Document Title	Specification of Floating Point Math Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	397

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R22-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2022-11-24	R22-11	AUTOSAR Release Management	• Updated SWS_Mfl_00285 and SWS_Mfl_00289. • New function added SWS_Mfl_00820 and updated the following: SWS_Mfl_00142, SWS_Mfl_00152, SWS_Mfl_00157, SWS_Mfl_00163, SWS_Mfl_91000, SWS_Mfl_91001, SWS_Mfl_00854, SWS_Mfl_91003 and SWS_Mfl_91004.
2021-11-25	R21-11	AUTOSAR Release Management	• New requirements SWS_Mfl_00843 and SWS_Mfl_00848 added for "Mfl_FloatToIntCvrt_f32" function. • Editorial changes.
2020-11-30	R20-11	AUTOSAR Release Management	• New function added Mfl_DivLim_f32 with the requirements SWS_Mfl_00844, SWS_Mfl_00845, SWS_Mfl_00846 and SWS_Mfl_00847. • Updated Chapter 7.1 Error classification.
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published

2019-07-29	4.4.1	AUTOSAR Release Management	• New function description for Mfl_Mod_f32 function • Added new 64 bit variants in conversion function - SWS_Mfl_00836, SWS_Mfl_00839, SWS_Mfl_00838, SWS_Mfl_00842
2018-10-31	4.4.0	AUTOSAR Release Management	• Added description for Mfl_Pow_f32 function • Updated name of parameter dT_f32 in the requirements SWS_Mfl_00045, SWS_Mfl_00047, SWS_Mfl_00301 & SWS_Mfl_00303
2017-12-08	4.3.1	AUTOSAR Release Management	• A note has been added in Section 8.1 of MFL specification to provide clarity in usage of mnemonic for Boolean data types. • Inclusion of Pointer to Constant (P2CONST) for SWS_Mfl_00260, SWS_Mfl_00246, SWS_Mfl_00225 & SWS_Mfl_00223 and proper categorization of Parameters as Out/InOut for SWS_Mfl_00266, SWS_Mfl_00285 & SWS_Mfl_00037.

2016-11-30	4.3.0	AUTOSAR Release Management	• Section 2 has been revisited to update Default Error Tracer instead of Development Error tracer. • SWS_Mfl_00362 has been updated to provide clarity in requirements. • SWS_Mfl_00363 has been modified to provide clear requirements. • Updated the parameters in SWS_Mfl_00360 for Mfl_ArcTan2_f32 service to be in sync with standard C library. • Updated SWS_Mfl_00122 to provide better clarity on the input parameter limits. • Verified that the spec SWS_Mfl_00122 has been updated to provide better clarity on input parameter limits. • Updated MFL document to support MISRA 2012 standard. (Removed Reference related to MISRA 2004 from chapter 3.2 ans redundant statements in SWS_Mfl_00809 which already exist in SWS_BSW document and SWS_SRS document) • Modified the reference to SRS_BSW_General (SRS_BSW_00437) & (SRS_BSW_00448) for SWS_Mfl_00810 & SWS_Mfl_00822 requirements.

2015-07-31	4.2.2	AUTOSAR Release Management	• BSWUML Model for "Mfl_HystCenterHalfDelta_f32_u8", "Mfl_HystLeftRight_f32_u8", "Mfl_HystDeltaRight_f32_u8" & "Mfl_HystLeftDelta_f32_u8" functions were updated in the Word Document. • Statement has been updated for Mfl_DT1Typ1Calc and Mfl_DT1Typ2Calc to clearly mention the data type for the Time Equivalent parameter. • Description field has been updated/rectified for Tv_C and Tnrec_C parameters in Mfl_ParamPID_Type. • Updated naming convention for TeQ_f32 Parameter. • Corrected the description for TeQ_<Size> in section 8.5.4.1 and statement in section 8.5.4.4. • Naming convention followed for Tnrec Parameter in Mfl_PISetParam function. • Statement has been updated to correct naming convention for TeQ_f32. • Updated SWS_Mfl_00001 for naming convention under Section 5.1, File Structure • BSWUML Model for "Mfl_ArrayAverage_f32_f32" function was updated to include pointer to constant to avoid MISRA violation/warning. (SWS_Mfl_00192) • Valid range for float32 has been updated in Section 8.2 and removed float64 data type from Section 8.1, 8.2 and Section 2 • Removed the requirements SWS_Mfl_00240, SWS_Mfl_00245, SWS_Mfl_00250 & SWS_Mfl_00255 • Removed redundant requirements SWS_Mfl_00034, SWS_Mfl_00046 & SWS_Mfl_00302, which were covered as part of section 8.5.4.4.

