entry" in the group "Group" or the entry exists but does not contain a certain searched string.

6.1.3.11 IfLicenceNameContains

With this condition it is possible to calculate a single group only if the *License Name* of the software contains a certain string. Normally this is only used in encrypted CAL-files prepared by 2D for a specific customer.

Syntax	Meaning
[MyCalc]	Start of group
IfLicenceNameContains(LicenceName)	Condition
Result = ...	This calculation is only executed if the <i>License Name</i> of the software contains the searched string.

6.1.3.12 IfNotLicenceNameContains

This condition is the opposite of the one above. A single calculation group is only calculated if the *License Name* of the software does not contain a certain string.

Syntax	Meaning
[MyCalc]	Start of group
IfNotLicenceNameContains(LicenceName)	Condition
Result = ...	This calculation is only executed if the <i>License Name</i> of the software does not contain the searched string.

6.1.3.13 IfSpecValues

The calculation group is only calculated if a SpecSheet-entry is a certain value.

Syntax	Meaning
[MyCalc]	Start of group
IfSpecValues(Group.Entry, SearchValue)	Condition
Result = ...	This calculation is only executed if the <i>Group.Entry</i> is the SearchValue

6.1.3.14 IfNotSpecValues

The calculation group is only calculated if a SpecSheet-entry is not a certain value.

Syntax	Meaning
[MyCalc]	Start of group
IfNotSpecValues(Group.Entry, SearchValue)	Condition
Result = ...	This calculation is only executed if the <i>Group.Entry</i> is not the SearchValue

6.1.4 Substitution of place holders



- A place holder always starts with the “@” character

The following for four kinds of place holders can be used for substitution:

- Special channels (see 6.1.4.1)
- Pre-defined constants (see 6.1.4.2)
- *SpecSheet*-Values (see 6.1.4.3)
- Local and global variables (see 6.1.4.4)



- When executing a CAL file, the Preprocessor replaces all found special character with the respective parameters/channels

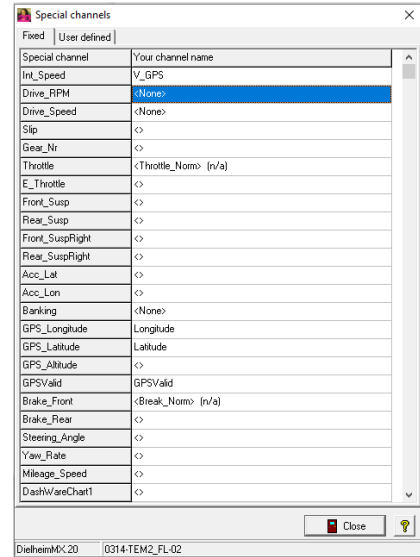
6.1.4.1 Special channels

In the *Analyzer*, special channels can be defined.

The advantage is that flexibility is increased when working with measurements where different channel names have been assigned to channels with the same purpose. A calculation file with fixed channel names could only be applied to certain measurements!

With the special channels this problem is solved because the respective special channels inside the calculation files are replaced by the channels linked in the adjoining list by the Preprocessor.

In the *Analyzer*, the adjacent *Special channels* list can be opened under the tab *Settings*.



- Also <SHIFT> + <s> can be used.

In this list the user can define the special channels for the measurement currently selected in the *Analyzer*. In addition to defined special channels, the user can also assign his own special channels.



- The list can also be edited in the *Event.ini*
- In Tab *User defined*, own special channels can be defined!

There are three different possibilities to select at special channels list:

<>	Default setting when no channel is selected	This option cannot be chosen by the user. When a special channel is used in calculation file, and this option is used, the <i>CalcTool</i> will bring up an error, that no channel is selected at the respective special channel. → Conditional execution leads to an error!
<None>	No channel can be linked to respective special channel	This option is selected by the user when no channel can be linked to the respective special channel. When a special channel is used in calculation file, and this option is used, the <i>CalcTool</i> will bring up no error, that no channel is selected at the respective special channel. → Conditional execution leads to result FALSE!
V_GPS	Channel linked to respective special channel	This option is selected by the user when channel can be linked to the respective special channel. → Conditional execution leads to result TRUE!



- If a channel is selected, but currently not present in measurement, the special channel is automatically marked with (n/a). [Conditional execution](#) leads to result FALSE!

Example:

Front_Susp	SUS_F	Channel #SUS_F is linked to special channel @Front_Susp
C1 = F(@Front_Susp, F(L_PHAMM3H277))		Special Channel @Front_Susp is called in CAL-file
Replaced @Front_Susp by #SUS_F Computing line: C1=F[#SUS_F, F(L_PHAMM3H277)]		Preprocessor replaces Special channel @Front_Susp with linked channel #SUS_F

Frequently used special channels with description

Special channel	Description
Int_Speed	Speed channel used for integration of speed
Drive_RPM	Engine RPM
Drive_Speed	Engine speed (for gear calculation when no Gear_Nr)
Slip	Slip rear wheel
Gear_Nr	Gear number
Throttle	Throttle-grip signal
Throttle_E	Throttle butterfly position (drive by wire)
Front_Susp	Signal of front suspension (Cars: front left)
Rear_Susp	Signal of rear suspension (Cars: rear left)
Front_SuspRight	Signal of front suspension (Cars: front right)
Rear_SuspRight	Signal of rear suspension (Cars: rear right)
Acc_Lat	Lateral acceleration
Acc_Lon	Longitudinal acceleration
Banking	Banking of motorcycle
GPS_Longitude	Longitudinal position
GPS_Latitude	Latitudinal position
GPS_Altitude	Altitude
GPSValid	Valid GPS-Signal
Brake_Front	Pressure front brake
Brake_Rear	Pressure rear brake
Steering_Angle	Position signal of steering wheel
Yaw_Rate	Yaw-Rate of vehicle
Mileage_Speed	Speed channel used for determining the mileage

 6.1.4.2 Predefined constants

The following constants can be used in calculation files:

Constant	Description
@PI	3.145...
@2PI	6.283...
@PI/2	1.571...
@E	2.71828 (Euler constant)
@G	9.80665 (Gravity constant)
@LN2	Natural logarithm of 2
@RAD2DEG	180/PI constant to calculate radiant to degrees
@DEG2RAD	PI/180 constant to calculate degrees to radiant

Others	Description
@MainSamplingRate	Mainsamplingrate of measurement

6.1.4.3 SpecSheet variables

How to use SpecSheet-Values as constants please see 4.2.

6.1.4.4 Variables

In *CalcTool* it is also possible to use global and locale variables.



- Either global or local variables can be used!

6.1.4.4.1 Local variables

Local variables are defined inside a calculation file in a group called [Variables]. This group is located before each other calculation groups in the file.

Syntax	Meaning
[Variables]	Start of group
Variable1=815	Defining Variable 1
Variable2=666	Defining Variable 2



- The call of the variables in the following code is again done via the character "@"

6.1.4.4.2 Global variables

Global variables must be defined in standalone VAR files. This VAR file can be created with a text editor.

Within this file the variable definition is done in the same way as for local variables via the group [Variables].

Syntax	Meaning
{ \$V ... \Filename.VAR }	Include VAR-file



- The Filename must contain the full path information!
- The created VAR-file must first be included before the variables can be used in the actual calculation groups!
- A VAR-file always contains only on group [Variables]
- The VAR-file entries can be accessed in CAL-file by using @



- For the path of file, placeholders can be used (see 3.3)!
- Another predefined filename placeholder could be used only at global variables to load a variable file depending on the name of the current logger:
<LoggerName> will be replaced by the name of the logger with which the measurement was recorded → { \$V ... <LoggerName> .VAR }

Example: Two-stage placeholder replacement

How to recalculate the speed channel of a bike if the circumference of the tire was not set correctly before the measurement.

- Use SpecSheet Value
- Use global variables
- Create new channel
- Overwrite originally recorded channel

SpecSheet:
[Tire] Front=Soft05

Tires.VAR:
[Variables] Soft05 = 1885 Hard05 = 1885 Another = 1885 M4124C = 1887 DefaultCircumference=1890

Syntax	Meaning
{\$V <EventDir>\Tires.VAR}	; Include Tires.VAR from Event directory
[SpeedRecalc]	; Defining group name
C1=#VFront, @DefaultCircumference)	; @DefaultCircumference" is replaced by Tires.VAR-Value 1890
C2=#C1, @Tire.Front)	; Division of #V_Front and 1890 ; @Tire.Front → @Soft05 → 1885 ; SpecValue → Tires.VAR → Value
Result=WORD(#C2)	; The resulting-channel C2 receives the product ; of multiplication of C1 and the new circumference
[VFront]	; Creating new channel #SpeedRecalc defined by ; name of group in word-format
IfExists(#SpeedRecalc)	; Defining group name ; Execute group [VFront] only if channel ; #SpeedRecalc exists
NewResult= Word(#SpeedRecalc, 0, 500)	; #SpeedRecalc exists ; Convert the recalculated speed channel to word ; using the borders 0 and 500 km/h ; If the keyword "NewResult" is used, the ; calculated channel will replace an originally ; recorded channel.

Two-stage placeholder replacement:

The PreProcessor first replaces SpecSheet-Placeholder Tire.Front by Soft05. Therefore, the placeholder @Soft05 is created which is then also converted by the Preprocessor to the value 1885.

```
11:50:32 Executing preprocessor on cal-file.
11:50:32 Using variable-file: C:\Racedata\DemoData\Moto3\Tires.var
2 *: 11:50:32 Replaced variable @DefaultCircumference by value: 1890
2 *: 11:50:32 Replaced Tire.Front by @Soft05
2 *: 11:50:32 Replaced variable @Soft05 by value: 1885
11:50:32 Preprocessor finished.
```

6.1.4.5 Channel properties

In some cases, it is useful to use channel properties like sampling rate for calculation. These properties are replaced by their corresponding values before the calculation is started.

CalcTool allows to use the following channel properties:

Syntax	Meaning
#CH.Rate	Sampling rate of the channel (i.e. 100 for a channel sampled with 100 Hz)
#CH.DT	Sampling time of the channel (i.e. 0.01 for a 100 Hz channel)
#CH.P1	First of the three parameters of channel. Meaning depends on channel type
#CH.P2	Second of the three parameters of channel. Meaning depends on channel type
#CH.P3	Third of the three parameters of channel. Meaning depends on channel type



- Parameters P1, P2 and P3 are only for 2D-internal use!

6.1.5 Mileage

The 2D software structure makes it possible to record and document in the respective *SpecSheet* which mileage in kilometres various components have completed on the vehicle.



- This functionality can be used for planning the maintenance and replacement intervals of components!



- For more information about *Mileage*-function please see the manual *Mileage Function* on www.2d-datarecording.com/Downloads

6.1.6 Format of numeric parameters

Some functions of *CalcTool* require one or more numbers as parameter. There are three different possibilities to write a numeric parameter:

Decimal	The number is written using the usual syntax, i.e. 350 or 17.5. The decimal separator is independent of your computer's language settings. Always use a <u>point</u> as decimal separator since the colon is used as parameter separator.
Hexadecimal	This notation starts with "0x" or "0h" and is followed by hexadecimal digits (0...F); i.e. 0xFF or 0hFF for the decimal value of 255. Only integer numbers can be written in this notation
Binary	This notation starts with "0b" and is followed by binary digits (0 and 1) i.e. 0b11111111 for the decimal value of 255. Only integer numbers can be written in this notation.

6.1.7 User interaction

To make calculation files more flexible, CalcTool provides two functions to enable the user to enter data.

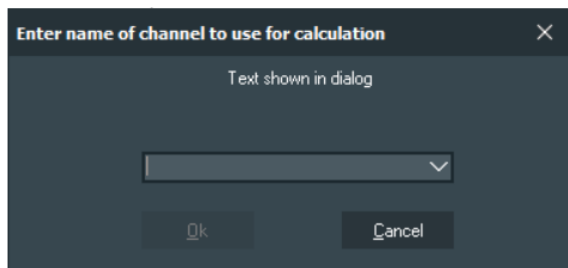


- Commands are executed by the Preprocessor

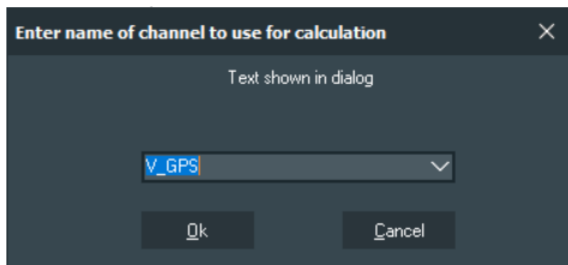
6.1.7.1 Entering a channel name

This function opens a dialog to ask the user for the name of a channel to use in further calculation. The user may choose from a list of channels available in the specific measurement.

Syntax	Meaning
C1=EnterChannel('Text shown in dialog')	Opens a dialog to ask the user for a channel name. The text in the parameter is shown in the dialog.



Syntax	Meaning
C1=EnterChannel('Text Text shown in dialog', #CH)	Like above, but with a preselected channel



Example:

The user is queried at execution of respective CAL-file to select a Video_File_Index-Channel for Dashware-export. #VideoFile_Index_1 is preselected.

```
[Dialog_Camera]
Camera = EnterChannel('Select Camera for Dashware', #Video_file_index_1)
```



- The created channel (e.g. Camera) can be used like any other channel in following calculations

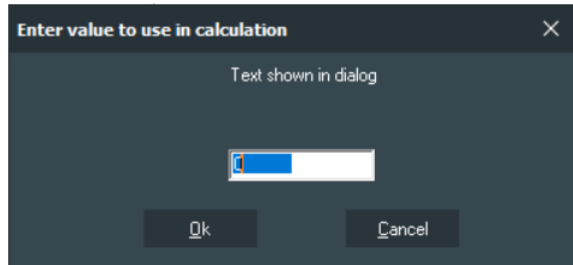


- When the user cancels the dialog, the calculation will stop at this line. Channels which were calculated in lines before will be stored, lines and groups after this line will be ignored.

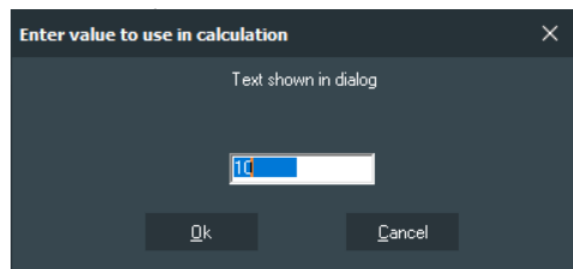
6.1.7.2 Entering a numeric value

This function opens a dialog asking the user to enter a numeric value to use in further calculation

Syntax	Meaning
C1=EnterValue('Text shown in dialog')	Opens a dialog to ask the user for a numeric value. The text in the parameter is shown in the dialog. By default, value 0 is set.



Syntax	Meaning
C1=EnterValue('Text shown in dialog', Value)	Like above, but with a preselected value.



Example:

The user is queried to select, which video should be used for Dashware-export. Value 1 is preselected.

```
[Dialog_Video]
C0      = EnterValue('Select video for Dashware', 1)
Video   = Const(@C0, 1)
```



- The dialog shows the text parameter of the function. The entered value is stored in a constant with the name which is given on the left side of the “=”.
- The constant must be called in the following calculations with character “@” (e.g. @C0)



- When the user cancels the dialog without entering a value, the calculation will stop at this line. Channels which were calculated in lines before will be stored, lines and groups after this line will be ignored

6.3 Postprocessor-functions

6.3.1	Additional formulas	36
6.3.3	ChannelArray	37
6.3.4	Rotational correction of channels	37
6.3.5	Madgwick_IMU_AHRS	39
6.3.6	Filter function	40
6.3.7	Frequency adjustments	45
6.3.8	GPS commands	46
6.3.9	Channel manipulation	48
6.3.10	If function	52
6.3.11	SOD	52
6.3.12	Mathematical functions	53
6.3.13	Logical functions	61
6.3.14	Signal analysis	62
6.3.15	Table functions	84
6.3.16	Changing the storage type	89
6.3.17	Additional channel information	92
6.3.18	Channel handling	93
6.3.19	Laps	94
6.3.20	Sections	95
6.3.21	Bike and car physical formulas	96
6.3.22	Execute external programs	98
6.3.23	Execute CAL-files	99
6.3.24	Error handling	100



- Postprocessor functions are done at the main calculation of the channels.



- Instead of temporary channel C1, a channel name can be used for creating a permanent channel!
- #CH stands for #Channel!

6.3.1 Additional formulas

6.3.1.1 Integer formula

The integer formula can be used to subsequently adjust a channel, if it was not set correctly before measurement.

The formula is given by:

$$C1 = \frac{\text{Multiplicand}}{\text{Divisor}} * (\#Channel - \text{Offset})$$

Syntax	Meaning
C1=F(#CH, I(Multiplicand,Divisor,Offset))	#CH is linearized with the respective parameters and outputted with channel #C1



- Parameters are integers!



- Parameters can be channels as well as constants
- Use command if zero position was not set correctly at recording
- Use command if channel calibration was not set correctly at recording

6.3.1.2 Real formula

The real formula can be used to subsequently adjust a channel I, if it was not set correctly before measurement

You can use any channel as parameter of the function. The formula is given by:

$$C1 = \text{Factor} * (\#Channel + \text{Offset})$$

Syntax	Meaning
C1=F(#CH, R(Factor, Offset))	#CH is linearized with the respective parameters and outputted with channel #C1



- Parameters are floats (single)!



- Parameters can be channels as well as constants
- Use command if zero position was not set correctly at recording
- Use command if channel calibration was not set correctly at recording

6.3.3 ChannelArray

The command ChannelArray was implemented to prepare output channels for rotation and then only need one command line for the actual rotation command. This reduces input errors of the user because the parameterization of the command is much clearer.



- The rotation commands Rot_3D... and Madgwick_AHRS IMU need prepared channels!

Syntax
C1=ChannelArray('Name_VAR1', 'Name_VAR2', 'Name_VAR3')
Meaning
Preparing the output channels for a subsequent calculation which expects prepared output channels.



- The by ChannelArray created channels are not saved if only ChannelArray without subsequent calculation is used!

Example:

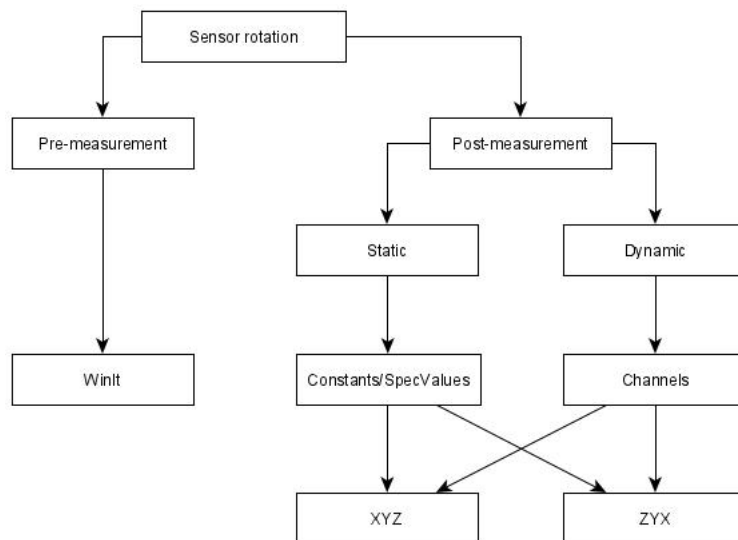
See example at chapter 6.3.4.

6.3.4 Rotational correction of channels

A sensors position can be corrected around all three axes either pre-measurement in WinIt or post-measurement via a CalcTool function.

At CalcTool a static rotation of sensor is possible, where constants or SpecSheet values are used, or a dynamic rotation with channel-values is possible.

For static as well as dynamic rotational correction, two different conventions are possible.



- Before any post-measurement rotation is executed, the output variables must be prepared with the function ChannelArray!

Rotation by Constants/SpecSheet values

6.3.4.1.1 Rot 3D_XYZ

Syntax
C1=Rot_3D_XYZ(#SourceCH1, #SourceCH2, #SourceCH3, Angle1, Angle2, Angle3)
Meaning
Rotating the input Channels #SourceCH1 by value of Angle 1, #SourceCH2 by value of Angle2 and #SourceCH3 by value of Angle3

6.3.4.1.2 Rot 3D_ZYX



- This function is same as previous but using another rotational convention!

6.3.4.2 Rotation by channel values

6.3.4.2.1 Rot 3D_XYZ_Var

Syntax
C1=Rot_3D_XYZ_Var(#SourceCH1, #SourceCH2, #SourceCH3, #RotateCH1, #RotateCH2, #RotateCH3)
Meaning
Rotating the input Channels #SourceCH1 by value of #RotateCH1, CH2 by value of #RotateCH2 and #SourceCH3 by value of #RotateCH3

6.3.4.2.2 Rot 3D_ZYX_Var



- This function is same as previous but using another rotational convention!

Example:

Rotating all three axes of an Accelerometer by SpeSheet-Values.

```
[RotateAcc]
Acc= ChannelArray('x_1_R', 'y_1_R', 'z_1_R')
Acc= Rot_3D_XYZ(#Acc_x_1_F, #Acc_y_1_F, #Acc_z_1_F, Sensor_1.Rot_x, Sensor_1.Rot_y, Sensor_1.Rot_z)
```

Code explanation:

At first the output channels #Acc_x_1_R, #Acc_y_1_R and #Acc_z_1_R of the subsequent rotation are prepared.

Afterwards, the actual rotation takes place:

- ➔ Channel #Acc_x_1_F is rotated by the SpecSheet value of group *Sensor_1* entry *Rot_x* and the values are transferred to output channel #Acc_x_1_R
- ➔ Channel #Acc_y_1_F is rotated by the SpecSheet value of group *Sensor_1* entry *Rot_y* and the values are transferred to output channel #Acc_y_1_R
- ➔ Channel #Acc_z_1_F is rotated by the SpecSheet value of group *Sensor_1* entry *Rot_z* and the values are transferred to output channel #Acc_z_1_R

6.3.5 Madgwick IMU AHRS

The function Madgwick_IMU_AHRS uses the gyroscope and accelerometer channels of all respective axes to create quaternions, which is a four-dimensional complex number that can be used to represent the orientation of a rigid body or coordinate frame in three-dimensional space.

Madgwick filter fuses angular velocities, not solid angles. From the accelerometer data, a gradient method is used to calculate a vector with direction and magnitude of rotation. The same is calculated from the data of the gyroscope and then merged. Only after this calculation an integration after time takes place.

Inclusion of the accelerometer data takes place to compensate the gyroscope drift and magnetic interference.