2014-10-31	4.2.1	AUTOSAR Release Management	• New Functions are added to convert values between Float and Integer. (SWS_Mfl_00837, SWS_Mfl_838, SWS_Mfl_840, SWS_Mfl_841 & SWS_Mfl_842) • BSWUML Model was updated for "Mfl_FloatToIntCvrt_f32" & "Mfl_IntToFloatCvrt" functions. (SWS_Mfl_00836 & SWS_Mfl_839) • Updated usage of const in a consistent manner.
2014-03-31	4.1.3	AUTOSAR Release Management	• Removed SWS_Mfl_00206, SWS_Mfl_00207 and SWS_Mfl_00281 from Mfl_RampCalc & Mfl_RampCalcJump functions.
2013-10-31	4.1.2	AUTOSAR Release Management	• Deprecated: Mfl_DeadTime function • Removed: SWS_Mfl_00197 from Mfl_Hypot function • Added: SWS_Mfl_00835 for Mfl_RampCalc function, a note for Mfl_RampGetSwitchPos function • Modified: Description for Mfl_RampSetParam function, Parameter (in) definition for Mfl_RateLimiter_f32 • Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Description and requirements are modified for Mfl_RampCalcJump, Mfl_RampCalc • Formatting error in superscipts are corrected • Corrected "DT1" to "I" in I-Controller functions • Description of the parameter "State" is corrected in Mfl_Debounce and Mfl_DebounceInit functions • Corrected for 'DependencyOnArtifact'
2011-12-22	4.0.3	AUTOSAR Administration	• Removal of 'Accumulator routine' • Revised 'Trigonometric routines' names • Added 'Median Sort Routines'

2010-09-30	3.1.5	AUTOSAR Administration	• Introduction of additional LIMITED Functions for controllers • Ramp functions optimised for effective usage • Separation of DT1 Type 1 and Type 2 Controller functions • Introduction of additional approximative function for calculatio of TeQ
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

Contents

1 Introduction and functional overview 11

2 Acronyms and Abbreviations 13

3 Related documentation 14

3.1 Input documents & related standards and norms 14
3.2 Related specification 14

4 Constraints and assumptions 15

4.1 Limitations 15
4.2 Applicability to car domains 15

5 Dependencies to other modules 16

5.1 File structure 16

6 Requirements Tracing 17

7 Functional specification 19

7.1 Error Classification 19
7.1.1 Development Errors 19
7.1.2 Runtime Errors 19
7.1.3 Transient Faults 19
7.1.4 Production Errors 19
7.1.5 Extended Production Errors 19
7.2 Error detection 19
7.3 Error notification 20
7.4 Initialization and shutdown 20
7.5 Using Library API 20
7.6 Library implementation 21

8 Routine specification 23

8.1 Imported types 23
8.2 Type definitions 23
8.3 Comment about rounding 24
8.4 Comment about routines optimized for target 24
8.5 Routine definitions 25
8.5.1 Floating point to Fixed-Point Conversion 25
8.5.2 Fixed-Point to Floating-Point Conversion 26
8.5.3 Rounding 27
8.5.4 Controller routines 28
8.5.4.1 Structure definitions for controller routines 29
8.5.4.2 Proportional Controller 34
8.5.4.3 Proportional controller with first order time constant . 35
8.5.4.4 Differential component with time delay : DT1 38
8.5.4.5 Proportional & Differential controller 44

8.5.4.6 Integral component . . . 47
8.5.4.7 Proportional & Integral controller . . . 51
8.5.4.8 Proportional, Integral & Differential controller . . . 58
8.5.5 Magnitude and Sign . . . 66
8.5.6 Limiting . . . 67
8.5.7 Logarithms and Exponentials . . . 70
8.5.8 Trigonometry . . . 73
8.5.9 Average . . . 81
8.5.10 Array Average . . . 81
8.5.11 Hypotenuse . . . 82
8.5.12 Ramp routines . . . 82
8.5.12.1 Ramp routine . . . 84
8.5.12.2 Ramp Initialisation . . . 85
8.5.12.3 Ramp Set Slope . . . 86
8.5.12.4 Ramp Out routine . . . 87
8.5.12.5 Ramp Jump routine . . . 88
8.5.12.6 Ramp switch routine . . . 89
8.5.12.7 Get Ramp Switch position . . . 92
8.5.12.8 Check Ramp Activity . . . 92
8.5.13 Hysteresis routines . . . 93
8.5.13.1 Hysteresis center half delta . . . 93
8.5.13.2 Hysteresis left right . . . 94
8.5.13.3 Hysteresis delta right . . . 95
8.5.13.4 Hysteresis left delta . . . 96
8.5.14 Mfl_DeadTime . . . 97
8.5.15 Debounce routines . . . 98
8.5.15.1 Mfl_Debounce . . . 98
8.5.15.2 Mfl_DebounceInit . . . 100
8.5.15.3 Mfl_DebounceSetParam . . . 101
8.5.16 Ascending Sort Routine . . . 102
8.5.17 Descending Sort Routine . . . 103
8.5.18 Median sort routine . . . 103
8.5.19 Modulus . . . 107
8.5.20 Division with limitation . . . 108
8.5.21 Bit pattern . . . 109
8.5.22 Fast inverse square root . . . 109
8.5.23 Trigonometric routines . . . 110
8.6 Examples of use of functions . . . 111
8.7 Version API . . . 111
8.7.1 Mfl_GetVersionInfo . . . 111
8.8 Callback notifications . . . 112
8.9 Scheduled functions . . . 112
8.10 Expected interfaces . . . 112
8.10.1 Mandatory interfaces . . . 112
8.10.2 Optional interfaces . . . 112
8.10.3 Configurable interfaces . . . 112

9 Sequence diagrams 113

10 Configuration specification 114

10.1 Published Information 114
10.2 Configuration option 114

A Not applicable requirements 115

1 Introduction and functional overview

AUTOSAR Library routines are the part of system services in AUTOSAR architecture & below figure shows position of AUTOSAR library in layered architecture.

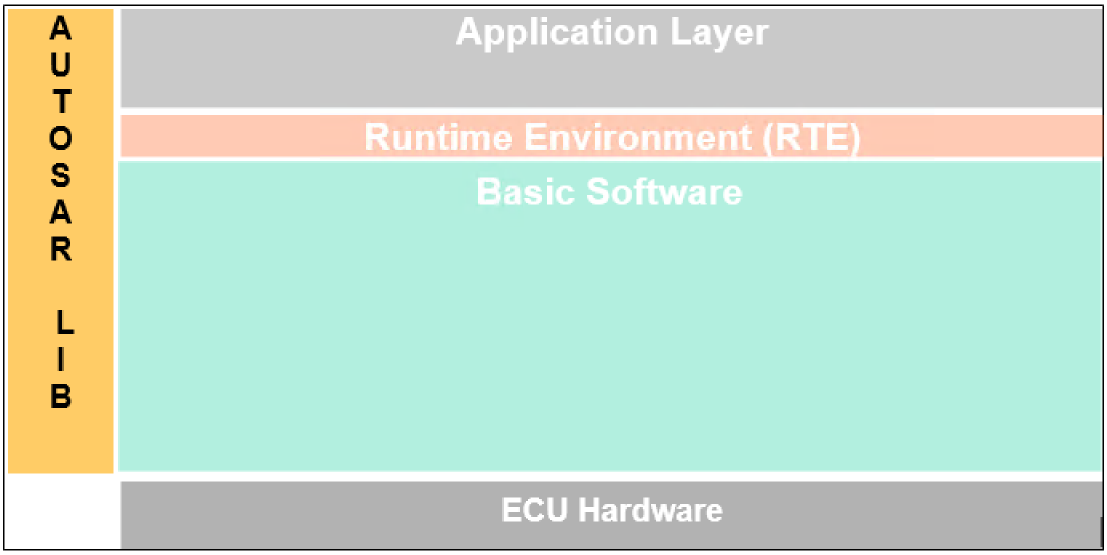

Figure 1.1: Layered Architecture

This specification specifies the functionality, API and the configuration of the AUTOSAR library dedicated to arithmetic routines for floating point values.