- This function is able either to output quaternions (4 values) or Eulers angles (3 values). It is controlled by the number of the prepared output channels of ChannelArray!

Syntax
C1=Madgwick_AHRS_IMU(#GYRO_x, #GYRO_y, #GYRO_z, #ACC_x_1, #ACC_y, #ACC_z, Value)
Meaning
Creating a four-dimensional complex number or either three Eulers angles to represent an orientation of a rigid body or coordinate frame in three-dimensional space. The parameter Value determines the inclusion of accelerometer data for gyro-drift compensation.



- The amount of prepared output channels determines if output is in
- Value is 0.0 when no compensation should take place

Example 1:

Creating four quaternion-channels #Quat_q0, #Quat_q1, #Quat_q2 and #Quat_q3.

```
[Madgwick_Quaternionen]
Quat=ChannelArray('_q0', '_q1', '_q2', '_q3')
Quat=Madgwick_AHRS_IMU(#GYRO_x, #GYRO_y, #GYRO_z, #ACC_x_1, #ACC_y, #ACC_z, 0.0)
```

Example 2:

Creating three Eulers angles #Quat_x, #Quat_y and #Quat_z.

```
[Madgwick_Euler]
Quat=ChannelArray('_x', '_y', '_z')
Quat=Madgwick_AHRS_IMU(#GYRO_x, #GYRO_y, #GYRO_z, #ACC_x, #ACC_y, #ACC_z, 0.0)
```

6.3.6 Filter function

Filters are used to smoothen a noisy signal or to obtain information about the dynamic movement of a signal (suspension, etc.). With the following functions, filtered channels can be calculated!

CalcTool provides two methods of defining a filter:

Simple filter interface

Only filter frequency must be defined by user. For all other values of the filter definition, default values are taken.

In most cases these values lead to the wanted result.

Full filter interface

For more individual or more advanced tasks, FIR-Filter (Finite Impulse Response-Filter) and IIR-Filters (Infinite Impulse Response-Filters) can be used!

6.3.6.1 Simple filter interface

This function is a predefined FIR (Finite Impulse Response) Low Pass filter with Hamming-window.



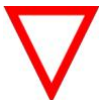
- Due to the FIR behaviour, only values preceding the current sample are included in the filtering.

The parameter *Number of Iterations* is determined as follows:

$$NumberOfIterations = 20 * \frac{SamplingRate}{CrossFrequency}$$

OR

$$NumberOfIterations = 401$$



- Always the smaller value is taken! Thus, maximal NumberOfIterations is 401!
- NumbersOfIterations must be odd!

Syntax	Meaning
C1=F(#CH, F(CrossFrequency))	Filtering #CH Hamming window and a determined CrossFrequency in Hz/Sec/s or Scans

Example:

```
[FIR_LP_SimpleInterface]
Result=F(#V_Sat, F(2.0 Hz))
```

Assuming, the speed channel #V_Sat is sampled with 25 Hz and should be filtered with a CrossFrequency of 2.0 Hz using Hamming Window.

The calculated Number of Iterations therefore is:

$$NumberOfIterations = 20 * \frac{SamplingRate}{CrossFrequency} = 20 * \frac{25Hz}{2Hz} = 250$$

Since the result is less than 401, the calculated value is used, but set to an odd value:

$$NumberOfIterations = 251$$

6.3.6.2 Full filter interface



- The full filter interface of *CalcTool* gives you also the possibility to define all parameters of a filter function

Syntax
 C1= F(#CH, F(FilterType WindowType CrossFrequency NumberOfIterations))



- There is only one space character between the filter parameters

Filter Type:	
L_P	Low pass filter
H_P	High pass filter
B_P	Bandpass filter
AVG	Center average filter (Average built from NumberOfIterations of values, half left of current sample and the other half right of current sample)
AVG+	Look ahead average filter (Average built from NumberOfIterations of values, only with samples right of current sample)
AVG-	Look back average filter (Average built from NumberOfIterations of values, only with samples left of current sample)
MED	Median filter
TOP	Maximum filter (like Median, but using the maximum value from the given interval)
MAX	
BOT	Minimum filter (like median, but using the minimum value from the given interval)
MIN	
TAU	Tau filter according to formula: $X_{Filtered}(t) = X_{Filtered}(t - 1) + 1/Iterations * (X(t) - X_{Filtered}(t - 1))$ Iterations can be a floating-point number! If filter is used as in Magneti Marelli: $Iterations = \frac{1}{filter\ value}$

Window Type:	
	Rectangle
Hann	Hanning
Hamm	Hamming
Blck	Blackman
Kaib	Kaiser Bessel
Parz	Parzen
Wlch	Welch

Cross frequency:
 Input of desired cross frequency

Number of Iterations:
 Input of desired Number of Iterations



- If characters “Sec” or “s” are placed at position of NumberOfIterations, *CalcTool* calculates the iterations from given time value using the sampling rate of input channel!

Example 1:

Calculating an average of 100Hz-channel V_GPS 0.25 sec before and after current sample with a center average filter using a rectangular window.

Time-input in [sec]:

```
[Center_AVG_Filter_Time_SEC]
Result=F(#V_GPS, F(AVG 0.5 sec))
```

Time-input in [s]:

```
[Center_AVG_Filter_Time_S]
Result=F(#V_GPS, F(AVG 0.5 s))
```

Frequency-input in [Hz]

```
[Center_AVG_Filter_Time_Hz]
Result=F(#V_GPS, F(AVG 2 Hz))
```

Scan-input:

$$\text{Scans} = t * f = 0.5 \text{ sec} * 100 \text{ Hz} = 50$$

```
[Center_AVG_Filter_Scans]
Result=F(#V_GPS, F(AVG 50))
```

⇒ The calculation results are the same!

Example 2:

The speed #V_GPS is filtered with a low pass filter with a Hanning window, a cross frequency of 2.0 Hz and the number of iterations is 100.

```
[LP_Hanning]
Result=F(#V_GPS, F(L_P Hann 2.0Hz 100))
```

6.3.6.3 Full filter interface with variable depth

In some cases, it might be useful not to use a fix depth for filtering but a variable. To address these cases, it is possible to define a filter with a variable filter depth.



- The depth of the filter is given by a channel



- Only determine FilterTypes can be used with variable depth!

Syntax	Meaning
C1=F(#CH, F(FilterType #DepthChannel))	Filtering #CH with different FilterTypes with a variable depth depending on value of #DepthChannel

Filter Type:	
AVG	Centre average filter (Average built from NumberOfIterations of values, half left of current sample and the other half right of current sample)
AVG+	Look ahead average filter (Average built from NumberOfIterations of values, only with samples right of current sample)
AVG-	Look back average filter (Average built from NumberOfIterations of values, only with samples left of current sample)
MED	Median filter
TOP	Maximum filter (like Median, but using the maximum value from the given interval)
MAX	
BOT	Minimum filter (like median, but using the minimum value from the given interval)
MIN	

Example:

The example shows a centre average filter, where the depth of the filter is dependent on the temporary channel "C1". This channel C1 is calculated to 1/250 of the RPM signal of the vehicle in the first line. The higher the RPM signal of the engine, the deeper is the filter.

```
[Center_AVG_Filter_VarDepth]
C1      = /(#RPM, 250)
Result = F(#Susp_F, F(AVG #C1))
```

6.3.6.4 RCLP-Filter

The RCLP (resistor-capacitor lowpass) function behaves like an analogue 6dB RC lowpass filter.



- Due to the analogue filter attribute, this function can also be executed online in Logger during recording!

Syntax	Meaning
C1=RCLP(#CH, CrossFrequency HZ)	Filtering #CH with an analogue RC lowpass filter with CrossFrequency in Hz/Sec/s

6.3.6.5 IIR-Lowpass-Filter

The IIR is an extension of the RCLP filter to a 12dB recursive filter. In addition to the forward RCLP filter, this filter calculates an inverse, backward RCLP filter to obtain a time discretization.

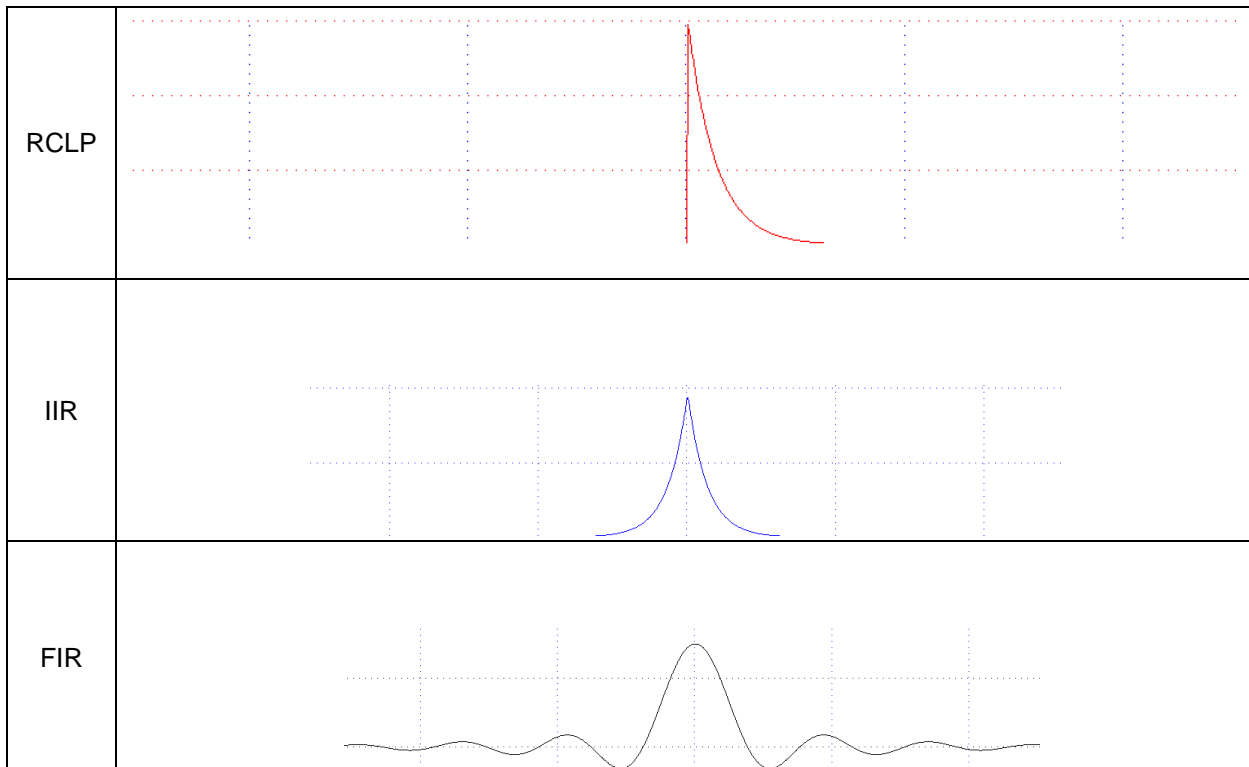


- Especially for time crucial calculations it is recommended to use IIR filters!
- With IIR-Lowpass-Filter a Bandpass-Filter can be formed!

Syntax	Meaning
C1=F(#CH,F(IIR(CrossFrequency HZ)))	Filtering #CH with IIR lowpass filter with CrossFrequency in Hz/Sec/s

6.3.6.6 Comparison of impulse responses

Impulse response to a peak with amplitude 1 and duration of 1 ms.



- Due to the forward and backward filtering of the IIR filter the amplitude of the filtered channel is smaller than the amplitude of the channel filtered by the RCLP filter!

6.3.7 Frequency adjustments

6.3.7.1 New main frequency

This function allows you to change the main frequency of the measurement.

Sampled frequency can be reduced (information will be lost) as well as increased (not more information is created).

Syntax	Meaning
NewMainFreq(NewRate)	Change samplingrate of the measurement to NewRate



- NewRate must be a multiple of the current rate!

6.3.7.2 Frequency reduction/increase

This function changes frequency of a channel with interpolation to changed frequency.

Sampled frequency can be reduced (information will be lost) as well as increased (not more information is created).

Syntax	Meaning
C1=Freq(#CH, NewRate)	Change Samplingrate of Channel #CH with interpolation to NewRate



- To reduce memory required for calculations, channel frequencies can be reduced!
- Values will be interpolated when changing frequency of channel!

Example:

See example at 6.3.7.3

6.3.7.3 Frequency reduction/increase without interpolation

This function changes frequency of a channel without interpolation to changed frequency.

Sampled frequency can be reduced (information will be lost) as well as increased (not more information is created).

Syntax	Meaning
C1=FreqNI(#CH, NewRate)	Change Samplingrate of Channel #CH with <u>no</u> Interpolation to NewRate



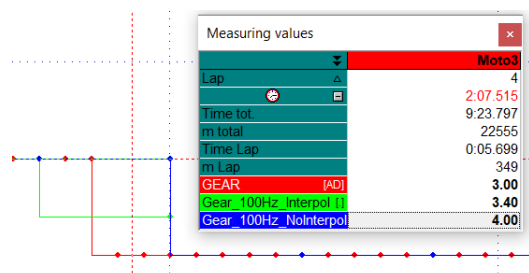
- To reduce memory required for calculations, channel frequencies can be reduced!

Example:

To save memory the frequency of #Gear is set from 500 Hz to 100 Hz with and without interpolation

#GEAR changes from 4 to 3. The interpolated channel #Gear_100Hz_Interpol obtains an intermediate value 3.40 by the interpolation during gear change while the next smaller gear is directly shown at #Gear_100Hz_NoInterpol.

#Gear_100Hz_Interpol and #Gear_100Hz_NoInterpol do not show the gear change at the same positions as #GEAR, so both lose information compared to #GEAR.



6.3.8 GPS commands

6.3.8.1 Sampling rate adjustment for non 2D sampling rates

If a 2D device is used to record data from CAN bus, it is possible that refresh rates of foreign GPS-CAN-devices does not match with one of the possible recording rates of the 2D-logger.

Example:

A foreign GPS device sends its data at 4 Hz.

The possible logger sampling rates are 400 Hz, 200 Hz, 100 Hz, 50 Hz, 25 Hz and 12.5 Hz.

50 Hz was selected to be the sampling rate for the loggers GPS channels.

In this case, the combination of 4 Hz refresh-rate and 50 Hz sampling rate, has the consequence, that one GPS-sample is each recorded 12.5 (50Hz/4Hz =12.5) times at the logger.

Since it is not possible to have a value a half time, the channel has 12 or 13 identical values for each GPS-sample. The different number of the same samples can result in aliasing.

This issue is solved by the command *FreqBase* which sets the number of identical values always to the same.

Syntax	Meaning
C1=FreqBase(#CH, NewRate, GPSRate)	Adjusting the sampling rates of foreign GPS-devices channel #CH with a refresh rate GPSRate to 2D-logger sampling rates NewRate.

#CH	Name of the channel
NewRate	New samplingrate
GPSRate	Rate at which the GPS sends new data



- Value for *NewRate* must be less or equal to the main sampling rate of the logger!
- *NewRate* must be divisible by the *GPSRate* without rest!

Continuation of example:

The GPS sends speed data (#V_SAT) with 4 Hz.

The data is recorded with a sampling rate of 50 Hz.

The NewRate must be a sampling rate of the logger and must be divisible by 4.

Thus, in this case the lowest possible NewRate is 100:

```
C1=FreqBase(#V_Sat, 100, 4)
```

6.3.8.2 Find GPS frequency

This function is a special function for GPS recordings. Normally the GPS channels are recorded with a higher rate than the GPS antenna sends new values.

This function determines the real refresh rate of the GPS antenna based on the value changing of a channel.

Syntax	Meaning
C1= FindGPSFreq(#GPS_Channel)	Finds the GPS frequency via the used GPS channel



- #V_Sat or #SSHH are adequate channels to determine real refresh rate of GPS antenna

6.3.8.3 GPS shift



- Usable for all GPS modules which provide a channel named "T_Shift" additionally to normal GPS channels
- This channel indicates, how much milliseconds ago the values of the other channels were valid.

Syntax	Meaning
C1=GPSShift(#GPSChannel, #ShiftChannel, NewFreq)	Create channel C1 with the frequency of NewRate and the data of #GPSChannel shifted by the time of #ShiftChannel

GPSChannel	Determine which channel will be outputted as #C1 with applied shift time
ShiftChannel	Additional GPS channel #T_Shift provided by GPS module in [Scans]
NewFreq	Sampling Rate of resulting channel #C1



- The higher the frequency is, the Shift function works more exactly (it must be always rounded on whole scans).
- NewFreq must be less or equal main sampling rate! If bigger, a warning appears that main sampling rate is used!
- Result channels are always channels of type float (double if the GPS channel was double, otherwise channels with a single precision)
- Between the "shifted" GPS values a linear interpolation will be done.

6.3.8.4 Make32GPS

Syntax	Meaning
C1=Make32GPS(#CH)	Change data type from 32-bit, integer GPS-channel to double precision with output range of -180°...+180°



- Input channel #CH must be a 32-bit-channel (normally #Latitude or #Longitude)
- The resulting channel #C1 will be a double precision channel with range -180° ... +180°

6.3.9 Channel manipulation

6.3.9.1 Demultiplexing a multiplexed channel

To reduce the traffic on the CAN bus, sometimes more than one channel is sent on a single identifier at the same byte position. In order to interpret the data on the data channel the interpretation information must be sent to my other channel.

Syntax	Meaning
C1=MUX(#DataChannel, #MuxChannel, MuxVal)	Channel MuxVal specifies with which multiplex information #MuxChannel the multiplex data on #DataChannel should be read

#DataChannel	Name of the channel containing the multiplexed data
#MuxChannel	Name of the channel containing the multiplex information (0/1)
MuxVal	Decision value which data of #DataChannel should be read

Example:

```
[OilTemp]
Result=MUX(#Fluid_Temp, #Fluid_Temp_MUX, 1)
```

The channel "Fluid_Temp" contains multiplexed data for water- and oil-temperature

In this case #Fluid_Temp_MUX=0 stands for water temperature and #Fluid_Temp_MUX=1 stands for oil temperature.

Due to the MuxVal=1 the resulting channel #Oil_Temp will have the oil temperature values.

6.3.9.2 Masking out used bits

To use as few identifiers as possible it is usual to aggregate more than one channel in a 8, 16 or 32 bit value on the CAN bus. With 2D devices it is only possible to record up to four whole bytes from the CAN-bus.

Sometimes it is necessary to mask out only several bits from the recorded channel, this can be done by using the UseBits function.

Syntax	Meaning
C1=UseBits(#DataChannel, MaskValue)	Extracts the values of #DataChannel using #MaskValue to mask out bits.

Example:

The channel "DataChannel" is a 16-bit channel recorded from the CAN bus.

- ➔ The four least significant bits represent the selected gear of the vehicle.
- ➔ The 12 most significant bits represent the speed.

```
[Gear]
Result = UseBits(#DataChannel, 0b1111)

[Speed]
Result = UseBits(#DataChannel, 0b1111111111110000)
```

Written in hexadecimal notation:

```
[Gear]
Result = UseBits(#DataChannel, 0xF)

[Speed]
Result = UseBits(#DataChannel, 0xFFFF0)
```



- Different notations are possible!

6.3.9.3 Shifting a channel

This function allows to shift a channel on the time axis. The unit for the shift may either be seconds or scans.

Syntax	Meaning
C1=Shift(#CH, Constant)	Shifting #CH by the number of scans
C1=Shift(#CH, Constant sec)	Shifting #CH by time in seconds



- One scan is the time between two measured values of the channel. That means if the channel is recorded with 50 Hz one scan is $1/(50 \text{ Hz}) \Leftrightarrow 0.02 \text{ sec}$.
- Negative number of scans or time → Shifting to left
- Positive number of scans or time → Shifting to right

Examples:

```
[V_Front_ShiftBySamplesToLeft]
C1= Shift(#VFront, -200)
```

```
[V_Front_ShiftByTimeToLeft]
C1= Shift(#VFront, -0.05 sec)
```



- In both cases, #VFront was shifted the same, because of the formula $f = \frac{1}{t}$

6.3.9.4 Combining 16-bit channels

Most 2D modules have the possibility to record CAN channels with 16-bit resolution as well as 32-bit resolution. If the logger does not provide enough 32-bit channels to record data, the information can be split in two 16-bit channels.

After downloading the data, these two 16-bit channels can be combined to one 32-bit channel using the following functions:

Name	Meaning	Output-Range
Make32	Combine two 16-bit channels to one <u>integer</u> 32-bit channel	0°...360°
MakeGPS	Combine two 16-bit channels to one <u>integer</u> 32-bit channel	-180°...+180°
MakeSingle	Combine two 16-bit channels to one <u>single-precision</u> 32-bit channel	

6.3.9.4.1 Make32

Syntax	Meaning
C1=Make32(#CHHi, #CHLo)	Combine two 16-bit channels to one <u>integer</u> 32-bit channel

#CHHi	Input channel containing the high 16 bits
#CHLo	Input channel containing the low 16 bits



- Output-Range: 0°...360°
- The resulting channel C1 receives the calibration formula of #CHLo

Example:

Combining two 16-bit channels to global Latitude with range 0°...360°

```
[Latitude]
Result = MAKE32(#LatituHi, #LatituLo)
```

6.3.9.4.2 MakeGPS

Syntax	Meaning
C1=MakeGPS(#CHHi, #CHLo)	Combine two 16-bit channels to one <u>integer</u> 32-bit channel
#CHHi	Input channel containing the high 16 bits
#CHLo	Input channel containing the low 16 bits



- Output-Range: -180°...+180°
- The resulting channel C1 receives the calibration formula of #CHLo

Example:

Combining two 16-bit channels to global Latitude with range -180°...+180°

[Latitude]

Result=MakeGPS(#Lat_deHi, #Lat_deLo)

6.3.9.4.3 MakeSingle

Syntax	Meaning
C1=MakeSingle(#CHHi, #CHLo)	Combine two 16-bit channels to one <u>single-precision</u> 32-bit channel
#CHHi	Input channel containing the high 16 bits
#CHLo	Input channel containing the low 16 bits



- The resulting channel C1 receives the calibration formula of #CHLo

6.3.9.5 Combining 32-bit channels

If channels are recorded from the CAN bus (see chapter before), it might be possible, that the data sent by a CAN device has a resolution of 64 bit. As there are not many modules able to record 64-bit channels, the information might be split in two 32-bit channels or four 16-bit channels. Therefore, a 64-bit channel must be created.

The following functions can be used:

Name	Meaning
Make64	Combine two 32-bit channels to one <u>integer</u> 64-bit channel
MakeDouble	Combine two 32-bit channels to one <u>single-precision</u> 64-bit channel



- If only 16-bit channels are provided, this must be done in two steps:
 1. Combining four 16-bit to two 32-bit channels using the *Make32* function
 2. Combining these two 23-bit channels to one 64-bit channel.