The float math library contains routines addressing the following topics:

- Conversion
- Rounding
- Magnitude and sign
- Limiting
- Logarithms and exponential
- Trigonometric
- Controller routines
- Average
- Array Average
- Hypotenuse
- Ramp routines
- Hysteresis function
- Dead Time

- Debounce
- Ascending Sort Routine
- Descending Sort Routine

All routines are re-entrant. They may be used by multiple runnables at the same time.

2 Acronyms and Abbreviations

Acronyms and abbreviations, which have a local scope and therefore are not contained in the AUTOSAR glossary, must appear in a local glossary.

Abbreviation / Acronym:	Description:
abs	Absolute value
Lib	Library
DET	Default Error Tracer
f32	Mnemonic for the float32, specified in AUTOSAR_SWS_PlatformTypes
Limit	Limitation routine
max	Maximum
MFL	Mathematical Floating point Library
min	Minimum
Mn	Mnemonic
s16	Mnemonic for the sint16, specified in AUTOSAR_SWS_PlatformTypes
s32	Mnemonic for the sint32, specified in AUTOSAR_SWS_PlatformTypes
s8	Mnemonic for the sint8, specified in AUTOSAR_SWS_PlatformTypes
u16	Mnemonic for the uint16, specified in AUTOSAR_SWS_PlatformTypes
u32	Mnemonic for the uint32, specified in AUTOSAR_SWS_PlatformTypes
u8	Mnemonic for the uint8, specified in AUTOSAR_SWS_PlatformTypes
boolean	Boolean data type, specified in AUTOSAR_SWS_PlatformTypes

3 Related documentation

3.1 Input documents & related standards and norms

[1] ISO/IEC 9899:1990 Programming Language - C
http://www.iso.org

[2] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, SWS BSW General], which is also valid for MFLLibrary.

Thus, the specification SWS BSW General shall be considered as additional and required specification for MFLLibrary.

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

[SWS_Mfl_00001] ⌈The Mfl module shall provide the following files:

- C files, Mfl_<name>.c used to implement the library. All C files shall be prefixed with 'Mfl_'.

⌋*(SRS_LIBS_00005)*

Implementation & grouping of routines with respect to C files is recommended as per below options and there is no restriction to follow the same.

Option 1 : <Name> can be function name providing one C file per function,

eg.: Mfl_Pt1_f32.c etc.

Option 2 : <Name> can have common name of group of functions:

- 2.1 Group by object family:

  eg.:Mfl_Pt1.c, Mfl_Dt1.c, Mfl_Pid.c

- 2.2 Group by routine family:

  eg.: Mfl_Conversion.c, Mfl_Controller.c, Mfl_Limit.c etc.

- 2.3 Group by method family:

  eg.: Mfl_Sin.c, Mfl_Exp.c, Mfl_Arcsin.c, etc.

- 2.4 Group by other methods: (individual grouping allowed)

Option 3 : <Name> can be removed so that single C file shall contain all Mfl functions, eg.: Mfl.c.

Using above options gives certain flexibility of choosing suitable granularity with reduced number of C files. Linking only on-demand is also possible in case of some options.