6.3.9.5.1 Make64

Syntax	Meaning
C1=MAKE64(#CHHi, #CHLo)	Combine two 32-bit channels to one <u>integer</u> 64-bit channel

#CHHi	Input channel containing the high 32 bits
#CHLo	Input channel containing the low 32 bits



- The resulting channel C1 receives the calibration formula of #CHLo

6.3.9.5.2 MakeDouble

Syntax	Meaning
C1=MakeDouble(#CHHi, #CHLo)	Combine two 32-bit channels to one <u>single-precision</u> 64-bit channel

#CHHi	Input channel containing the high 32 bits
#CHLo	Input channel containing the low 32 bits



- The resulting channel C1 receives the calibration formula of #CHLo

Example:

Combining four 16-bit channels to a double-precision, 64-bit channel

[Double]

C1 = Make32 (#LoHi, #LoLo)

C2 = Make32 (#HiHi, #HiLo)

Result= MakeDouble (#C1, #C2)

6.3.10 If function

This function is used to compare one channel with a constant or another channel. Depending on whether the condition was fulfilled, the resulting receives either the value of TrueResult or FalseResult.

Syntax	Meaning
C1=if(#CH, Operator, #CH , TrueResult, FalseResult)	Comparing two channels
C1=if(#CH, Operator, Constant , TrueResult, FalseResult)	Comparing channel with constant

Possible Operators:	
>	Bigger than
<	Smaller than
>=	Same and bigger than
<=	Same and smaller than
<>	Bigger and smaller than
=	Same



- Depending on whether the condition was fulfilled, channel C1 receives either the value of TrueResult or FalseResult
- TrueResult & FalseResult can also be constants
- If function is often used to get a True/False evaluation

6.3.11 SOD

To provide a more convenient use of the time of day, the time is converted into seconds. This is done with the command SOD (SecondsOfDay).

Syntax	Meaning
C1=SOD(#HHMM, #SSHH)	Calculating Seconds of Day from channels #HHMM and #SSHH
C1=SOD(#HHMM, #SSHH, Rate)	Calculating Seconds of Day from channels #HHMM and #SSHH and set output rate of created channel

Example:

Time 13:40:45.258
 #HHMM 13.40
 #SSHH 45.258

HH	13 h	*3600	46800.000 sec
MM	40 min	*60	2400.000 sec
SS	45 sec	*1	45.000 sec
HH	0.258 sec	*1	0.258 sec
SecondsOfDay			49245.258 sec

6.3.12 Mathematical functions

6.3.12.1	Addition.....	54
6.3.12.2	Subtraction.....	54
6.3.12.3	Multiplication.....	54
6.3.12.4	Division.....	54
6.3.12.5	Modulo.....	55
6.3.12.6	Integer division.....	55
6.3.12.7	Logarithm.....	55
6.3.12.8	Absolute value.....	55
6.3.12.9	Minimum.....	55
6.3.12.10	Maximum.....	56
6.3.12.11	Derivation.....	56
6.3.12.12	Integration.....	57
6.3.12.13	Sum.....	57
6.3.12.14	Power.....	57
6.3.12.15	Eulers exponent.....	58
6.3.12.16	Square root.....	58
6.3.12.1	Signum.....	58
6.3.12.2	Rounding values.....	59
6.3.12.3	Trigonometric functions.....	60

6.3.12.1 Addition

CalcTool supports following types of additions:

Syntax	Meaning
C1=#CH1, #CH2)	Addition of channels
C1=#CH, Constant)	Addition of channel and constant



- Multiplication in connection with if-function can also be used for Logic-OR-function

6.3.12.2 Subtraction

CalcTool supports following types of subtractions:

Syntax	Meaning
C1=#CH1, #CH2)	Subtraction of channel
C1=#CH, Constant)	Subtraction of channel and constant

6.3.12.3 Multiplication

CalcTool supports following types of multiplications:

Syntax	Meaning
C1=#CH1, #CH2)	Multiplication of channels
C1=#CH, Constant)	Multiplication of channel and constant

Example:

Squaring a channel.

```
[Speed_Square]
Result =*(#Speed, #Speed)
```



- Multiplication in connection with if-function can also be used for Logic-AND-function

6.3.12.4 Division

CalcTool supports following types of divisions:

Syntax	Meaning
C1=#CH1, #CH2)	Division of channels
C1=#CH, Constant)	Division of channel and constant

Example1:

Conversion of speed from [km/h] to [m/s]

```
[Speed_MS]
Result =/(#Speed_kmh, 3.6)
```

Example2:

Conversion of speed from [km/h] to [mph]

```
[Speed_mph]
Result =/(#Speed_kmh, 1.62)
```

6.3.12.5 Modulo

The result of the modulo function is the rest of a division:

$$10 \text{Mod} 7 = 10 : 7 = 1.43 \Rightarrow 1.43 \rightarrow 1 \Rightarrow \text{Rest} = 10 - 7 * 1 = 3$$

$$10 \text{Mod} 5 = 10 : 5 = 2.00 \Rightarrow 2.00 \rightarrow 2 \Rightarrow \text{Rest} = 10 - 5 * 2 = 0$$

CalcTool support following kinds of modulo functions:

Syntax	Meaning
C1=Mod(#CH1, #CH2)	Modulo of channels
C1=Mod(#CH, constant)	Modulo of channel and constant

Example:

Masking out lowest 5 bits → Lo5Bit=Mod(#CH, 32)



- Can be used for masking out bits
- Can be used for incremental counters

6.3.12.6 Integer division

The result of the DIV function (integer division) is the quotient of an integer division without rest

$$10 \text{DIV} 7 = 10 : 7 = 1.43 \rightarrow \text{Result} = 1$$

$$10 \text{DIV} 5 = 10 : 5 = 2.00 \rightarrow \text{Result} = 2$$

CalcTool supports following kinds of DIV functions:

Syntax	Meaning
C1=Div(#CH1, #CH2)	Integer Division of channels
C1=Div(#CH, Constant)	Integer Division of channel and constant

6.3.12.7 Logarithm

This function allows you to calculate the decimal logarithms from any channel.

$$C1 = \log_{10}(\#CH)$$

Syntax	Meaning
C1=Log(#CH)	Calculating the decimal logarithm of #CH

6.3.12.8 Absolute value

This function gives the absolute value of the calculated channel.

$$C1 = |\#CH|$$

Syntax	Meaning
C1=Abs(#CH)	Calculating the absolute value of #CH

6.3.12.9 Minimum

The channel which currently has the lower value is the resulting channel.

Syntax	Meaning
C1=Min(#CH1, #CH2)	Result of this function is the minimum of the two compared channels

Example:

Comparing car left and right rear suspension and always take the minimum of these two channels.

6.3.12.10 Maximum

The channel which currently has the higher value is the resulting channel.

Syntax	Meaning
C1=Max(#CH1, #CH2)	Result of this function is the maximum of the two compared channels

Example:

Comparing car left and right rear suspension and always take the maximum of these two channels.

6.3.12.11 Derivation

Formation of the mathematical derivation of a channel.

$$C1 = \frac{\Delta(\#CH)}{\Delta t}$$

There are two different possibilities to execute a derivation:

Syntax	Meaning
C1=Derivate(#CH)	Executing a mathematical derivation of #CH
C1=F'(#CH)	



- The names of derivation-commands differ, but the functionality of both is the same.

Example1:

Deriving the speed of the vehicle to get longitudinal acceleration [km/h] → [m/s²]



- If a channel with dimension [km/h] is derived, the conversion from [km/h] to [m/s] is done automatically and the resulting channel gets the dimension [m/s²]

Example2:

Deriving travel of the fork to get the speed of fork movement [mm] → [mm/s]

Example3:

Deriving a channel to get local turn-around points.

6.3.12.12 Integration

Formation of the mathematical integration of a channel

$$C1 = Integrate(\#Channel) \Leftrightarrow C1 = \int (\#Channel)dt$$

Syntax	Meaning
C1=Integrate(#CH)	Executing a mathematical integration of #CH
C1=Integrate(#CH , Value)	Executing a mathematical integration of #CH, set resulting channel #C1 at every laptrigger to Value



- Value 0 must be used to calculate the resulting value per lap



- For an example of integration per lap please watch the video MyFirstCal on our Youtube channel <https://www.youtube.com/watch?v=8COo3toU44s>

Example1:

Calculate energy from the physical values of a channel (e.g. on brake signal to know energy spent by lap).

Example2:

Integrate speed of longitudinal acceleration of vehicle to get speed [m/s²] → [m/s]



- If a velocity is first derived to an acceleration by means of derivation and this acceleration is then integrated so that a velocity is obtained again, the velocity resulting from integration will have a small difference to the original velocity which is due to the integration constant!

6.3.12.13 Sum

This function calculates the sum of all values of a channel.

$$C1 = Sum(\#Channel) \Leftrightarrow C1 = \sum(\#Channel)$$

Syntax	Meaning
C1=Sum(#CH)	Calculating the sum of all values of a channel
C1=Sum(#CH , Value)	Calculating the sum of all values of a channel, set resulting channel #C1 at every laptrigger to Value



- Value 0 must be used to calculate the resulting value per lap

Example1:

Counting up or down-shifts per lap.

6.3.12.14 Power

Executing exponential calculations.

$$C1 = Power(Base, Exponent) \Leftrightarrow C1 = Base^{Exponent} \Leftrightarrow C1 = \#Channel1^{\#Channel2}$$

Syntax	Meaning
C1=Power (#CH1, #CH2)	Base as channel & Exponent as channel
C1=Power (#CH, Constant)	Base as channel & Exponent as constant
C1=Power (Constant, #CH)	Base as constant & Exponent as channel

6.3.12.15 Eulers exponent

Executing exponential calculations with base Euler's constant.

$$C1 = \text{Exp}(\#Channel) \Leftrightarrow C1 = e^{\#channel}$$

Syntax	Meaning
C1=Exp(#CH)	Executing exponential calculations with base Euler's constant.



- Also possible: C1=Power(@E, #CH)

6.3.12.16 Square root

Calculating the square root of a channel.



- Calculation is made with the absolute value of channel!

$$C1 = \text{SQRT}(\#Channel) \Leftrightarrow C1 = \sqrt{|\#Channel|}$$

Syntax	Meaning
C1=SQRT(#CH)	Calculating the square root of a channel.

Example: Calculation of combined Gs for a car to define the total tractive effort available for braking while turning.

$$\text{Combined_Gs} = \sqrt{(\text{Lateral}_{Gs})^2 + (\text{Longitudinal}_{Gs})^2}$$

6.3.12.1 Signum

Evaluating sign of channel value

Syntax	Meaning
C1=Sig(#CH)	Creating a channel value (-1, 0 or +1), depending on the channels value sign

Example:

> 0 → +1
 < 0 → -1
 = 0 → 0

6.3.12.2 Rounding values

CalcTool provides three different functions to round fractional numbers to integers.

6.3.12.2.1 Round

Rounding a channel value to an integer.

Syntax	Meaning
C1=Round (#CH)	Rounding channel to an integer



- The rounding function set the result to the next higher integer if the fraction of the channel value is greater than or equal to 0.5
- If the fraction is smaller than 0.5 it goes to the next smaller integer value

Example:

1.5 → 2
 -1.5 → -1
 1.49 → 1

6.3.12.2.2 Floor

Rounding a channel value to next smaller integer.

Syntax	Meaning
C1= Floor (#CH)	Rounding channel to next smaller integer

Example:

1.5 → 1
 -1.5 → -2
 1.99 → 1

6.3.12.2.3 Trunc

Cutting the fractional part of a channel value.

Syntax	Meaning
C1=Trunc(#CH)	Cutting the fractional part of a channel value

Example:

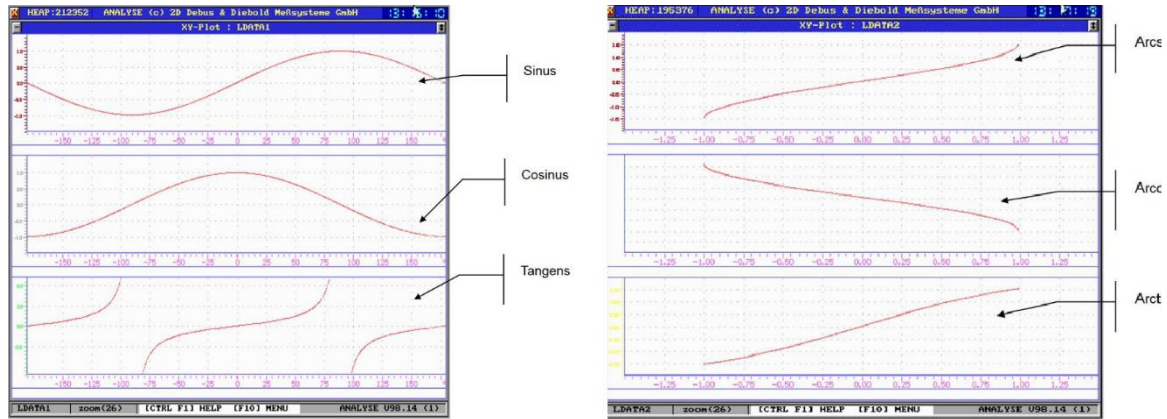
1.5 → 1
 -1.5 → -1
 1.00 → 1

6.3.12.3 Trigonometric functions

These functions allow you to calculate the sine, cosine, tangent, arcsine, arccosine and arctangent of a channel.



- The input for sine, cosine and tangent functions can be in radians or degrees
- The result for arcsine, arccosine and arctangent will be only in radians.



Syntax	Meaning	Comment
C1=dsin(#CH)	Sine(#CH)	(Resulting channel in degrees)
C1=dcos(#CH)	Cosine(#CH)	(Resulting channel in degrees)
C1=dtan(#CH)	Tangent(#CH)	(Resulting channel in degrees)
C1=sin(#CH)	Sine(#CH)	(Resulting channel in radians)
C1=cos(#CH)	Cosine(#CH)	(Resulting channel in radians)
C1=tan(#CH)	Tangent(#CH)	(Resulting channel in radians)
C1=arcsin(#CH)	Arcsine(#CH)	(Resulting channel in radians)
C1=arccos(#CH)	Arccosine(#CH)	(Resulting channel in radians)
C1=arctan(#CH)	Arctangent(#CH)	(Resulting channel in radians)
C1=arctan2(#CH1, #CH2)	Arctangent2(#CH1, #CH2)	(Resulting channel in radians)

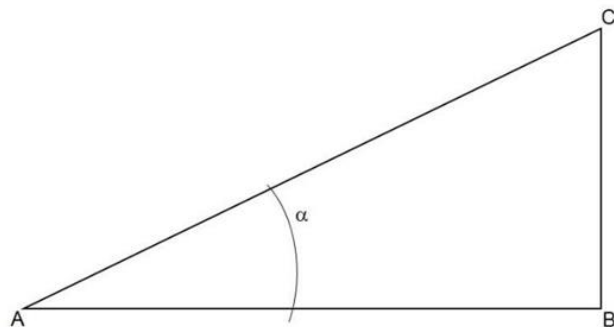


- Arctan2 extends the output of arctan-function from two to four quadrants!
- For conversion from degrees in radians and the other way round, pre-defined constants (see 6.1.4.2) can be used.

$$\sin(\alpha) = \frac{BC}{AC}$$

$$\cos(\alpha) = \frac{AB}{AC}$$

$$\tan(\alpha) = \frac{BC}{AB}$$



$$\alpha \text{ in [rad]} = \arcsin\left(\frac{BC}{AC}\right) = \arccos\left(\frac{AB}{AC}\right) = \arctan\left(\frac{BC}{AB}\right)$$

6.3.13 Logical functions

Logical functions are binary, logical calculations.



- These functions only work with WORD channels! (see 6.3.16.1)

6.3.13.1 AND

CalcTool supports following types of AND functions:

Syntax	Meaning
C1=AND(#CH1,#CH2)	Logical AND between channels
C1=AND(#CH, Constant)	Logical AND between channel and constant

Example:

Mask out the lowest 8 bit of a channel (Lo8Bit=And(#CH, 0xFF)).

6.3.13.2 NAND

CalcTool supports following types of NAND functions:

Syntax	Meaning
C1=NAND(#CH1,#CH2)	Logical NAND between channels
C1=NAND(#CH, Constant)	Logical NAND between channel and constant

6.3.13.3 OR

CalcTool supports following types of OR functions:

Syntax	Meaning
C1=Or(#CH1, #CH2)	Logical OR between channels
C1=Or(#CH, Constant)	Logical OR between channel and constant

6.3.13.4 XOR

CalcTool supports following types of XOR functions:

Syntax	Meaning
C1=Xor(#CH1, #CH2)	Logical XOR between channels
C1=Xor(#CH, Constant)	Logical XOR between channel and constant

6.3.13.5 Not

CalcTool supports following types of NOT function:

Syntax	Meaning
C1=Not(#CH)	Logical negation of the channel

6.3.14 Signal analysis

6.3.14.1	VCount.....	63
6.3.14.2	CountValue.....	63
6.3.14.3	DecWhileTrue.....	63
6.3.14.4	Minima	64
6.3.14.5	Maxima.....	67
6.3.14.6	First value	71
6.3.14.7	Last value.....	71
6.3.14.8	AvgValue.....	71
6.3.14.9	AverageWhileTrue.....	71
6.3.14.10	Limit.....	71
6.3.14.11	Rising Edge	72
6.3.14.12	Falling Edge	72
6.3.14.13	Zero Cross.....	72
6.3.14.14	FillFromBool	73
6.3.14.15	FillWithNextBool	74
6.3.14.16	TimeForTrue	75
6.3.14.17	TimeSinceTrue.....	75
6.3.14.18	HoldWhileTrue	76
6.3.14.19	ExpandWhileTrue	77
6.3.14.20	DeltaWhileTrue	78
6.3.14.21	InterpolateWhileFalse.....	79
6.3.14.22	InterpolateWhileTrue.....	80
6.3.14.23	Countchanges.....	80
6.3.14.24	VDVWhileTrue.....	80
6.3.14.25	RMSWhileTrue	81
6.3.14.26	MOVWhiletrue	82

6.3.14.1 VCount

Syntax	Meaning
C1= VCount(#CH)	Increments a counter if channel value and its successive scan are equal



- If the successive values are not equal, counter is reset to 0

6.3.14.2 CountValue

The function CountValue sums up all values of samples as long as #CH is inside the tolerance around value.

Syntax	Meaning
C1=CountValue(#CH, value, tolerance)	Summing up all sample values as long as #CH is inside the tolerance around value

Example:

Evaluating the sum of sample values when #T_Oil was between 90° +/-10°.

```
[SumOilTempValuesInRange]
Result=CountValue(#T_Oil, 90, 10)
```

6.3.14.3 DecWhileTrue

Decrements the first value of SourceChannel by 1 as long BoolChannel is > 0.

Syntax	Meaning
C1=DecWhileTrue(#SourceChannel, #BoolChannel)	Decrements the first value of SourceChannel by 1 as long BoolChannel is > 0.



- If #CheckChannel is = 0, #C1 is reset to 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.4 Minima

6.3.14.4.1 MinValue

The MinValue function searches all values of a given channel to find the minimum. It creates a channel with the constant value of this minimum.

Syntax	Meaning
C1=MinValue(#CH, Rate)	Creates a channel #C1 with a constant corresponding to minimum value of #CH. Rate defines the sampling rate at which channel #C1 is saved.



- If no Rate is defined, the resulting channel #C1 gets @MainSamplingrate as frequency

6.3.14.4.2 MinHoldWhileTrue

This function determines the minimum value of a SourceChannel as long as BoolChannel is > 0.



- The minimum value of the SourceChannel in the BoolChannel area is determined successively as long as the BoolChannel is > 0.

Syntax	Meaning
C1=MinHoldWhileTrue(#SourceChannel, #BoolChannel)	Successive determination of minimum value of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel = 0 the resulting channel C1 carries the value of SourceChannel
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.4.3 MinWhileTrue

This function determines the minimum value of a SourceChannel as long as BoolChannel is > 0.



- The minimum value of the SourceChannel in the BoolChannel-Area is held as long as the BoolChannel is > 0.

Syntax	Meaning
C1=MinWhileTrue(#SourceChannel, #BoolChannel)	Holding the minimum value of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel = 0 the resulting channel C1 carries the value of SourceChannel
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.4.4 PosMinWhileTrue

This function highlights the minimum value of a SourceChannel as long as BoolChannel is > 0 with a Boolean TRUE.



- The minimum value of the SourceChannel in the BoolChannel-Area is highlighted with a Boolean TRUE.

Syntax	Meaning
C1=PosMinWhileTrue(#SourceChannel, #BoolChannel)	Highlighting the minimum value of SourceChannel as long as BoolChannel is > 0.