6 Requirements Tracing

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_Mfl_00815]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Mfl_00809]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Mfl_00812]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Mfl_00813]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_Mfl_00815]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Mfl_00815]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Mfl_00811]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Mfl_00814]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Mfl_00812]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Mfl_00814]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_Mfl_00814]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Mfl_00815] [SWS_Mfl_00816]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Mfl_00816]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_Mfl_00810]
[SRS_BSW_00448]	Module SWS shall not contain requirements from other modules	[SWS_Mfl_00822]
[SRS_LIBS_00001]	The functional behavior of each library functions shall not be configurable	[SWS_Mfl_00818]
[SRS_LIBS_00002]	A library shall be operational before all BSW modules and application SW-Cs	[SWS_Mfl_00800]
[SRS_LIBS_00003]	A library shall be operational until the shutdown	[SWS_Mfl_00801]

▽

△

Requirement	Description	Satisfied by
[SRS_LIBS_00005]	Each library shall provide one header file with its public interface	[SWS_Mfl_00001] [SWS_Mfl_00854] [SWS_Mfl_91001] [SWS_Mfl_91003] [SWS_Mfl_91004]
[SRS_LIBS_00009]	All library functions shall be re-entrant	[SWS_Mfl_00854] [SWS_Mfl_91001] [SWS_Mfl_91003] [SWS_Mfl_91004]
[SRS_LIBS_00011]	All function names and type names shall start with "Library short name_"	[SWS_Mfl_00854] [SWS_Mfl_91001] [SWS_Mfl_91003] [SWS_Mfl_91004]
[SRS_LIBS_00013]	The error cases, resulting in the check at runtime of the value of input parameters, shall be listed in SWS	[SWS_Mfl_00817] [SWS_Mfl_00819]
[SRS_LIBS_00015]	It shall be possible to configure the microcontroller so that the library code is shared between all callers	[SWS_Mfl_00806]
[SRS_LIBS_00017]	Usage of macros should be avoided	[SWS_Mfl_00807]
[SRS_LIBS_00018]	A library function may only call library functions	[SWS_Mfl_00808]

Table 6.1: RequirementsTracing

7 Functional specification

7.1 Error Classification

[SWS_Mfl_00821] ⌈Section 7.1 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.⌋*()*

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.1.1 Development Errors

There are no development errors.

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Transient Faults

There are no transient faults.

7.1.4 Production Errors

There are no production errors.

7.1.5 Extended Production Errors

There are no extended production errors.

7.2 Error detection

[SWS_Mfl_00819] ⌈Error detection: The validity of the parameters passed to library functions must be checked at the application level, there is no error detection or reporting within the library function. The library functions are required return a predefined but mathematically senseless value when they are called with invalid parameters. Warning, this strategy has the unsound consequence of masking errors throughout the software development process. All the invalid input cases shall be listed in the SWS specifying a predefined function return value that is not configurable. This value is dependant of the function and the error case so it is determined case by case.

If values passed to the routines are not valid and out of the function specification, then such error are not detected.⌋*(SRS_LIBS_00013)*

E.g. If passed value > 32 for a bit-position
or a negative number of samples of an axis distribution is passed to a routine.

7.3 Error notification

[SWS_Mfl_00817] ⌈The functions shall not call the DET for error notification.⌋*(SRS_-LIBS_00013)*

7.4 Initialization and shutdown

[SWS_Mfl_00800] ⌈Mfl library shall not require initialization phase. A Library function may be called at the very first step of ECU initialization, e.g. even by the OS or EcuM, thus the library shall be ready.⌋*(SRS_LIBS_00002)*

[SWS_Mfl_00801] ⌈Mfl library shall not require a shutdown operation phase.⌋*(SRS_-LIBS_00003)*

7.5 Using Library API

Mfl API can be directly called from BSW modules or SWC. No port definition is required. It is a pure function call.

The statement 'Mfl.h' shall be placed by the developer or an application code generator but not by the RTE generator

Using a library should be documented. if a BSW module or a SWC uses a Library, the developer should add an Implementation-DependencyOnArtifact in the BSW/SWC template.

minVersion and maxVersion parameters correspond to the supplier version. In case of AUTOSAR library, these parameters may be left empty because a SWC or BSW module may rely on a library behavior, not on a supplier implementation. However, the SWC or BSW modules shall be compatible with the AUTOSAR platform where they are integrated.

7.6 Library implementation

[SWS_Mfl_00806] ⌈The Mfl library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋*(SRS_LIBS_00015)*

[SWS_Mfl_00807] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used.⌋*(SRS_LIBS_00017)*

[SWS_Mfl_00808] ⌈A library function shall not call any BSW modules functions, e.g. the DET. A library function can call other library functions. Because a library function shall be re-entrant. But other BSW modules functions may not be re-entrant.⌋*(SRS_LIBS_00018)*

[SWS_Mfl_00809] ⌈The library, written in C programming language, should conform to the MISRA C Standard.