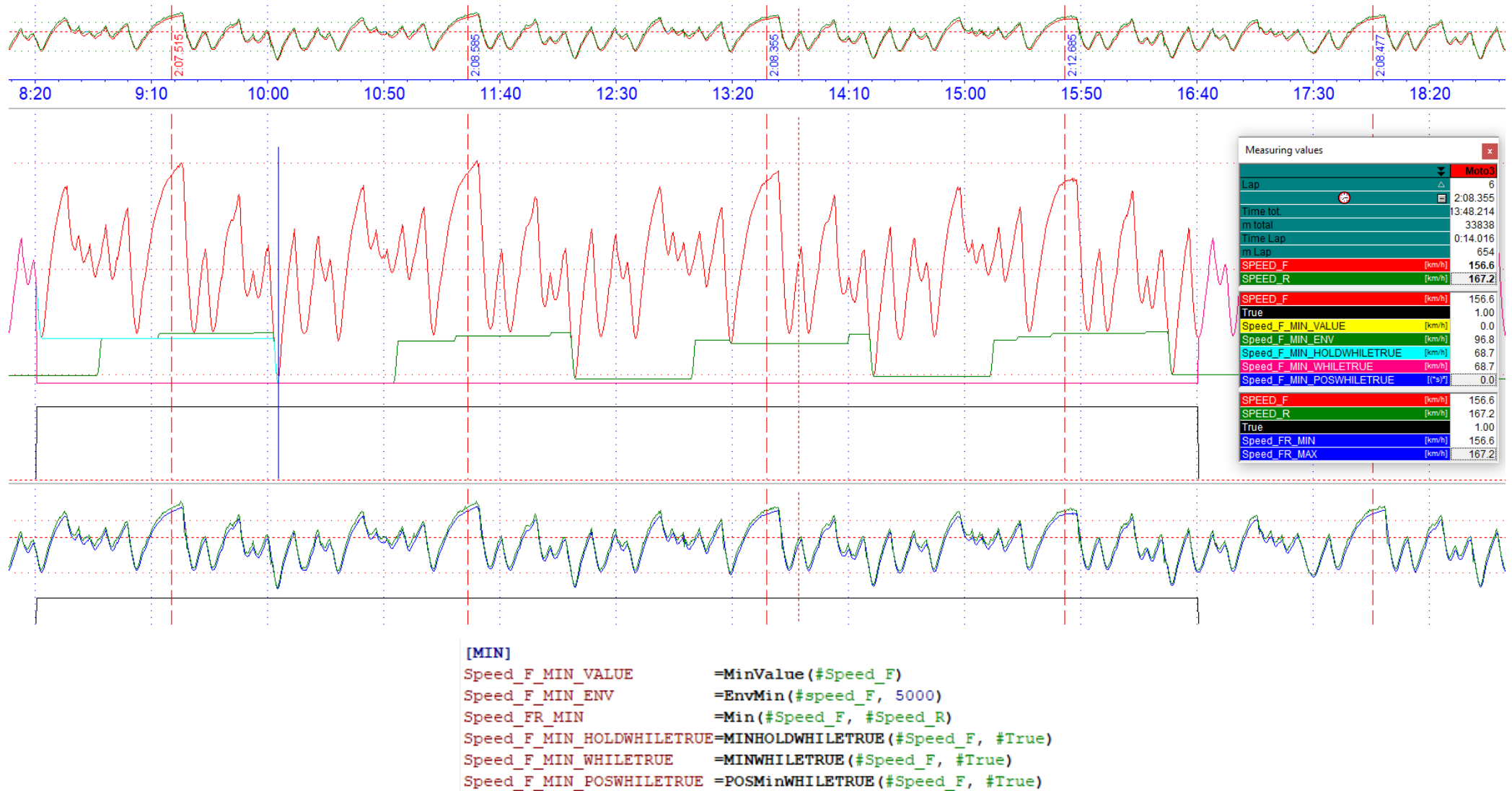
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.4.5 EnvMin

The EnvMin function searches a certain number of values of a certain channel to find the minimum within this range.

Syntax	Meaning
C1=EnvMin(#CH, samples number)	The EnvMin function searches a certain number of values of a certain channel to find the minimum within this range.

6.3.14.4.6 Different minima functions



This document is subject to change at 2D decision. 2D assumes no responsibility for any claims or damages arising out of the use of this document, or from the use of modules based on this document, including but not limited to claims or damages based on infringement of patents, copyrights or other intellectual property rights.

6.3.14.5 Maxima

6.3.14.5.1 MaxValue

The MaxValue function searches all values of a given channel to find the maximum. It creates a channel with the constant value of this maximum.

Syntax	Meaning
C1=MaxValue(#CH, Rate)	Creates a channel #C1 with a constant corresponding to maximum value of #CH. Rate defines the sampling rate at which channel #C1 is saved.



- If no Rate is defined, the resulting channel #C1 gets @MainSamplingrate as frequency

6.3.14.5.2 MaxHoldwhiletrue

This function determines the maximum value of a SourceChannel as long as BoolChannel is > 0.



- The maximum value of the SourceChannel in the BoolChannel area is determined successively as long as the BoolChannel is > 0.

Syntax	Meaning
C1=MaxHoldWhileTrue(#SourceChannel, #BoolChannel)	Successive determination of maximum value of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel = 0 the resulting channel C1 carries the value of SourceChannel
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.5.3 MaxWhileTrue

This function determines the maximum value of a SourceChannel as long as BoolChannel is > 0.



- The maximum value of the SourceChannel in the BoolChannel-Area is held as long as the BoolChannel is > 0.

Syntax	Meaning
C1=MaxWhileTrue(#SourceChannel, #BoolChannel)	Holding the maximum value of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel = 0 the resulting channel C1 carries the value of SourceChannel
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.5.4 PosMaxWhileTrue

This function highlights the maximum value of a SourceChannel as long as BoolChannel is > 0 with a Boolean TRUE.



- The maximum value of the SourceChannel in the BoolChannel-Area is highlighted with a Boolean TRUE.

Syntax	Meaning
C1=PosMaxWhileTrue(#SourceChannel, #BoolChannel)	Highlighting the maximum value of SourceChannel as long as BoolChannel is > 0.



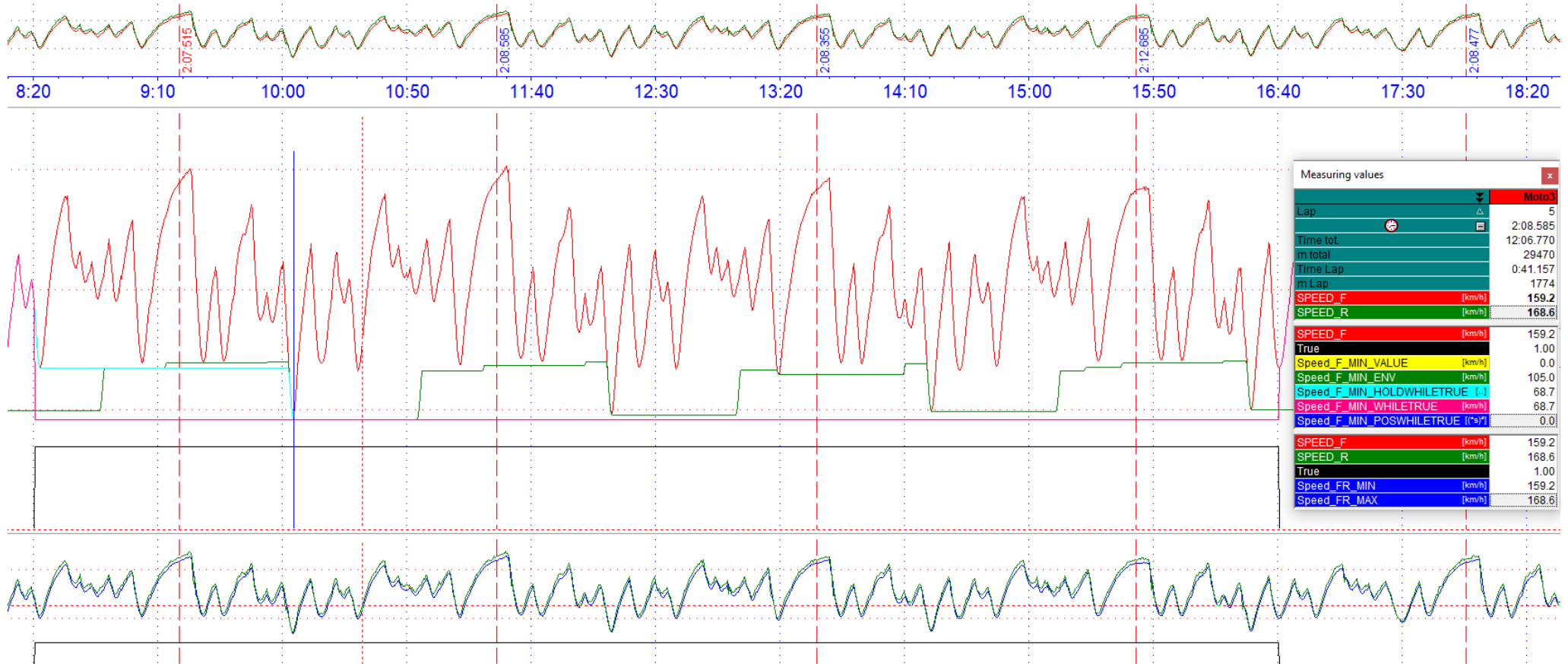
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.5.5 EnvMax

The EnvMax function searches a certain number of values of a certain channel to find the maximum within this range.

Syntax	Meaning
C1=EnvMax(#CH, samples number)	The EnvMax function searches a certain number of values of a certain channel to find the maximum within this range.

6.3.14.5.6 Different maxima functions



[MAX]

```

Speed_F_MAX_VALUE      =MaxValue(#Speed_F)
Speed_F_MAX_ENV        =EnvMax(#speed_F, 5000)
Speed_FR_MAX           =Max(#Speed_F, #Speed_R)
Speed_F_MAX_HOLDWHILETRUE=MAXHOLDWHILETRUE(#Speed_F, #True)
Speed_F_MAX_WHILETRUE  =MAXWHILETRUE(#Speed_F, #True)
Speed_F_MAX_POSWHILETRUE =POSMAXWHILETRUE(#Speed_F, #True)
  
```

This document is subject to change at 2D decision. 2D assumes no responsibility for any claims or damages arising out of the use of this document, or from the use of modules based on this document, including but not limited to claims or damages based on infringement of patents, copyrights or other intellectual property rights.

6.3.14.5.7 PeakPreView

The PeakPreView function determines a local maximum in an area defined by FallbackTreshholdNumber and holds the value of the local maxima and transfers it back to the previous local maxima.



- Before a new local maximum is found, the value of #CH must fall below the FallbackTreshhold

$$FallbackTreshhold = ValueOfLastLocalMaxima * FallbackTreshholdNumber$$

Syntax	Meaning
C1=PeakPreview(#CH, FallbackThresholdNumber)	Determine a local maximum of #CH in an area defined by FallbackThresholdNumber and transfers value of local maxima back to previous one.



- FallbackTreshholdNumber must be between 0 and 1

Example:

```
[PeakPreview]
Speed_F_PeakPreview      =PeakPreView(#SPEED_F, 0.5)
```



6.3.14.6 First value

The FirstValue function reads the first value of a given channel and creates a constant channel using this value.

Syntax	Meaning
C1=FirstValue(#CH, Rate)	Creates a channel #C1 with a constant corresponding to first value of #CH. Rate defines the sampling rate at which channel #C1 is saved.

6.3.14.7 Last value

The LastValue function reads the last value of a given channel and creates a constant channel using this value.

Syntax	Meaning
C1=LastValue(#CH, Rate)	Creates a channel #C1 with a constant corresponding to last value of #CH. Rate defines the sampling rate at which channel #C1 is saved.

6.3.14.8 AvgValue

The AvgValue function calculates the average value of a given channel and creates a constant channel using this value.

Syntax	Meaning
C1= AvgValue(#CH, Rate)	Creates a channel #C1 with a constant corresponding to average value of #CH. Rate defines the sampling rate at which channel #C1 is saved.

6.3.14.9 AverageWhileTrue

AvgWhileTrue function continuously calculates the average value of a SourceChannel if a BoolChannel is > 0.

Syntax	Meaning
C1=AvgWhileTrue(#SourceChannel, #BoolChannel)	Continuous calculation of average value of a SourceChannel if a BoolChannel is > 0.



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.10 Limit

This function uses only the channel value between the limits. Minimum and maximum limit can be fix parameters or channel values.



- If the value of the checked channel is below the minimum limit, this value will be used, if it is greater than the maximum limit, that value will be used.

Syntax	Meaning
C1=Limit(#CH, minLimit, maxLimit)	The value of the #CH will be used if its value is between the limits. Otherwise the value of the minimum or maximum limits will be used.



- The limits can be constants or channels.

6.3.14.11 Rising Edge

The RisingEdge function is used for boolean detection of rising edges.



- If subsequent value of a channel is greater than the current value, then the function's value is 1, otherwise 0.

Syntax	Meaning
C1=RisingEdge(#CH)	#CH is checked on rising edges

6.3.14.12 Falling Edge

The FallingEdge function is used for boolean detection of falling edges.



- If subsequent value of a channel is smaller than the current value, then the function's value is 1, otherwise 0
- If falling edge is used for subsequent functions like FillFromBool (6.3.14.14) please check position of Boolean detection and use Shift-function (6.3.9.3) if necessary!

Syntax	Meaning
C1=FallingEdge(#CH)	#CH is checked on falling edges

6.3.14.13 Zero Cross

The function ZeroCross is used to mark zero crossings of a signal boolean.

Syntax	Meaning
C1=ZeroCross(#CH)	#CH is checked on zero crossings



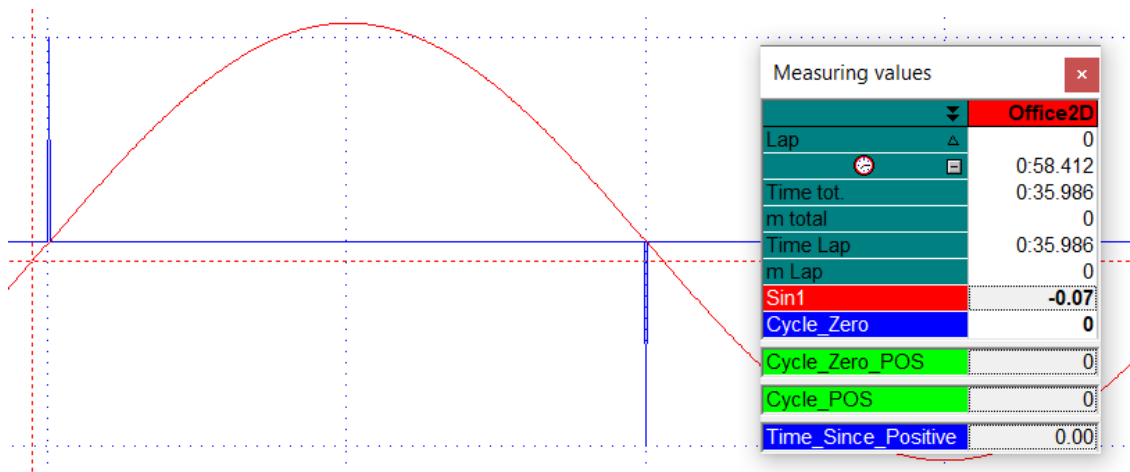
- With a zero crossing from the positive to the negative range the result of the ZeroCross-function = 1
- With a zero crossing from the negative to the positive range the result of the ZeroCross-function = -1

Example:

Marking zero crossing of a sinusoidal function.

[Cycle_Zero]

Result = ZeroCross(#Sin1)



6.3.14.14 FillFromBool

With this function the value of a SourceChannel will be continued from the moment the BoolChannel was last > 0.

Syntax	Meaning
C1=FillFromBool(#SourceChannel, #BoolChannel)	Take value from SourceChannel when BoolChannel is > 0

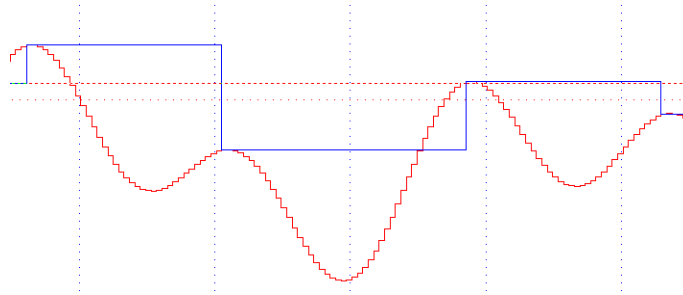


- Frequency of the resulting channel C1 corresponds to the frequency of BoolChannel.
- If BoolChannel is never > 0 inside the respective measurement, the resulting channel C1 permanently carries the **first** value of SourceChannel.

Example:

Analysis of local maxima

```
[LocalMax]
C1 = F'(#Sin)           ; Derivation of #Sin
C1 = ZeroCross(#C1)     ; Find ZeroCrossing of Derivation
C1 = If(#C1, =, 1, 0, #C1) ; Only Maxima
Result= FillFromBool(#Sin, #C1) ; FillwithnextBool
```



6.3.14.15 FillWithNextBool

With this function the value of a source channel will be continued from the moment the Boolean channel was next > 0.

Syntax	Meaning
C1=FillWithNextBool(#SourceChannel, #BoolChannel)	Take value from SourceChannel when BoolChannel was <u>last</u> > 0

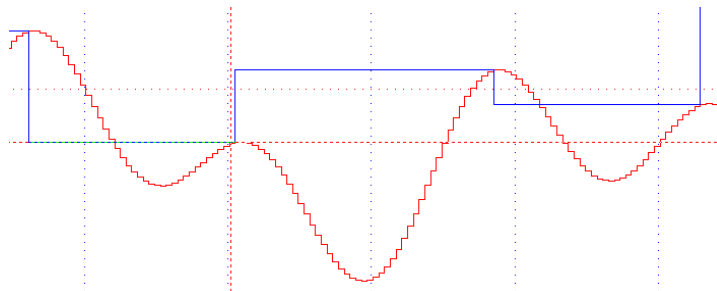


- Frequency of the resulting channel C1 corresponds to the frequency of BoolChannel.
- If BoolChannel is never > 0 inside the respective measurement, the resulting channel C1 permanently carries the value 0.

Example:

Analysis of local maxima

```
[LocalMax]
C1 = F'(#Sin) ; Derivation of #Sin
C1 = ZeroCross(#C1) ; Find ZeroCrossing of Derivation
C1 = If(#C1, =, 1, 0, #C1) ; Only Maxima
Result= FillwithnextBool(#Sin, #C1) ; FillwithnextBool
```



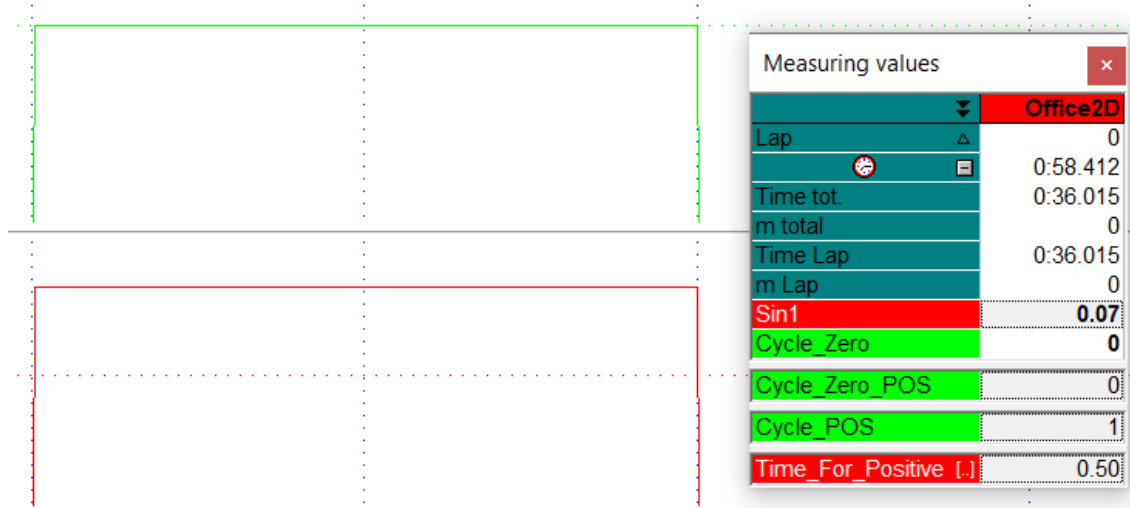
6.3.14.16 TimeForTrue

The TimeForTrue function displays the absolute time in seconds if the value of the channel is not zero. When the channel's value goes back to zero, the value of this calculated channel is zero as well.

Syntax	Meaning
C1=TimeforTrue(#CH)	As long as the value of the channel is bigger or smaller than 0, the absolute time the channel is not zero is displayed in seconds

Example:

```
[Time_For_Positive]
Result=TimeForTrue(#Cycle_POS)
```



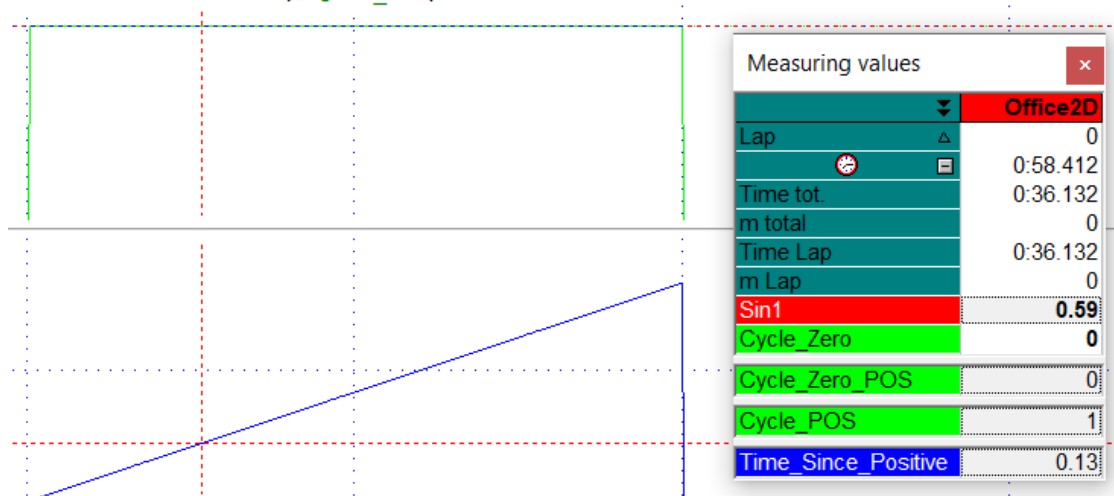
6.3.14.17 TimeSinceTrue

The TimeSinceTrue function counts the seconds if the channel value of a channel is not zero. When the channel's value goes back to zero, the value of this resulting channel is zero as well.

Syntax	Meaning
C1=TimeSinceTrue(#CHI)	As long as the value of the channel is bigger or smaller than 0, this time will be counted in seconds

Example:

```
[Time_Since_Positive]
Result=TimeSinceTrue(#Cycle_POS)
```



6.3.14.18 HoldWhileTrue

HoldWhileTrue function writes the value of SourceChannel to resulting channel C1 as long as BoolChannel is > 0.

Syntax	Meaning
C1=HoldWhileTrue(#SourceChannel, #BoolChannel)	Writes the value of SourceChannel to resulting channel C1 as long as BoolChannel is > 0.



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of the input channels

Example: Show value of *Distance2D* only as long as *IsTrack* is > 0.

[Distance2D_HoldWhileTrue]

Result = HoldWhileTrue(#Distance2D , #IsTrack)



6.3.14.19 ExpandWhileTrue

ExpandWhileTrue function holds the value of SourceChannel from rising edge of BoolChannel until falling edge of BoolChannel

Syntax	Meaning
C1=ExpandWhileTrue(#SourceChannel, #BoolChannel)	Holds the value of SourceChannel from rising edge of BoolChannel until falling edge of BoolChannel



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

Example 1: Hold first value of *Distance2D* only as long as *IsTrack* is > 0.

[Distance2D_ExpandWhileTrue]

Result = ExpandWhileTrue(#Distance2D , #IsTrack)

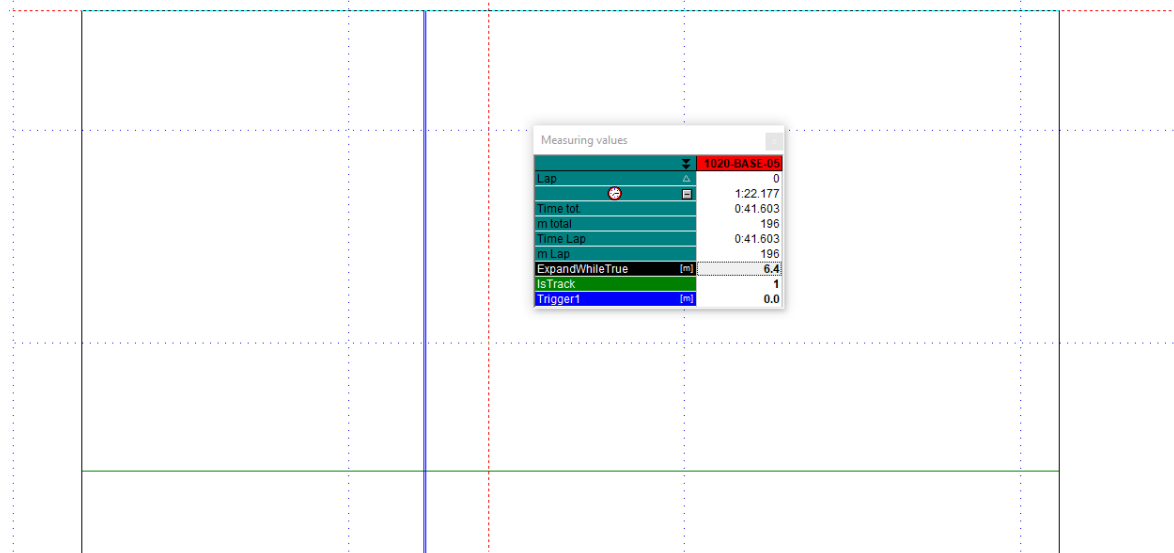


Example 2: Determine value of *Trigger1* as long as *IsTrack* > 0.

Trigger1=If(#Distance2D, =, 196, 6.4, 0)

[ExpandWhileTrue]

Result=ExpandWhileTrue(#Trigger1, #IsTrack)

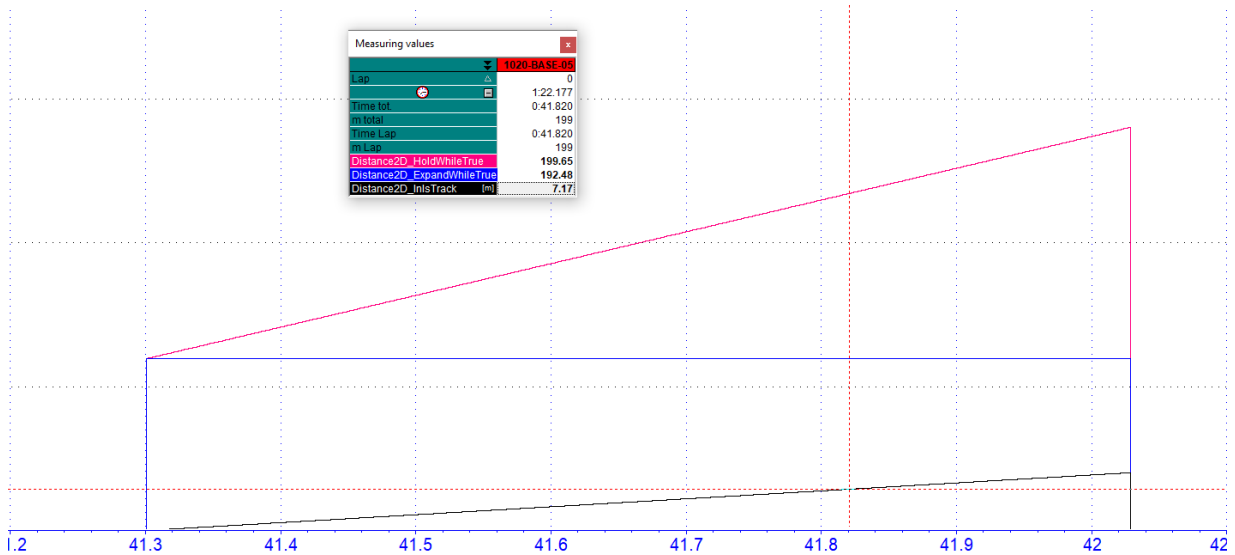


6.3.14.19.1 Combine HoldWhileTrue and ExpandWhileTrue

Example: Show the continuously travelled distance as long as #IsTrack is > 0.

[Distance2D_InIsTrack]

Result =-(#Distance2D_HoldWhileTrue, #Distance2D_ExpandWhileTrue)



6.3.14.20 DeltaWhileTrue

DeltaWhileTrue function calculates the difference between the values of SourceChannel from rising edge of BoolChannel to falling edge of BoolChannel

Syntax	Meaning
C1=DeltaWhileTrue(#SourceChannel, #BoolChannel)	Calculates the difference between the values of SourceChannel from rising edge of BoolChannel to falling edge of BoolChannel



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.21 InterpolateWhileFalse

The function InterpolateWhileFalse linearly interpolates the values of SourceChannel between falling and rising edge of BoolChannel. Additionally the interpolation can be made dependent on how long BoolChannel is < 1.



- WhileFalse-condition!



- This function can be used to eliminate short signal dropouts
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

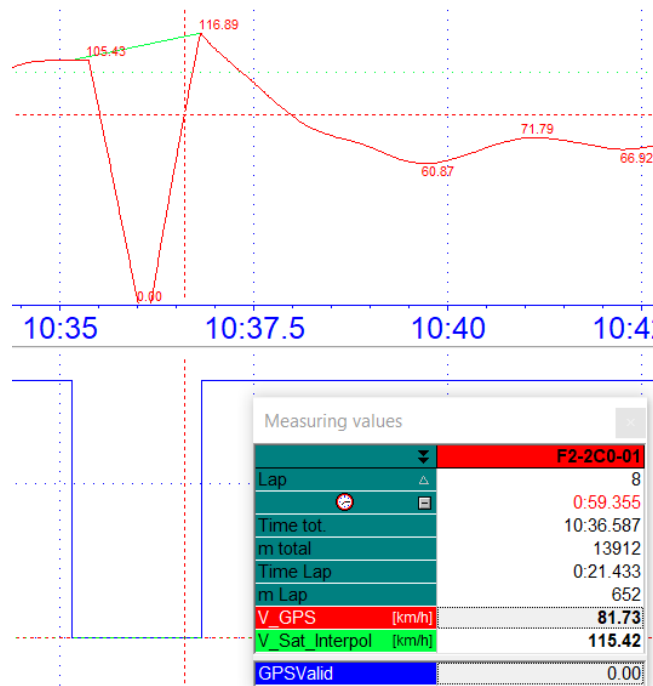
Syntax
C1=InterpolateWhileFalse(#SourceChannel, #BoolChannel)
C1=InterpolateWhileFalse(#SourceChannel, #BoolChannel, MaxGap)
Meaning
The function InterpolateWhileFalse linearly interpolates the values of SourceChannel between the falling and rising edge of BoolChannel. If additionally, a Value MaxGap is given, the interpolation only takes place if BoolChannel was not longer < 1 than the value of MaxGap is. MaxGap is specified in seconds!

Example:

Correction of the GPS channel #V_GPS in case of GPS signal dropouts which are shorter than 2 seconds due to e.g. tunnels or trees.

[V_Sat_Interpol]

Result=InterpolateWhileFalse(#V_GPS, #GPSValid,2)



The corrected, green channel #V_Sat_Interpol interpolates the signal at the 1.67 second drop of #GPS_Valid.



- This GPS-channel correction is already be implemented in AutoCal-File 2D_GPSAuto.CCF!

6.3.14.22 InterpolateWhileTrue

The function InterpolateWhileTrue linearly interpolates the values of SourceChannel between rising and falling edge of BoolChannel. Additionally, the interpolation can be made dependent on how long BoolChannel is > 0.



- This function can be used to eliminate value gaps that can result from signal failures
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

Syntax
C1=InterpolateWhileTrue(#SourceChannel, #BoolChannel)
C1=InterpolateWhileTrue(#SourceChannel, #BoolChannel, MaxGap)
Meaning
The function InterpolateWhileTrue linearly interpolates the values of SourceChannel between rising and falling edge of BoolChannel. If additionally, a Value MaxGap is given, the interpolation only takes place if BoolChannel was not longer < 1 than the value of MaxGap is. MaxGap is specified in seconds!

6.3.14.23 Countchanges

The function CountChanges increments the resulting channel C1 by 1 if the difference of SourceChannel compared to the next sample exceeds the StepSize-value.



- Only positive StepSize-values are possible!

Syntax	Meaning
C1=CountChanges(#SourceChannel, StepSize)	Incrementing the resulting channel C1 by 1 if the difference of SourceChannel compared to the next sample exceeds the StepSize-value.

6.3.14.24 VDVWhileTrue

VDVWhileTrue function calculates the Vibration Dose Value (VDV) of SourceChannel as long as BoolChannel is > 0.

Vibration Dose Value (VDV):

Vibration Dose is a parameter that combines the magnitude of vibration and the time for which it occurs to evaluate the impact of whole-body vibrations. The VDV depends on the integral of the acceleration in the fourth power over time and thus reacts more sensitively to the highest measured values, which are often due to shocks. Therefore, VDV is often used at comfort tests because it is suitable for deciding whether it is dealt with shock excitation.

$$VDV \text{ in } \left[\frac{m}{s^{1.75}} \right] = \sqrt[4]{ \int_0^T Acc_w^4(t) dt }$$

Syntax	Meaning
C1=VDVWhileTrue(#SourceChannel, #BoolChannel)	Calculating the Vibration Dose Value (VDV) of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

6.3.14.25 RMSWhileTrue

RMSWhileTrue function calculates the Root Mean Square (RMS) of SourceChannel as long as BoolChannel is > 0.

$$RMS = \sqrt{\frac{1}{n} * (x_1^2 + x_2^2 + \dots + x_n^2)}$$

Syntax	Meaning
C1=RMSWhileTrue(#SourceChannel, #BoolChannel)	Calculating the Root Mean Square (RMS) of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel is = 0, resulting channel C1 is also 0
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

Example:

Determine the Root Mean Square of one period of a sinusoidal signal with amplitude 1 and frequency 1 Hz.

[Variables]

```
Frequency=1000 ; in Hz
Amplitude=1 ; in V
Period=1 ; in Hz
```

[Sin]

```
; u(T) = U̇ * Sin(2PI*f*t)
C0 = Const(2, @frequency)
PI2 =*(#C0, @PI)
```

```
C0 = Const(1, @Frequency)
Time = I(#C0)
```

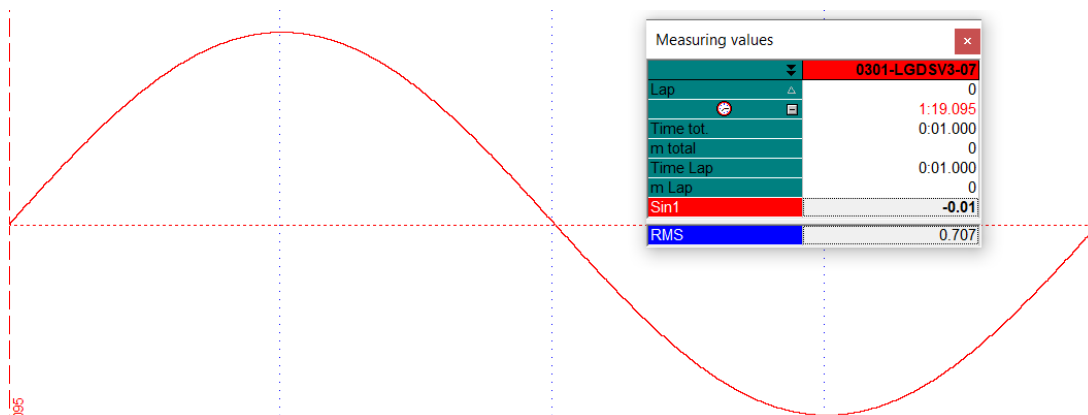
```
C0 =*(@period, #Time)
C1 =*(#PI2, #C0)
C2 = Sin(#C1)
Result=*(@amplitude, #C2)
```

[True]

```
Result=If(#Time, <, 1, 1, 0)
```

[RMS]

```
Result=rmswhiletrue(#Sin, #True)
```



6.3.14.26 MOVWhiletrue

MOVWhileTrue function calculates the Movement (MOV) of SourceChannel as long as BoolChannel is > 0.

The MOV command is a command specially developed by 2D-Datarecording with which a change of a signal can be evaluated by the movement coefficient.

The function sums up the absolute difference between two samples of the SourceChannel as long as BoolChannel is > 0. Therefore, a channel is created which is an indicator how a signal is changing over time.

$$MOV = \sum (|SourceChannel_{n+1} - SourceChannel_n|) = \int ABS(f'(SourceChannel_n))$$

This creates a channel which is an indicator how a signal is changing over time.

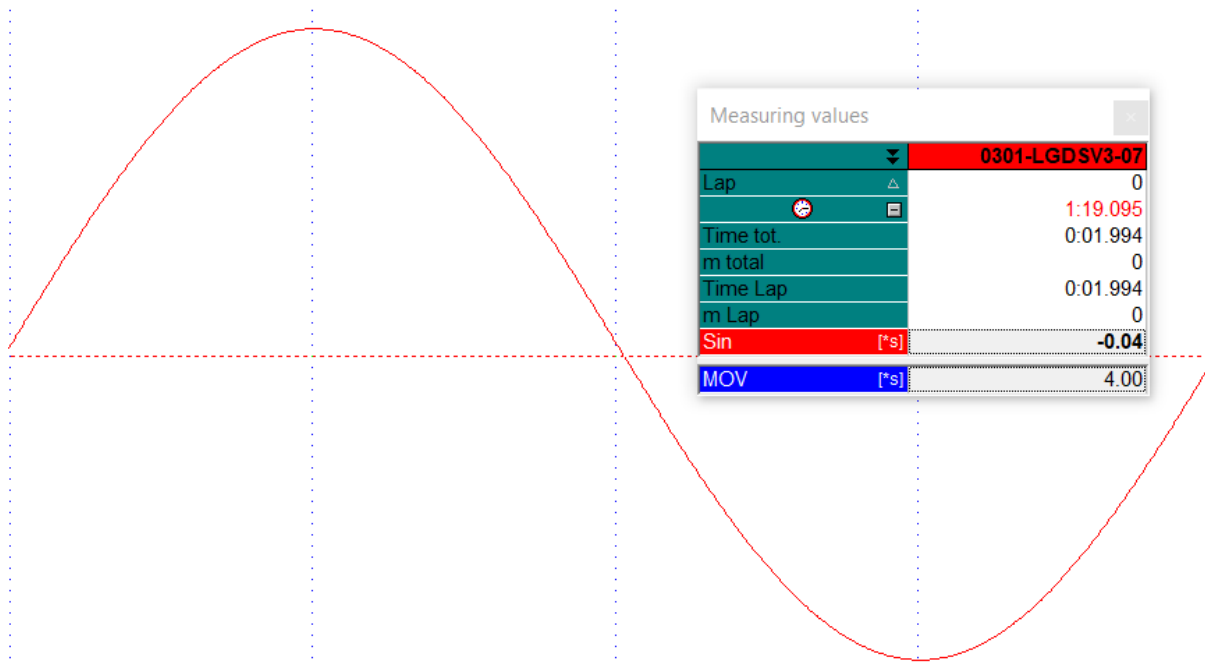
Syntax	Meaning
C1=MOVWhileTrue(#SourceChannel, #BoolChannel)	Calculating the Movement Value (MOV) of SourceChannel as long as BoolChannel is > 0.



- If BoolChannel is = 0, resulting channel C1 is also 0
- Resulting channel C1 is not reseted at a laptrigger
- Frequency of the resulting channel C1 corresponds to the fastest frequency of #SourceChannel

Example 1:

Determine the Movement value of one period of a sinusoidal signal with amplitude 1 and frequency 1Hz.



This example shows the ideal value of the MOV-function:

Sin	MOV
0 → 1	+1
1 → 0	+1
0 → -1	+1
-1 → 0	+1
	<hr/>
	= 4

Example 2:

Analysis of the throttle actuation of a driver at one point of a racetrack in different laps.



- For this purpose, the same measurement is loaded twice in the Analyzer and superimposed

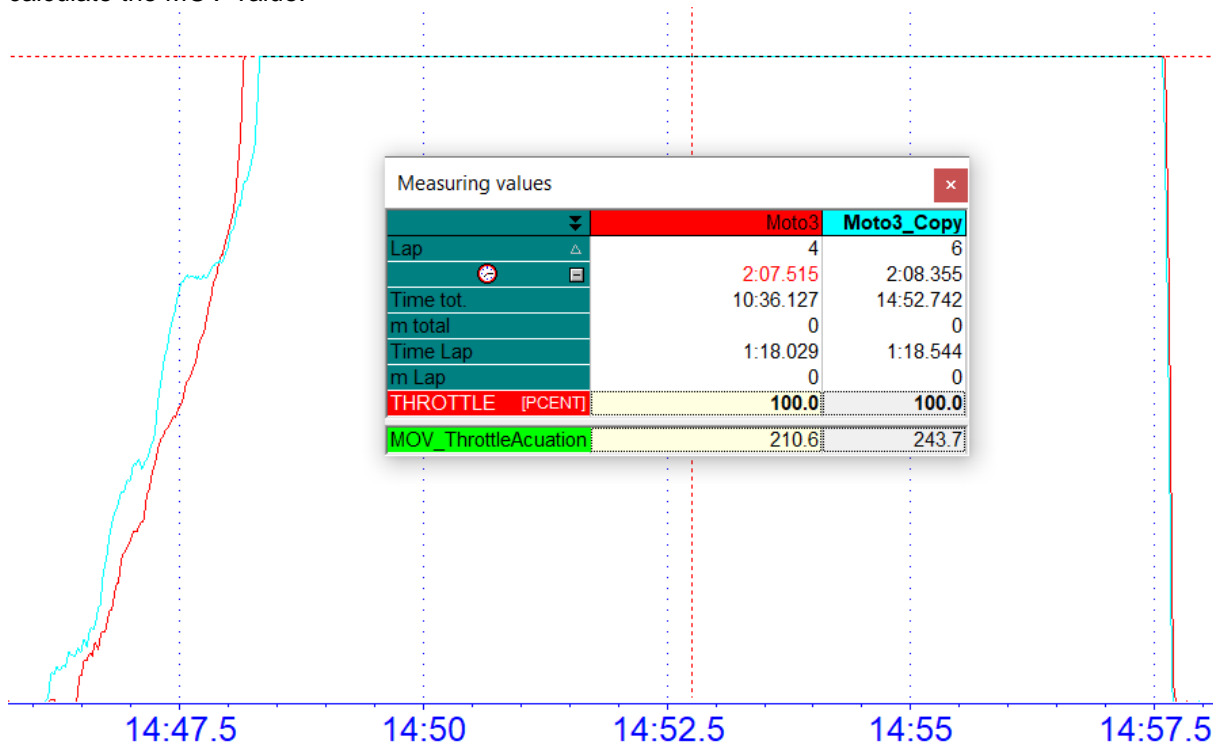
[True_ThrottleActuation]

Result = If(#Throttle, >, 0.6, 1, 0)

[MOV_ThrottleAcuation]

Result= MovWhileTrue(#Throttle, #True_ThrottleActuation)

Code explanation: At first a Boolean condition #True_ThrottleActuation is created which is TRUE if #Throttlet is bigger than 0.6%. This Boolean condition is used together with SourceChannel #Throttle to calculate the MOV-value.



The red measurement Moto3 shows a position of the fastest lap, while the blue measurement Moto3_Copy shows the same position in the following lap. In the fastest lap the driver has opened the throttle very definitely, while in the following lap the throttle was even closed a little bit during the acceleration phase.

These different opening sequences are reflected in the value of the MOV command, which is higher in the blue measurement Moto3_Copy due to the interim closing/delayed opening of the throttle grip than in the red measurement Moto3

For example, the throttle grip movement can be classified as better the closer the value of the #MOV_ThrottleActuation is to 200 (ideal value is 200 → throttle grip open and closed once).



- This command is also suitable for suspension adjustment, as it can be used to analyse the oscillation of the suspension due to a too wide opened rebound

6.3.15 Table functions

Most analogue sensors do have a linear behaviour. That means the output voltage of the sensor is direct proportional to the measured physical value. The physical value can be calculated by a simple rule of three:

$$PhysicalValue = Multiplier * Digits + Offset$$

Other sensors like NTC temperature sensors do not have a linear characteristic curve. In this case, sensor manufacturers provide a specific characteristic curve, which can be translated to a 2D-useable table giving the physical value for all measured voltages.

The table function is used to linearize the measured voltage values to the physical values.

Syntax	Meaning
C1=F(#CH, T(TableName.Ext))	#CH will be connected to the respective table



- If no extension (.Ext) is given to the table name, *TBL* is taken as default.

Example1:

Obtain the power of the engine with RPM using the table linking the power with RPM.

Example2:

Generate a linear channel from a non-linear one with the calibration sheet you get from the manufacturer (e.g. lambda sensor).

Storage location of table:

If no specific path to the respective table *TableName.EXT* is specified, the system searches in the respective event.

If a calculation is then carried out for the first time with this table, CalcTool copies the table into the MES directory of the measurement currently being used when executing the table function.

CalcTool accesses this table in the MES directory for all further executions!

If a modification is made to the table used, the table in question should be deleted from any MES directory already in use and the table replaced if necessary.



- if the table is not in the respective event, the known placeholders from chapter 3.3 can also be used.

6.3.15.1 2D-table function

The 2D-table function can be seen as the further development of the normal table function shown before, because the 2D-table connects a second channel to a table.

Syntax	Meaning
C1=F(#CH1, #CH2, T(TableName.Ext, IntType))	#CH1 and #CH2 will be connected to find corresponding value in table “ <i>TableName.Ext</i> ”

#CH1 corresponds to the first row of the table *TableName.Ext*

#CH2 corresponds to the first column of the table *TableName.Ext*

#C1 is the resulting table value and depends on the values of #CH1 and #CH2

IntType (Type of Interpolation)	
LIN	Linear Interpolation



- This function necessarily needs the CSV as table format!
- Column separation character must be a semicolon “;”
- Decimal separation character must be a point “.” (½ = 0.5)



- CSV files can be created with simple text editor or with *MS Excel™*

Example:

The radius of a wheel of a motorcycle is a function depending on the banking angle of the bike and the speed. The higher the banking angle is, the lower is the effective radius of the wheel (in fact the size of the wheel is the same, but the distance between the axle centre and the point where the wheel touches the ground becomes smaller).

The faster a wheel rotates, the higher is the rotational energy and the tire expands a bit.

The 2D-table formula for this calculation is:

```
[RadiusOfWheel]
Result=F(#Speed,#Banking,T(WheelRadius.CSV,Lin))
```

This is an example for a table returning the radius of a wheel as a function of the speed and the banking angle of a bike:

	#Speed							
	0	50	100	150	200	250	300	350
0	315	317	320	322.5	325	327	329	330
10	309	311	314	316.4	318.8	320.7	322.7	323.7
20	301	302.9	305.8	308.2	310.5	312.5	314.4	315.3
30	295	296.9	299.7	302	304.4	306.2	308.1	309
40	288	289.8	292.6	294.9	297.1	299	300.8	301.7
50	285	286.8	289.5	291.8	294	295.9	297.7	298.6
60	283	285.5	288	290	392.5	394.3	296.1	297
#Banking	#RadiusOfWheel							

For example, radius of the wheel at a speed of 100 km/h and an angel of 20° is 305.8.

When *CalcTool* tries to find a value depending on speed and banking angle, it searches for the columns and row that frames speed and the banking angle.

Then a linear interpolation is performed to calculate the value.

Example1:

#Speed is assumed as 110 km/h

#Banking is assumed as 33°

	0	50	100	150	200	250	300	350
0	315	317	320	322.5	325	327	329	330
10	309	311	314	316.4	318.8	320.7	322.7	323.7
20	301	302.9	305.8	308.2	310.5	312.5	314.4	315.3
30	295	296.9	299.7	302	304.4	306.2	308.1	309
40	288	289.8	292.6	294.9	297.1	299	300.8	301.7
50	285	286.8	289.5	291.8	294	295.9	297.7	298.6
60	283	285.5	288	290	392.5	394.3	296.1	297

The 2D-table function internally calculates a percentage graded (linear interpolated) table to find the intermediate value:

	100	110	120	130	140	150
30	299.7	300.16	300.62	301.08	301.54	302
31	298.99	299.45	299.91	300.37	300.83	301.29
32	298.28	298.74	299.2	299.66	300.12	300.58
33	297.57	298.03	298.49	298.95	299.41	299.87
34	296.86	297.32	297.78	298.24	298.7	299.16
35	296.15	296.61	297.07	297.53	297.99	298.45
36	295.44	295.9	296.36	296.82	297.28	297.74
37	294.73	295.19	295.65	296.11	296.57	297.03
38	294.02	294.48	294.94	295.4	295.86	296.32
39	293.31	293.77	294.23	294.69	295.15	295.61
40	292.6	293.06	293.52	293.98	294.44	294.9

The resulting radius of the wheel would be 298.03.



- In the following figure, it can be seen how the table looks, if it is created via an Editor:

```
;0;50;100;150;200;250;300;350;
0;315;317;320;322.5;325;327;329;330;
10;309;311;314;316.4;318.8;320.7;322.7;323.7;
20;301;302.9;305.8;308.2;310.5;312.5;314.4;315.3;
30;295;296.9;299.7;302;304.4;306.2;308.1;309;
40;288;289.8;292.6;294.9;297.1;299;300.8;301.7;
50;285;286.8;289.5;291.8;294;295.9;297.7;298.6;
60;283;285.5;288;290;392.5;394.3;296.1;297;
```

Example2:

#Speed is assumed as 360 km/h

#Banking is assumed as 60°

The resulting radius of the wheel would be 297.

6.3.15.2 2D-table search function

CalcTool provides a kind of inverse function for the 2D-table function. It uses the same type of table files like the function above.

Syntax
C1=T2DSearch(#CH1, #CH2, T(TableName.Ext, IntType), [SearchDirection])
Meaning
#CH1 and #CH2 will be used to find corresponding value in the table "TableName.Ext"

#CH1 corresponds to the table values of the table TableName.Ext
 #CH2 corresponds to the first column of the table TableName.Ext
 #C1 corresponds to the first row of table TableName.Ext and is approximated by #CH1 and #CH2

The T2DSearch function interpolates linearly between given values.

IntType (Type of Interpolation)	
LIN	Linear Interpolation

SearchDirection	
Left	Searching table from left to right
Right	Searching table from right to left

Example:

```
[Speed]
Result=T2DSearch(#RadiusOfWheel,#Banking,T(WheelRadius.CSV,Lin))
```

#Banking is assumed as 33°.
 #RadiusOfWheel is assumed as 298.03mm.

	0	50	100	150	200	250	300	350
0	315	317	320	322.5	325	327	329	330
10	309	311	314	316.4	318.8	320.7	322.7	323.7
20	301	302.9	305.8	308.2	310.5	312.5	314.4	315.3
30	295	296.9	299.7	302	304.4	306.2	308.1	309
40	288	289.8	292.6	294.9	297.1	299	300.8	301.7
50	285	286.8	289.5	291.8	294	295.9	297.7	298.6
60	283	285.5	288	290	392.5	394.3	296.1	297

The corresponding #Speed is 110 km/h.

6.3.15.3 3D-table function

Like the 2D-table function *CalcTool* provides a 3D-table function.

This function can be used similar to the 2D table function, the only difference is that the table used depends on the value of the third channel #CH3.

Syntax	Meaning
C1= F(#CH1, #CH2, #CH3, T(TableName.Ext, IntType))	#CH1 and #CH2 will be used to find corresponding value in through #CH3 defined table TableName.Ext

#CH1 corresponds to the first row of the table TableName.Ext

#CH2 corresponds to the first column of the table TableName.Ext

#CH3 corresponds to the name of the used table

#C1 is the resulting table value and depends on the values of #CH1 and #CH2 and the used table

IntType (Type of Interpolation)	
LIN	Linear Interpolation

Example:

The torque that an engine delivers to a wheel is dependent on the selected gear, the engine revolutions, and the throttle position.

[Torque]

```
Result=F(#Gear,#RPM,#Throttle ,T(Torque_at_*.CSV,Lin))
```

You can see in the example that the filename of the table contains an asterisk ("**"). Depending on the value of channel #throttle a different table is used.

Different tables can be used:

```
Torque_At_0.CSV
Torque_At_20.CSV
Torque_At_40.CSV
Torque_At_50.CSV
Torque_At_60.CSV
Torque_At_70.CSV
Torque_At_80.CSV
Torque_At_90.CSV
Torque_At_100.CSV
```

Depending on the throttle grip positions 0, 20, 40, 50, 60, 70, 80, 90 and 100 %., the respective table is then used for the table calculation

CalcTool makes a linear interpolation for each value of #Throttle, #Gear and #RPM.

6.3.16 Changing the storage type

Reasons why the data type must be changed:

- Other functions expect word channel for input
- Save memory on hard disk

The following data types are usable:

- Word
- Binary
- LongInt
- Single
- Double
- Signed
- Unsigned



- Most channels in a 2D system are recorded as 16-bit integer values (what is called a word) combined
- Channels created by the calculation tool are stored as float values (single precision with 32 bit) which need the double space on the hard disk of the computer.
- Data type can be seen in Formel editor (Analyzer → Functions → Formel Editor)



- Changing data type might lowers precision of a channel!

6.3.16.1 WORD-function

If the Word-function is used with only one parameter (a channel), *CalcTool* determines the minimum and the maximum value of the original channel. The formula is calculated that the minimum is transformed to 0 and the maximum to 65535 (maximum word).

In some cases, it is helpful to use other values. If the signal has spikes you might use your own border values.



- Values higher than the upper border are set to the upper border value.
- Values lower than the lower border are set to the lower border value.

The third possibility to use the Word-function is to give the step (the accuracy) which should be used for one digit. The minimum value for the resulting channel is the smallest value of the input channel and the maximum 65535 multiplied with this step added to that minimum.

Syntax	Meaning
C1=Word(#CH)	Changing data type of #CH to WORD and set limits of resulting channel to minimum and maximum of channel
C1=Word(#CH, LowerBorder, UpperBorder)	Changing data type of #CH to WORD and set limits of resulting channel to LowerBorder and UpperBorder
C1=Word(#CH, Step)	Changing data type of #CH to WORD and set limits of resulting channel to minimum and [65535 * Step + minimum]

Example:

[Speed_WORD]

Result=Word(#Speed, 0.01)

The resulting channel has an accuracy of 0.01, the minimum value is 0 and the maximum is 655.35.

6.3.16.2 BINARY-function

the binary command is very similar to the word command and can be seen as an extension, because with binary borders and accuracy of a channel can be set simultaneously.

Syntax	Meaning
C1=Binary(#CH, Range)	Changing data type of #CH to WORD and determination of accuracy with actual channel borders
C1=Binary(#CH, Range, LowerBorder, UpperBorder)	Changing data type of #CH to WORD and determination of accuracy with respective channel borders



- A Binary-function with the range 65536 (0...65535) has the same result as a word function would have.



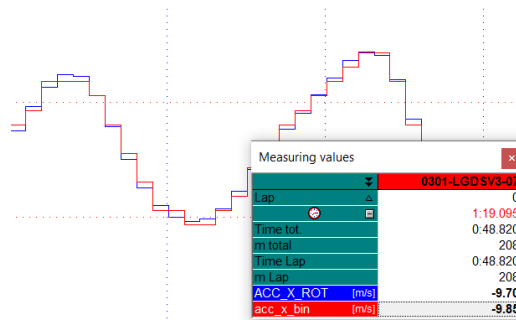
- Values higher than the upper border are set to the upper border value.
- Values lower than the lower border are set to the lower border value.

Example:

[Acc_x_Bin]

Result=Binary(#Acc_x_rot, 1024, -160, 160)

All values outside the borders -160 and +160 are set to the respective border values and the resulting channel is a Word channel with an accuracy of $320/1024=0.03125$.



6.3.16.3 Long Int

Set data type to LongInt.

Syntax	Meaning
C1=LongInt(#CH)	Changing data type of #CH to LongInt and set limits of resulting channel to minimum and maximum of channel
C1=LongInt(#CH, LowerBorder, UpperBorder)	Changing data type of #CH to LongInt and set limits of resulting channel to LowerBorder and UpperBorder
C1=LongInt(#CH, Step)	Changing data type of #CH to LongInt and set limits of resulting channel to minimum and $[65535 * Step + minimum]$

6.3.16.4 Single-precision float

Set data type to single-precision float (32-bit).

Syntax	Meaning
C1=Single(#CH)	Changing data type of #CH to single-precision float

6.3.16.5 Double precision float

Set data type to double-precision float (64-bit).

Syntax	Meaning
C1=Double(#CH)	Changing data type of #CH to double-precision float

6.3.16.6 Signed function



- Input channel must be an unsigned integer channel, that means Byte, Word or Double Word (=DWord) format

Syntax	Meaning
C1=Signed(#CH)	Changing range of #CH from unsigned to signed

Examples:

Type	Signed range		Unsigned range
Byte:	0...255	→	-127...128
Word:	0...65535	→	-32767...+32768
DWord	0...4294967295	→	-214748367...+214748368

6.3.16.7 UnSigned function



- Input channel must be an unsigned integer channel, that means Byte, Word or Double Word (=DWord) format

Syntax	Meaning
C1= UnSigned(#CH)	Changing range of #CH from signed to unsigned

Examples:

Type	Unsigned range		Signed range
Byte:	-127...128	→	0...255
Word:	-32767...+32768	→	0...65535
DWord	-214748367...+214748368	→	0...4294967295

6.3.16.8 Changing byte-order of channel

Some 2D data acquisition systems provide the possibility to record CAN-bus identifier which transfer data information between vehicle systems like ECU or ABS. To record this transferred data, byte-information must be set in communication software to determine which data from CAN-bus should be recorded. If the byte order of recorded channels is incorrect the resulting channel will be unusable.



- The SWAP-function was designed to change the byte order of CAN-bus channels
- Input channel with data type Word or LongInt are expected as input

Syntax	Meaning
C1= Swap(#CH)	Changing byte-order of channel #CH

6.3.17 Additional channel information

6.3.17.1 Dimension

To change the dimension of a channel, use the *Set*-function.



- Only at integration and derivation of speed and distance, channel dimension is set automatically
- At all other calculations, the resulting channel receives the dimension of first input channel!
- Dimension can be seen in channel list and measurement mode

Syntax	Meaning
C1=SET(DIM='Value')	Set dimension of channel #C1 to <i>Value</i>

Example: Set the dimension of a calculated gear channel to 'Gear'

```
Gear=Set (DIM='Gear')
```

Nr	Channel	Visible	Scaling	Overly	Lower	Upper	Color	Reduct	All	Key
60	 Gear [GEAR]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		0	6.5	 	AVG	<input type="checkbox"/>	1

Gear
Currently engaged gear

6.3.17.2 Sensorinfo

To add the additional information to a channel, use the *Set*-function.




- Only at integration and derivation of speed and distance, channel dimension is set automatically
- At all other calculations, the resulting channel receives the dimension of first input channel!
- Sensorinfo is displayed when cursor is put on respective channel in channel list!

Syntax	Meaning
C1=SET(Sensorinfo='Text')	Set additional Sensorinfo of channel #C1

Example: Set the Sensorinfo of a calculated gear channel to 'Gear'

```
Gear=Set (Sensorinfo='Currently engaged gear')
```

Nr	Channel	Visible	Scaling	Overly	Lower	Upper	Color	Reduct	All	Key
60	 Gear [GEAR]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		0	6.5	 	AVG	<input type="checkbox"/>	1

Gear
Currently engaged gear

6.3.18 Channel handling

6.3.18.1 Deleting channels

Function to delete channels, which were created by a previous calculation.

Syntax	Meaning
Delete(#CH)	Delete selected #CH



- It is not possible to delete channels, which were recorded by the data acquisition system



- Wildcard character "*" can be used → Delete(*) deletes all channels created by CalcTool

6.3.18.2 Hiding channels

Function to hide a channel. A hidden channel is present but not visible in the software.

Syntax	Meaning
Hide(#CH)	Hide selected #CH

6.3.18.3 Unhiding channels

Function to unhide a hidden, but still present channel.

Syntax	Meaning
Unhide(#CH)	Unhide selected #CH

6.3.18.4 Restoring channels

If a originally recorded channel was overwritten by NewResult (see 0), the originally recorded channel can be restored with this command.

Syntax	Meaning
Restore(#CH)	Restore selected #CH

6.3.18.5 NoOperation

Linking all values of a channel without further processing to the resulting channel.

Syntax	Meaning
C1=NoOp(#CH)	Link #CH to resulting channel C1

6.3.19 Laps



- Laptimes can be created either during by special logger settings or afterwards during data evaluation with the following functions!
- Laptriggers can also be useful for development purposes, e.g. if Laptriggers are generated for certain events like gear changes. Page-up and page-down keys can then be used to jump back and forth between the Laptriggers and thus every gear change.



- For more information about how to create Lap- and Section-times, please see the Manual *GPS_Laptime* on your website:
<http://2d-datarecording.com/downloads/manuals/>

6.3.19.1 Create lap trigger by channel

If there are no recorded lap triggers in measurement, they can be created afterwards.



- A trigger is set at each position where the value of the input channel changes! Summarizing boolean events with the sum function is particularly useful!



- Those laptriggers can be reseted via *Analyzer* → *File* → *Restore original laptrigger*

Syntax	Meaning
C1=SetLapTrigger(#CH)	Creates lap triggers depending on #CH
C1=SetLapTrigger(#CH, Timeout)	Creates lap triggers depending on #CH and with a defined Timeout in seconds in which no other lap trigger will be set

6.3.19.2 Create lap trigger by line

If there are no recorded lap triggers in measurement, they can be created afterwards.



- Function compares current Lat- and Lon-Positions with first in Event-folder founded line file

Syntax	Meaning
C1= CreateLapTriggerByLine(#CH_Lat, #CH_Lon, Radius)	Creates lap triggers depending on the distance of Latitude/Longitude channels to line file entries in respect to Radius

6.3.19.3 Create Laptimes

With the function *MakeLaptimeCh* a channel is created which contains the laptimes in seconds.



- For creating a laptime channel, lap trigger must be available!
- Laptime always corresponds to the previous lap!

Syntax	Meaning
C1=MakeLaptimeCH(Rate)	Creating a channel with the laptimes in seconds of the previous lap with defined Rate

Example:

Lap	Actual laptime	Channel value
1	2:25:210	0
2	1:59.482	145.2
3	1:56.795	119.5

6.3.20 Sections



- Lap can be divided into sections for more precise evaluation.
- Section times can be created either during by special logger settings or afterwards during data evaluation with the following functions!



- To create Sections, lap triggers must already be available!



- For more information about how to create Lap- and Section-times, please see the Manual *GPS_Laptime* on your website:
<http://2d-datarecording.com/downloads/manuals/>

6.3.20.1 Create section trigger by channel

If there are no recorded section triggers in measurement, they can be created afterwards.



- A trigger is set at each position where the value of the input channel changes! Summarizing boolean events with the sum function is particularly useful!



- Those section triggers can be reseted via *Analyzer* → *File* → *Restore original section trigger*

Syntax	Meaning
C1=SetSecTrigger(#CH)	Creates section triggers depending on #CH
C1=SetSecTrigger(#CH, Timeout)	Creates section triggers depending on #CH and with a defined Timeout in seconds in which no other section trigger will be set

6.3.20.2 Create Sectiontimes

With the function *MakeSectimeCh* a channel is created which contains the section times in seconds.



- For creating a section time channel, section trigger must be available!
- Section time always corresponds to the previous section!

Syntax	Meaning
C1= MakeSectimeCh(Rate)	Creating a channel with the section times in seconds of the previous section with defined Rate.

6.3.21 Bike and car physical formulas

6.3.21.1 Speed

The logger calculates the speed by using the digital signal from the speed sensor, the tire-dimension and the number of pulses for the turning wheel.

Slow wheel speeds cause a problem of inaccuracy in the measurement of wheel speed. This inaccuracy at low wheel speeds is corrected by the command *Speed_L4* when generating the speed via the wheel speed sensor principle.

Syntax	Meaning
C1=Speed_L4 (#CH)	#CH: input channel (Normally a speed channel)



- Do not use this function on filtered values
- Do not use this function in combination with tables

Example:

Speed_Rear = Speed_L4(#VRear)

6.3.21.2 Speed2

This function is used to calculate a major speed channel from two separately measured speeds. The result is the average of the two speeds. If the difference between the two channels is greater than 10%, the resulting value is the greater signal.

Syntax	Meaning
C1=Speed2 (#CH1,#CH2)	Input channels #CH1 and #CH2 (Normally two speed channels)

Example:

```
[Speed2]
Result=Speed2(#VFront, #VRear)
```



- Do not use this function on filtered values.
- Do not use this function in combination with tables
- Use this function only with speed-channels

6.3.21.3 Slip

This function allows you to calculate the percentage of the vehicle's slip between front and rear wheel.



- Front and rear speed must be provided for this function!

Syntax	Meaning
C1=Slip (#CH_Front, #CH_Rear)	Calculating percentage of vehicles slip between front and rear wheel

6.3.21.4 Banking



- This function must be used to calculate the banking angle of the bike with the gyro channel.

The gyro measures the banking angle speed of the bike. So, if the result of the integration of this signal is the inclination of the bike. But with this kind of sensor, a small error can lead to a totally wrong result. To correct this, the function integrates the gyro's signal with a linear regression to avoid that the result is modified by the error explained above.

For example, considering we've got a gyro measuring a range of 300° (from -150° to 150°) with 4095 bits. So, 0° will be at 2047 or 2048 bits. This means there is an error of ½ bit.

Assuming a lap lasts 2 minutes and sampling rate is 50 Hz. Therefore, there are 6000 samples per lap. As mentioned above, each sample contains an error of ½ bit.

Thus, the total error is ½ * 6000 = 3000 bits.

But for 1 bit you have approximately 0.1° (300/4095) of error.

Finally, your error will reach 300° by lap.

Syntax	Meaning
C1=Banking (#AGyro, P1)	Respective Inclination-Gyro-Channel #CH. P1 is the angle of the bike on the start/finish line.



- Use 0 for P1 if bike is straight on finish line

6.3.21.5 BANK2ACC function

It is not worth measuring lateral acceleration on a bike.

However, to use the racetrack calculation function (*Analyzer* → *Functions* → *Circuit*), a "virtual" lateral acceleration is required.

This function calculates the lateral acceleration of the bike from the inclination of the bike calculated with banking function.

Syntax	Meaning
C1= BANK2ACC (#Abanking)	#Abanking is the channel where the inclination of the motorcycle is calculated with the banking channel

6.3.21.6 Roll angle

This function calculates the roll angle of a bike from two accelerometers and two gyros.



- The roll angle describes the movement around the x-axis (longitudinal direction of the vehicle).

Syntax	Meaning
C1= RollAngle (#AccY, #AccZ, #GyroX, #GyroZ)	This function calculates the roll angle

6.3.22 Execute external programs



- In chapter 3.3 an overview over different path-placeholders can be found which can replace ... in the path calls like ... \FileName.BAT

6.3.22.1 Execute

By using Execute-function, external CommandLine-programs and thus Batch-Files can be executed via CalcTool.



- When executing the Execute function the following CalcTool calculations are stopped as long as the external program is executed!
- For every *Execute*-command a new [Group] must be used!

Syntax	Meaning
Execute(CommandLine , Param1, ...)	Start external program defined by Commandline Parameters to pass to the program are given separated by commas



- If additional functions are required which are not already included in our Software, please contact us via Info@2D-datarecording.com

Example:

At every execution of a CAL-file, the file C:\ECU\CurrentSetting.bat must be copied to the respective event folder.

[RunExternal]

Execute(c:\Race\CopyEcuFile.bat, <MesDir>)

When executing the CAL-File which contains the previous call, first *CalcTool* replaces the placeholder <MesDir> (see 3.3) and then passing over the respective path to the Batchfile *CopyEcuFile.bat*. Inside the handed over path is implemented with %1. A second handover parameter would be passed over with %2.

CopyEcuFile.bat:

copy C:\ECU\CurrentSetting.bat %1

6.3.22.2 ExecuteNoWait

By using Execute-function, external CommandLine-programs and thus Batch-Files can be executed via CalcTool.



- When executing the ExecuteNoWait function the following CalcTool calculations will not be stopped as long as the external program is executed!
- For every *ExecuteNoWait*-command a new [Group] must be used!

Syntax	Meaning
ExecuteNoWait(CommandLine, Param1, ...)	Start external program defined by Commandline Parameters to pass to the program are given separated by commas



- If additional functions are required which are not already included in our Software, please contact us via Info@2D-datarecording.com

6.3.23 Execute CAL-files

6.3.23.1 FlushFiles

Syntax	Meaning
FlushFiles	At point in code the already calculated channels of the calculation file will be outputted during execution



- Can be helpful at calling external programs from CAL-file (see 6.3.22)

6.3.23.2 FlushQuiet

Syntax	Meaning
FlushQuiet	The command described above is executed without user message.

6.3.23.3 QuietMode

Disabling the report function shown below, when executing a CAL-file.

```
Calculating channels 0314-TEM2_FL-02...
Current group: Video_Time_8_1NotExists
Saving channel Video_Time_8_1
```

Syntax	Meaning
QuietMode	Disabling the report function.



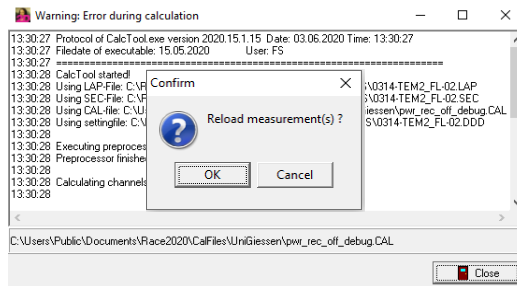
- Command can be called anywhere in the code and is valid for the whole CAL-file

6.3.24 Error handling

6.3.24.1 Show no errors

Disabling the error report function shown below, when an error occurs at executing a CAL-file.

Syntax	Meaning
ShowNoErrors	Disabling the <u>error</u> report function.



- Use only, when sure, that no errors occur!



- Command can be called anywhere in the code and is valid for the whole CAL-file

When an error occurs in the calculation it sometimes makes no sense to continue calculating after the error, because all following calculations depend on the error free results of previous lines.

For this purpose, *CalcTool* provides three different abort commands.

6.3.24.2 Abort calculation of group when an error occurred

Syntax	Meaning
OnErrorExitGroup	Stop executing the calculation of the <u>current group</u> if an error occurred

The calculation of the current group is aborted when an error occurred.

Channels calculated before are saved.

CalcTool continues with the next group.

6.3.24.3 Abort whole calculation when an error occurred

Syntax	Meaning
OnErrorExitTotal	Stop executing the calculation of the <u>whole CAL-file</u> if an error occurred.

The calculation of the current group is aborted when an error occurred.

Channels calculated before are saved.

CalcTool does not continue with the next group.

6.3.24.4 Abort whole calculation

Syntax	Meaning
ExitTotal	Stop executing the calculation of the <u>whole CAL-file</u> .

Calculating the current group is aborted even if there was no error before this command.

Channels calculated before are saved.

CalcTool does not continue with the next group.

7 Table of all calculation functions

+	C1=#CH1, #CH2)
	C1=#CH, Constant)
-	C1=#CH1, #CH2)
	C1=#CH, Constant)
*	C1=#CH1, #CH2)
	C1=#CH, Constant)
/	C1=#CH1, #CH2)
	C1=#CH, Constant)
{\$!	{\$! ...\FileName.EXT}
{\$!	{\$IfExists(#CH) ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfLaptimesExist ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfLicenceNameContains('Name') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotExists(#CH) ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotLaptimesExist ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotLicenceNameContains('Name') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotSpecIs(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotSpecValueContains(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfNotSpecValueExists(Group.Entry) ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfSpecIs(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfSpecValueContains(Group.Entry, 'Value') ...\FileName.CAL, P(P1, P2, ...)}
{\$!	{\$IfSpecValueExists(Group.Entry) ...\FileName.CAL, P(P1, P2, ...)}
Abs	C1=Abs(#CH)
And	C1=AND(#CH1,#CH2)
	C1=AND(#CH, Constant)
ArcCos	C1=arccos(#CH)
ArcSin	C1=arcsin(#CH)
ArcTan	C1=arctan(#CH)
ArcTan2	C1=arctan2(#CH1, #CH2)
AvgValue	C1= AvgValue(#CH, Rate)
AvgWhileTrue	C1=AvgWhileTrue(#SourceChannel, #BoolChannel)
Bank2Acc	C1=BANK2ACC (#Abanking)
Banking	C1=Banking (#AGyro, P1)
Binary	C1=Binary(#CH, Range)
	C1=Binary(#CH, Range, LowerBorder, UpperBorder)
ChannelArray	C1=ChannelArray('Name_VAR1', 'Name_VAR2', 'Name_VAR3')
Const	C1=Const(Value, Rate)
Cos	C1=cos(#CH)
CountChanges	C1=CountChanges(#SourceChannel, StepSize)
CountValue	C1=CountValue(#CH, value, tolerance)
CreateLapTriggerByLine	C1= CreateLapTriggerByLine(#CH_Lat, #CH_Lon, Radius)
DCos	C1=dcos(#CH)
DecWhileTrue	C1=DecWhileTrue(#SourceChannel, #BoolChannel)
Delete	Delete(#CH)

DeltaWhileTrue	C1=DeltaWhileTrue(#SourceChannel, #BoolChannel)
Derivate	C1=F'(#CH)
	C1=Derivate(#CH)
Div	C1=Div(#CH1, #CH2)
	C1=Div(#CH, Constant)
Double	C1=Double(#CH)
DSin	C1=dsin(#CH)
DTan	C1=dtan(#CH)
EnterChannel	C1=EnterChannel('Text shown in dialog')
	C1=EnterChannel('Text Text shown in dialog', #CH)
EnterValue	C1=EnterValue('Text shown in dialog')
	C1=EnterValue('Text shown in dialog', Value)
EnvMax	C1=EnvMax(#CH, samples number)
EnvMin	C1=EnvMin(#CH, samples number)
Execute	Execute(CommandLine , Param1, ...)
ExecuteNoWait	ExecuteNoWait(CommandLine, Param1, ...)
ExecutePreProcessor	ExecutePreprocessor
ExitTotal	ExitTotal
Exp	C1=Exp(#CH)
ExpandWhileTrue	C1=ExpandWhileTrue(#SourceChannel, #BoolChannel)
F	C1=F(#CH, F(CrossFrequency))
	C1= F(#CH, F(FilterType WindowType CrossFrequency NumberOfIterations))
	C1=F(#CH, F(FilterType #DepthChannel))
FallingEdge	C1=FallingEdge(#CH)
FillFromBool	C1=FillFromBool(#SourceChannel, #BoolChannel)
FillWithNextBool	C1=FillWithNextBool(#SourceChannel, #BoolChannel)
FindGPSFreq	C1= FindGPSFreq(#GPS_Channel)
FirstValue	C1=FirstValue(#CH, Rate)
Floor	C1= Floor (#CH)
FlushFiles	FlushFiles
FlushQuiet	FlushQuiet
Freq	C1=Freq(#CH, NewRate)
FreqBase	C1=FreqBase(#CH, NewRate, GPSRate)
FreqNI	C1=FreqNI(#CH, NewRate)
GPSShift	C1=GPSShift(#GPSChannel, #ShiftChannel, NewFreq)
Hide	Hide(#CH)
HoldWhileTrue	C1=HoldWhileTrue(#SourceChannel, #BoolChannel)
If	C1=if(#CH, Operator, #CH, TrueResult, FalseResult)
	C1=if(#CH, Operator, Constant , TrueResult, FalseResult)
IfExists	IfExists(#CH)
IfLaptimesExist	IfLaptimesExist
IfOrgExits	IfOrgExists(#CH)
IfLicenceNameContains	IfLicenceNameContains(LicenceName)
IfNotExists	IfNotExists(#CH)
IfNotLaptimesExist	IfNotLaptimesExist

IfNotOrgExits	IfNotOrgExits(#CH)
IfNotLicenceNameContains	IfNotLicenceNameContains(LicenceName)
IfNotSpecValueContains	IfNotSpecValueContains(Group.Entry, SearchStr)
IfNotSpecValueExists	IfNotSpecValueExists(Group.Entry)
IfNotSpecValueIs	IfNotSpecValueIs(Group.Entry, SearchValue)
IfSpecValueContains	IfSpecValueContains(Group.Entry, SearchStr)
IfSpecValueExists	IfSpecValueExists(Group.Entry)
IfSpecValueIs	IfSpecValueIs(Group.Entry, SearchValue)
IIR-Lowpass	C1=F(#CH, F(IIR(CrossFrequency HZ)))
Integer formular	C1=F(#CH, I(Multiplicand, Divisor, Offset))
Integrate	C1=Integrate(#CH) C1=Integrate(#CH, Value)
InterpolateWhileTrue	C1=InterpolateWhileTrue(#SourceChannel, #BoolChannel) C1=InterpolateWhileTrue(#SourceChannel, #BoolChannel, MaxGap)
InterpolateWhileFalse	C1=InterpolateWhileFalse(#SourceChannel, #BoolChannel) C1=InterpolateWhileFalse(#SourceChannel, #BoolChannel, MaxGap)
LastValue	C1=LastValue(#CH, Rate)
Limit	C1=Limit(#CH, minLimit, maxLimit)
Log	C1=Log(#CH)
LongInt	C1=LongInt(#CH) C1=LongInt(#CH, LowerBorder, UpperBorder) C1=LongInt(#CH, Step)
Madgwick_AHRS_IMU	C1=Madgwick_AHRS_IMU(#GYRO_x, #GYRO_y, #GYRO_z, #ACC_x_1, #ACC_y, #ACC_z, Value)
Make32	C1=Make32(#CHHi, #CHLo)
Make32GPS	C1=Make32GPS(#CH)
Make64	C1=MAKE64(#CHHi, #CHLo)
MakeDouble	C1=MakeDouble(#CHHi, #CHLo)
MakeGPS	C1=MakeGPS(#CHHi, #CHLo)
MakeLaptimeCh	C1=MakeLaptimeCH(Rate)
MakeSectimeCh	C1=MakeSectimeCh(Rate)
MakeSingle	C1=MakeSingle(#CHHi, #CHLo)
Max	C1=Max(#CH1, #CH2)
MaxHoldWhileTrue	C1=MaxHoldWhileTrue(#SourceChannel, #BoolChannel)
MaxValue	C1=MaxValue(#CH, Rate)
MaxWhileTrue	C1=MaxWhileTrue(#SourceChannel, #BoolChannel)
Mileage	
Min	C1=Min(#CH1, #CH2)
MinHoldWhileTrue	C1=MinHoldWhileTrue(#SourceChannel, #BoolChannel)
MinValue	C1=MinValue(#CH, Rate)
MinWhileTrue	C1=MinWhileTrue(#SourceChannel, #BoolChannel)
Mod	C1=Mod(#CH1, #CH2) C1=Mod(#CH, constant)
MovWhileTrue	C1=MOVWhileTrue(#SourceChannel, #BoolChannel)
MUX	C1=MUX(#DataChannel, #MuxChannel, MuxVal)
NAnd	C1=NAND(#CH1, #CH2)

	C1=NAND(#CH, Constant)
NewMainFreq	NewMainFreq(NewRate)
NewResult	
NoOp	C1=NoOp(#CH)
Not	C1=Not(#CH)
OnErrorExitGroup	OnErrorExitGroup
OnErrorExitTotal	OnErrorExitTotal
Or	C1=Or(#CH1, #CH2)
	C1=Or(#CH1, Constant)
PeakPreView	C1=PeakPreview(#CH, FallbackThresholdNumber)
PosMaxWhileTrue	C1=PosMaxWhileTrue (#SourceChannel, #BoolChannel)
PosMinWhileTrue	C1=PosMinWhileTrue(#SourceChannel, #BoolChannel)
Power	C1=Power (#CH1, #CH2)
	C1=Power (#CH, Constant)
QuietMode	QuietMode
RcLp	C1=RCLP(#CH, Rate)
Real formular	C1=F(#CH, R(Factor, Offset))
Restore	Restore(#CH)
Result	
RisingEdge	C1= RisingEdge(#CH)
RMSWhileTrue	C1=RMSWhileTrue(#SourceChannel, #BoolChannel)
Roll Angle	C1= RollAngle (#AccY, #AccZ, #GyroY, #GyroZ)
Rot_3D_XYZ	C1=Rot_3D_XYZ(#SourceCH1, #SourceCH2, #SourceCH3, Angle1, Angle2, Angle3)
Rot_3D_XYZ_Var	C1=Rot_3D_XYZ_Var(#SourceCH1, #SourceCH2, #SourceCH3, #RotateCH1, #RotateCH2, #RotateCH3)
Rot_3D_ZYX	C1=Rot_3D_ZYX(#SourceCH1, #SourceCH2, #SourceCH3, Angle1, Angle2, Angle3)
Rot_3D_ZYX_Var	C1=Rot_3D_ZYX_Var(#SourceCH1, #SourceCH2, #SourceCH3, #RotateCH1, #RotateCH2, #RotateCH3)
Round	C1=Round (#CH)
SaveTemporaryChannels	
Set	C1=SET(DIM='Value')
	C1=SET(Sensorinfo='Text')
SetLapTrigger	C1=SetLapTrigger(#CH)
	C1=SetLapTrigger(#CH, Timeout)
SetSecTrigger	C1=SetSecTrigger(#CH)
	C1=SetSecTrigger(#CH, Timeout)
Shift	C1=Shift(#CH, Constant)
	C1=Shift(#CH, Constant sec)
ShowNoErrors	ShowNoErrors
Sig	C1=Sig(#CH)
Signed	C1=Signed(#CH)
Sin	C1=sin(#CH)
Single	C1=Single(#CH)
Slip	C1=Slip (#CH_Front, #CH_Rear)
SOD	C1=SOD(#HHMM, #SSH)
Speed_L4	C1=Speed_L4 (#CH)

Speed2	C1=Speed2 (#CH1,#CH2)
SQRT	C1=SQRT(#CH)
Sum	C1=Sum(#CH)
	C1=Sum(#CH , <i>Value</i>)
Swap	C1= Swap(#CH)
T	C1=F(#CH, T(TableName.Ext))
T2D	C1=F(#CH1, #CH2, T(TableName.Ext, IntType))
T2DSearch	C1=T2DSearch(#CH1, #CH2, T(TableName.Ext, IntType), [SearchDirection])
T3D	C1= F(#CH1, #CH2, #CH3, T(TableName.Ext, IntType))
Tan	C1=tan(#CH)
TimeForTrue	C1=TimeforTrue(#CH)
TimeSinceTrue	C1=TimeSinceTrue(#CH1)
Trunc	C1=Trunc(#CH)
UnHide	Unhide(#CH)
UnSigned	C1= UnSigned(#CH)
UseBits	C1=UseBits(#DataChannel, MaskValue)
VCount	C1= VCount(#CH)
VDVWhileTrue	C1=VDVWhileTrue(#SourceChannel, #BoolChannel)
Word	C1=Word(#CH)
	C1=Word(#CH, LowerBorder, UpperBorder)
	C1=Word(#CH, Step)
Xor	C1=Xor(#CH1, #CH2)
	C1=Xor(#CH, Constant)
ZeroCross	C1=ZeroCross(#CH)

8 Predefined CAL-files

All predefined calculation files are stored in the *2DCalFiles* directory in User directory (<UserDataDir>\Calfiles\2DCalfiles\ or <UserCal>).

If the calculation files are stored as CAL file it can be edited from every user.

8.1 2D_DistanceAndTimeCH.CAL

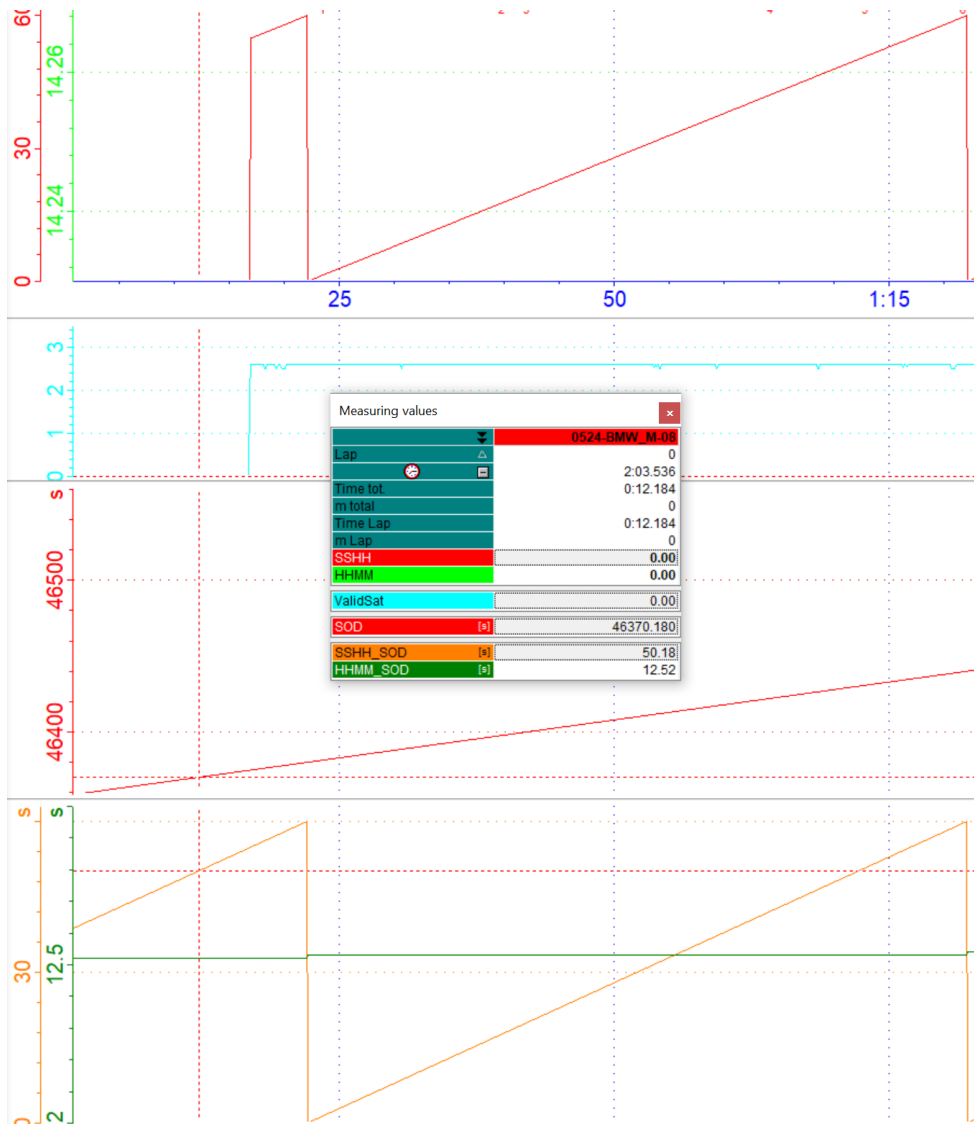
<u>Description:</u>	Creating time and distance channels with high resolution for evaluation purposes	
<u>Available since:</u>	Race2021	
<u>Location:</u>	User data CAL directory	
<u>Include calls:</u>	{\$I <UserDataDir>\Calfiles\2DCalfiles\2D_DistanceAndTimeCH, P()} {\$I <UserCal>\2DCalFiles\2D_DistanceAndTimeCH, P()}	
<u>Input parameters:</u>	@Int_Speed FilterSettings.ResampleFreq	
<u>Created channels:</u>	#Speed_Lo:	Channel generated from @Int_Speed by filtering and up sampling, which is used to calculate the Distance2D channel.
	#Distance2D:	Channel which outputs the distance already covered since the start of the measurement at the current point in time.
	#Time2D:	Channel which displays the time elapsed since the start of the measurement at the current point in time.
<u>Notes:</u>	If no frequency is deposited in SpecSheet-Group FilterSettings in entry ResampleFreq, the speed channel, which is selected at special channel @Int_Speed is filtered with an AVG-Filter and then up sampled to 1000 Hz. 2D_DistanceAndTimeCH.CAL is executed automatically with 2D_LapChannels.CAL	

8.2 2D_LapChannels.CAL

<u>Description:</u>	Creating Lap-related channels for evaluation purposes.	
<u>Available since:</u>	Race2022	
<u>Location:</u>	User data CAL directory	
<u>Include calls:</u>	{\$I <UserDataDir>\Calfiles\2DCalfiles\2D_LapChannels, P()} {\$I <UserCal>\2DCalFiles\2D_LapChannels, P()}	
<u>Input parameters:</u>	#Time2D #Speed_Lo	
<u>Created channels:</u>	#InLap:	Boolean channel which only is TRUE ("1") in lap (for WHILETRUE-calculations).
	#TimeInLap:	Up counting time in seconds in current lap.
	#TimeForLap:	Absolute time in seconds in current lap.
	#Lapmeter2D:	Up counting lapmeters in meters in current lap.
	#Lapmeter2D_Rel:	Up counting lapmeters in percent in current lap.
	#LapNr:	Lap counter
	#FastesLap:	Marking fastest lap with "1"
<u>Notes:</u>	2D_DistanceAndTimeCH.CAL is executed automatically with 2D_LapChannels.CAL	

8.3 2D_SOD Reverse.CAL

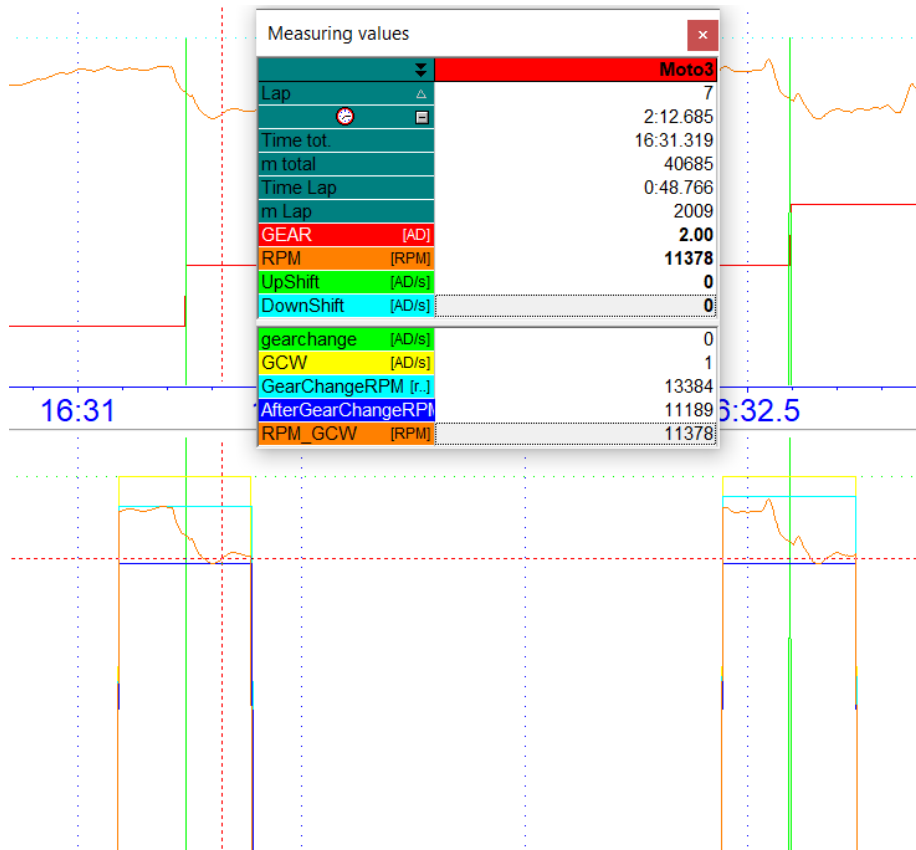
Description:	Recalculating GPS time channels (#HHMM and #SSHH) from channel #SOD (SecondOfDay).
Available since:	Race2022
Location:	User data CAL directory
Input parameters:	#SOD
Include calls:	{I <UserDataDir>\Calfiles\2DCalfiles\2D_SOD_Reverse, P()} {I <UserCal>\2DCalFiles\2D_SOD_Reverse, P()}
Created channels:	#HHMM_SOD: Recalculated #HHMM channel (Hours & Minutes) #SSHH_SOD: Recalculated #SSHH channel (Seconds & Hundredth)
Notes:	Recalculating the GPS time channels from #SOD is very important because #HHMM and #SSHH may not be correct with initial poor GNSS reception. With channel #SOD, the poor GNSS reception is compensated for as best as possible by extrapolation. By activating the lines in group [WriteToSpecSheet] the first and last values of created channels can be written to the SpecSheet.



- For more information about channel #SOD please see the manual “GPS postprocessing” on 2D website: <http://2d-datarecording.com/downloads/manuals/>

8.4 GearCount2D.CAL

Description:	Generation of channels for the evaluation of gear changes.	
Available since:	Race2020	
Location:	RaceData\DemoData\Moto3\	
Input parameters:	@MilageSpeed @Int_Speed @GearNr @Drive_RPM #Time2D #Distance2D	
Created channels:	Gear	Currently engaged gear
	RPM	Current engine rpm
	GCW	GearChangeWindow (Time defined by AVG-Filter)
	EOGCW	End of GearChangeWindow (Boolean marking of falling edge of GCW)
	SOGCW	Start of GearChangeWindow (Boolean marking of rising edge of GCW)
	RPM_GCW	Current engine rpm → outside GCW = 0
	GearChangeRPM	Maximum RPM inside GCW
	AfterGearChangeRPM	Minimum RPM inside GCW
	DeltaGearChangeRPM	Difference between maximum and minimum RPM inside GCW
Notes:	The size of the “window” around the gear change can be chosen in CAL file.	



```

6 [Mileage]
7 Mileage(@Mileage_Speed, Download.LoggerName) ; Calculates the by the Logger travelled distance for e.g. maintenance purposes -> SpecSheet Download.Mileage
8
9 [Time2D]
10 IfNotExists(#Time2D)
11
12 C1=Const(1)
13 Time2D=I(#C1) ; Create Time2D If Not already exists
14
15 [Distance2D]
16 IfNotExists(#Distance2D)
17 C1=/(@Int_Speed, 3.6) ; Converting speed from [km/h] in [m/s]
18 C1=Set(Dim="m/s") ; Set dimension to [m/s]
19 Distance2D=I(#C1) ; Integrate #C1
20 LapMeter=I(#C1,0) ; Integrate #C1 -> Set #Lapmeter to 0 at passing a Laptrigger
21
22 [If GearChannel Exists]
23 IfExists(@Gear_Nr) ; Query If Gear-signal already exists -> Special-channel (@) is queried! (Settings -> Special channels)
24 C1 = Shift(@Gear_Nr, 0.1 sec) ; Shift @Gear_Nr by 0.1 sec to the right -> Correcting ECU-Errors: Some ECUs send gears > 6 during shifting -> gear replacement = last gear
25 ; C1 = Shift(@Gear_Nr,-0.1) ; Shift @Gear_Nr by 0.1 sec to the left -> Correcting ECU-Errors: Some ECUs send gears > 6 during shifting -> gear replacement = next gear
26 C2 = If(@Gear_Nr,>,6,#C1,@Gear_Nr) ; If ECU creates some "errornumber" inbetween gear change, last gear is used -> C1 = Shift(@Gear_Nr, 0.1 sec)
27 gearchange= F'(#C2) ; Derivate #C2, to detect gear change
28 gearchange= Set(Sensorinfo='position of gear change by @Gear_Nr') ; Set Sensorinfo
29
30 [If Not GearChannel Exists]
31 IfNotExists(@Gear_Nr) ; Query If Gear-signal Not exists -> Special-channel (@) is queried! (Settings -> Special channels)
32 C1 = F(@Drive_RPM,F(IIR(3 Hz))) ; IIR-Filter with cut-off frequency of 3 Hz to smooth the signal
33 C2 = F'(#C1) ; Derivate #C1 to detect gear changes
34 gearchange= If(#C2, <, -4500, 1, 0) ; Boolean channel to detect gearchange -> only shifts are detect, but Not If up/down -> must be adapted to ECU!
35 gearchange= Set(Sensorinfo='position of gear change by RPM-Derivative') ; Set Sensorinfo
36
37 [UpDownShifts]
38 UpShift = If(#gearchange,>,0,1,0) ; Detect upshift -> #gearchange positive
39 DownShift = If(#gearchange,<,0,1,0) ; Detect downshift -> #gearchange negative
40
41 [Counters]
42 UpShift_cnt = Sum(#UpShift) ; Sum up total upshifts -> Alternative: UpShift_cnt=CountChanges(@Gear, 1) -> Counts all positive steps of #Gear with step-size >=1
43 UpShift_cnt = Set(Sensorinfo='Number of Upshifts') ; Set Sensorinfo
44 UpShift_cnt_Lap = Sum(#UpShift,0) ; Sum up upshifts per Round -> Sum(#UpShift, 0) -> #UpShift_cnt_Lap is Set to 0 at every Laptrigger
45 UpShift_cnt_Lap = Set(Sensorinfo='Number of Upshifts per Lap') ; Set Sensorinfo
46 DownShift_cnt = Sum(#DownShift) ; Sum up total downshifts
47 DownShift_cnt = Set(Sensorinfo='Number of Downshifts') ; Set Sensorinfo
48 DownShift_cnt_Lap = Sum(#DownShift,0) ; Sum up downshifts per Round -> Sum(#UpShift, 0) -> #UpShift_cnt_Lap is Set to 0 at every Laptrigger
49 DownShift_cnt_Lap = Set(Sensorinfo='Number of Downshifts per Lap') ; Set Sensorinfo
50
51 [GearChangeRPM]
52 C1 = FreqNI(@GearChange,@Drive_RPM.rate) ; Frequency no Interpolation -> Frequency interpolation is only applied to integer values
53 C1 = F(#C1,F(avg(0.3 sec))) ; Average filtering -> A trick to create a 0.3 sec time window -> 0.15 sec before And 0.15 after
54 C1 = Shift(#C1,-0.0 sec) ; Possibility to readjust window position -> negative: Shifting left! -> positive: Shifting right
55 GCW = I(#C1,>,0,1,0) ; GearChangeWindow status channel -> Set peak of time window to 1 (boolean)
56 EOGCW = FallingEdge(#GCW) ; Detecting End Of GearChangeWindow
57 EOGCW = Shift(#EOGCW,-1) ; Shifting End of GearChangeWindow by 1 sample to left
58 SOGCW = RisingEdge(#GCW) ; Detecting Start Of GearChangeWindow (Not used -> Just for demonstration purposes)
59 C1 = MaxHoldWhileTrue(@Drive_RPM,#GCW) ; Detect maximum RPM inside GearChangeWindow
60 C1 = FillWithNextBool(#C1,#EOGCW) ; Set maximum RPM value of current GearChangeWindow until falling edge of current GearChangeWindow
61 Result = ExpandWhileTrue(#C1, #GCW) ; Holds first value of first sample of true-area (rising edge) until last sample of true-area (falling edge)
62 Result = Set(Dim='rpm') ; Set dimension to [rpm]
63 Result = Set(Sensorinfo='Max RPM during GearChange') ; Set Sensorinfo
64 C1 = MinHoldWhileTrue(@Drive_RPM,#GCW) ; Detect minimum RPM inside GearChangeWindow
65 C1 = FillWithNextBool(#C1,#EOGCW) ; Set minimum RPM value of current GearChangeWindow until falling edge of current GearChangeWindow
66 AfterGearChangeRPM = ExpandWhileTrue(#C1, #GCW) ; Holds first value of first sample of true-area (rising edge) until last sample of true-area (falling edge)
67 AfterGearChangeRPM = Set(Dim='rpm') ; Set dimension to [rpm]
68 AfterGearChangeRPM = Set(Sensorinfo='Min RPM during GearChange') ; Set Sensorinfo
69 C1 = -(@GearChangeRPM,#AfterGearChangeRPM) ; Calculate RPM-Delta for GearChange
70 DeltaGearChangeRPM = FillWithNextBool(#C1,#EOGCW) ; Set RPM-Delta of current GearChangeWindow until falling edge of current GearChangeWindow
71 DeltaGearChangeRPM = Set(Dim='rpm') ; Set dimension to [rpm]
72 DeltaGearChangeRPM = Set(Sensorinfo='Delta RPM during this GearChange') ; Set Sensorinfo
73
74 [UpShiftRPM_2D]
75 C1 = FallingEdge(#GCW) ; Detect falling edge of GearChangeWindow
76 Result = If(#C1,>,0,#GearChangeRPM,0) ; If #C1 is bigger than 0 Result is #GearChangeRPM, else Result is 0
77 UpShiftRPM_SUM_2D = Sum(#UPShiftRPM_2D) ; Sum up each sample of #UPShiftRPM_2D
78 UpShiftRPM_LAP_2D = Sum(#UPShiftRPM_2D,0) ; Sum up each sample of #UPShiftRPM_2D -> #UPShiftRPM_2D is Set to 0 at every Laptrigger
79 UpShiftRPM_2D_AVG = /(UpShiftRPM_Lap_2D,#UpShift_cnt_Lap) ; Calculating Average
80
81 [HoldWhileTrue]
82 RPM_GCW=HoldWhileTrue(#RPM, #GCW) ; #RPM_GCW takes the values of #RPM as long as #GCW is true. Otherwise #RPM_GCW is 0.

```

This document is subject to change at 2D decision. 2D assumes no responsibility for any claims or damages arising out of the use of this document, or from the use of modules based on this document, including but not limited to claims or damages based on infringement of patents, copyrights or other intellectual property rights.



- CAL-file and Template can be found in Moto3-DemoData with DemoUser

In addition to the calculation of the mileage function, the first groups are queried whether a channel is available that has recorded the engaged gear. If no gear channel is available, a gear channel is created via the derivative of the RPM.

This is followed by the Boolean marking of the up- and downshifts and various counters.

From group [GearChange_RPM] on, the actual analysis of the up-switching operations takes place, whereby, at first, with the help of an AVG filter, a 0.3 second GearChangeWindow (GCW) is created around the gear change, in which the rpm during the gear change will be analysed:

```
C1 = FreqNI(#GearChange,@Drive_RPM.rate)
C1 = F(#C1,F(avg(0.3 sec)))
C1 = Shift(#C1,-0.0 sec)
GCW = If(#C1,>,0,1,0)
```

8.1 2D Conditions

<u>Description:</u>	Combining two trigger conditions to a Boolean TRUE condition
<u>Available since:</u>	Race2022
<u>Location:</u>	User data CAL directory
<u>Include calls:</u>	{\${ <UserDataDir>\Calfiles\2DCalfiles_2D_Conditions, P(In1, In2, OutName)} {\${ <UserCal>\2DCalFiles\2D_Conditions, P(ICH1, CH2, OutName)}
<u>Input parameters:</u>	CH1: In-Trigger condition 1 CH2: In-Trigger condition 2 OutName: Defining name of output channel
<u>Created channels:</u>	'OutName': Boolean TRUE channel (value "1")
<u>Notes:</u>	Boolean TRUE condition only will carry TRUE when a CH1-Trigger is followed directly by a CH2-Trigger. CH1 and CH2 must be 0/1/0 signals (e.g., created by RisingEdge(#CH) or FallingEdge(#CH)). Created channel 'OutName' can be used for WhileTrue-CalcTool commands or as Phases in various 2D Analyzer tools.

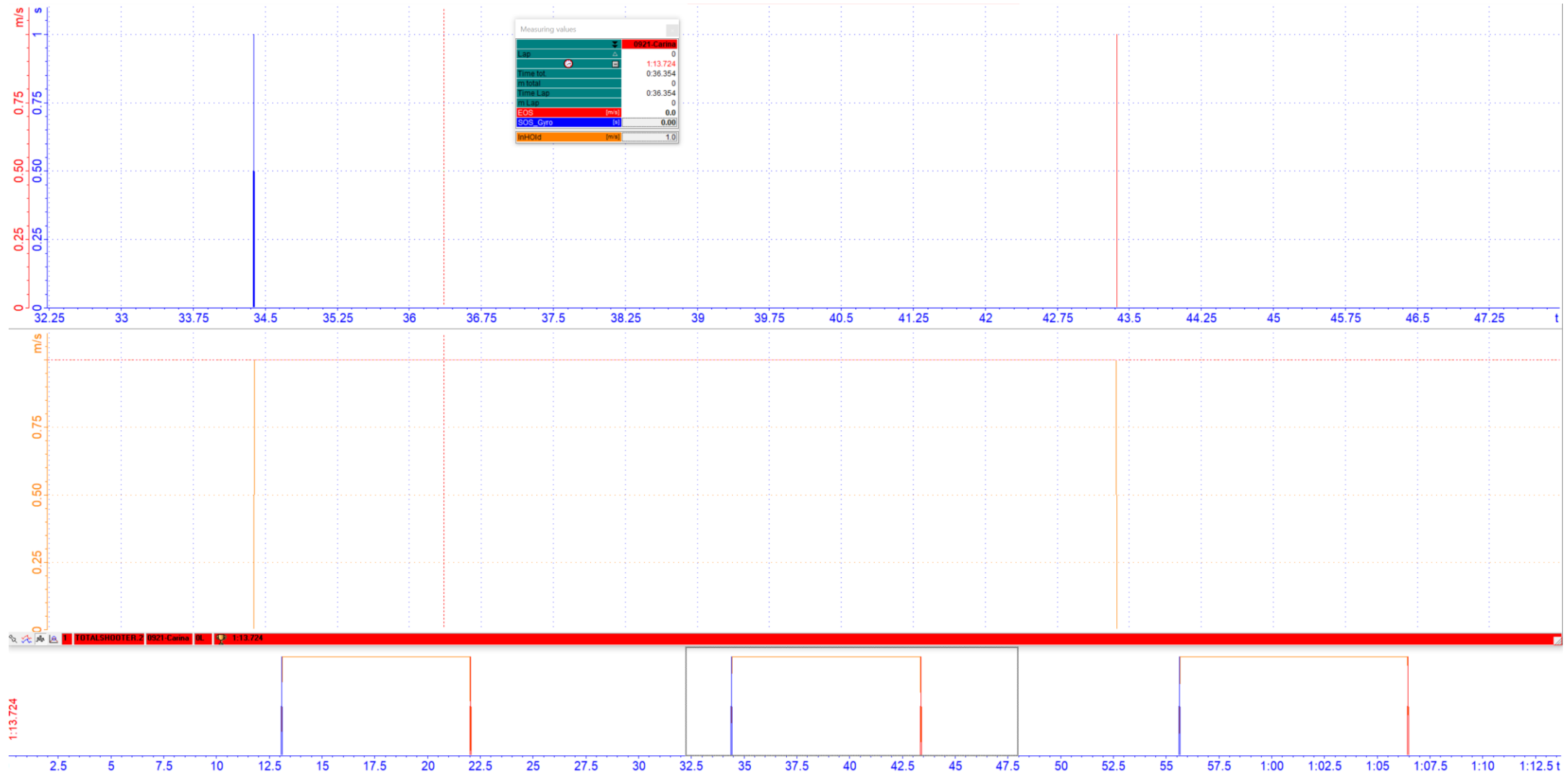


- For more information about *Phases* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>



- For more information about *WhileTrue commands* please see the chapter Signal analysis in this manual.

Include call: {\$I <UserCal>\2DCalFiles_2D_Conditions, P(SOS_Gyro, EOS, InHold)}



This document is subject to change at 2D decision. 2D assumes no responsibility for any claims or damages arising out of the use of this document, or from the use of modules based on this document, including but not limited to claims or damages based on infringement of patents, copyrights or other intellectual property rights.

9 Toolchains

Toolchains always consist of one or more calculation files and are used in postprocessing to realise various functions.

Many toolchains can be combined in a meaningful way to create a complex evaluation functionality. Toolchains can be executed automatically one after the other via Scripts.



- For more information about channel *Scripts* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.1 2D GPSAuto

<u>Description:</u>	Further processing of recorded GPS/GNSS channels (Filtering, Time Shift, Creation of further GPS/GNSS channels)
<u>Available since:</u>	Race2012
<u>Location:</u>	Race application CAL directory
<u>Notes:</u>	Toolchain is executed automatically at download via AutoCalcConfigurator



- For more information about channel *GPS postprocessing* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.2 2D GPSTracks

<u>Description:</u>	User definable GPS Triggers can be combined to create measurement track channels between or inside trigger point, which are very convenient for further data analysis in the CalcTool as BOOL Channels for further calculations or as phases for Exports, Min/Max Tables, plots, etc
<u>Available since:</u>	Race2021
<u>Location:</u>	User Data CAL directory
<u>Notes:</u>	



- For more information about channel *GPSTracks* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.3 2D ValidateTracks

<u>Description:</u>	With 2D_ValidateTracks additional, for the evaluation of the previously created GPSTracks channels, are created (DistanceInTrack, TimeInTrack, IsTrack, ...). For subsequent, developmental calculations (MinEntrySpeed, AVGSpeed, MinLength, ...) of the GPSTrack channels, different CalcTool commands can be used in combination with the ValidateTrack channels (#IsTrigger).
<u>Available since:</u>	Race2021
<u>Location:</u>	User Data CAL directory
<u>Notes:</u>	



- For more information about channel *ValidateTracks* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.4 2D_FilterAndRotate

<u>Description:</u>	The purpose of the <i>2D_FilterAndRotate</i> toolchain is to prepare the already recorded accelerometer and gyroscope channels for the later uses in the 2D Datarecording toolchains in post-processing. The preparation of accelerometer and gyroscope channels contains rotational correction and filtering.
<u>Available since:</u>	Race2022
<u>Location:</u>	User Data CAL directory
<u>Notes:</u>	



- For more information about channel *FilterAndRotate* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.5 2D_Comfort

<u>Description:</u>	Further processing of Accelerometer data to create conclusions for Comfort-based evaluation via RMS/VDV/MOV or other customisable calculations.
<u>Available since:</u>	Race2022
<u>Location:</u>	User Data CAL directory
<u>Notes:</u>	



- For more information about channel *Comfort* please see the respective manual on 2D website:
<http://2d-datarecording.com/downloads/manuals/>

9.6 Combining toolchains

Individual toolchains can be combined to produce calculation results and display the results in plots of the 2D Analyzer and export the results to Excel sheets.

A good example for the combination of toolchains is the evaluation of comfort driving tests of cars.

The channels received via GPS are first prepared via the *2D_GPS Auto* toolchain for processing the GPS channels and further channels are generated from original GPS channels.

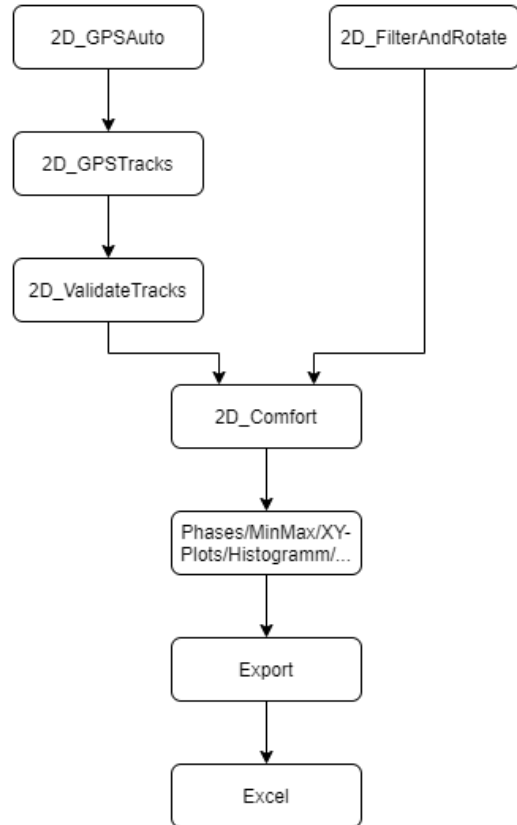
Different driving sections on the test field (rumble strips, curves, braking zones, acceleration zones, ...) can be defined using the *2D_GPSTracks* toolchain to create GPSTracks channels in 2D Analyzer.

With *2D_ValidateTracks* additional, for the evaluation of the previously created GPSTracks channels, are created (DistanceInTrack, TimeInTrack, IsTrack, ...). For subsequent, developmental calculations (MinEntrySpeed, AVGSpeed, MinLength, ...) of the GPSTrack channels, different CalcTool commands can be used in combination with the ValidateTrack channels.

For the driving comfort evaluation, the respective channels of the IMUs used are first rotated into an identical coordinate system in post-processing so that the channels of the different IMUs can be compared. The IMU signals can be also processed via filtering. Rotation and filtering are performed using the *2D_FilterAndRotate* toolchain.

Then the combination of GPSTrack channels provided via the *2D_ValidateTracks* toolchain and the *2D_Comfort* toolchain can be used to calculate the comfort parameters RMS, VDV and MOV.

Via exports, the plots and results of the calculations can be visualised in 2D Analyzer plots and exported in excel sheets, which can then be saved and printed out.



- 2D offers support, implementation, and evaluation as a service. If you are interested, please contact us via the contact form on the website.