Please refer to SWS_BSW_00115 for more details.⌋*(SRS_BSW_00007)*

[SWS_Mfl_00810] ⌈Each AUTOSAR library Module implementation <library>*.c and

<library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism.⌋*(SRS_BSW_00437)*

[SWS_Mfl_00811] ⌈Each AUTOSAR library Module implementation <library>*.c, that uses AUTOSAR integer data types and/or the standard return, shall include the header file Std_Types.h.⌋*(SRS_BSW_00348)*

[SWS_Mfl_00812] ⌈All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of native C data types, unless this library is clearly identified to be compliant only with a platform.⌋*(SRS_BSW_00304, SRS_BSW_00378)*

[SWS_Mfl_00813] ⌈All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, unless this library is clearly identified to be compliant only with a platform. eg. #pragma, typeof etc.⌋*(SRS_BSW_00306)*

[SWS_Mfl_00820] ⌈

Note: The following functions are exact equivalents to the C99 Standard library functions. A detailed description can be found here:[10](ISO/IEC 9899:1990 Programming Language - C)

Mfl_Sin_f32(float32) <=> sinf(float)

Mfl_Cos_f32(float32) <=> cosf(float)

Mfl_Tan_f32(float32) <=> tanf(float)

Mfl_Exp_f32(float32) <=> expf(float)

Mfl_Log_f32(float32) <=> logf(float)

Mfl_ArcSin_f32(float32) <=> asinf(float)

Mfl_ArcCos_f32(float32) <=> acosf(float)

Mfl_ArcTan_f32(float32) <=> atanf(float)

Mfl_ArcTan2_f32(float32) <=> atan2f(float)

Mfl_Hypot_f32f32_f32 <=> hypotf⌋*()*

8 Routine specification

8.1 Imported types

In this chapter, all types included from the following modules are listed:

Module	Imported Type
Std_Types.h	boolean, sint8, uint8, sint16, uint16, sint32, uint32, float32

8.2 Type definitions

It is observed that since the sizes of the integer types provided by the C language are implementation-defined, the range of values that may be represented within each of the integer types will vary between implementations.

Thus, in order to improve the portability of the software these types are defined in Platform_Types.h [AUTOSAR_SWS_PlatformTypes]. The following mnemonic are used in the library routine names.

Size	Platform Type	Mnemonic	Range
unsigned 8-Bit	boolean	u8	[TRUE, FALSE]
signed 8-Bit	sint8	s8	[-128, 127]
signed 16-Bit	sint16	s16	[-32768, 32767]
signed 32-Bit	sint32	s32	[-2147483648, 2147483647]
unsigned 8-Bit	uint8	u8	[0, 255]
unsigned 16-Bit	uint16	u16	[0, 65535]
unsigned 32-Bit	uint32	u32	[0, 4294967295]
32-Bit	float32	f32	[-3.4028235E38, 3.4028235E38]

Table 8.1: Mnemonic for Base Types

As a convention in the rest of the document:

- mnemonics will be used in the name of the routines (using <InTypeMn1> that means Type Mnemonic for Input 1)
- the real type will be used in the description of the prototypes of the routines (using <InType1> or <OutType>).

Note:

The naming convention for the api's with boolean return type/parameter type is given as _u8 which shall be interpreted as _b. (Boolean)

If there is no boolean data type present in the return type/parameter type then _u8 shall be interpreted as _u8 only.

8.3 Comment about rounding

Two types of rounding can be applied:

Results are 'rounded off', it means:

- 0 <= X < 0.5 rounded to 0
- 0.5 <= X < 1 rounded to 1
- -0.5 < X <= 0 rounded to 0
- -1 < X <= -0.5 rounded to -1

Results are rounded towards zero.

- 0 <= X < 1 rounded to 0
- -1 < X <= 0 rounded to 0

8.4 Comment about routines optimized for target

The routines described in this library may be realized as regular routines or inline functions. For ROM optimization purposes, it is recommended that the c routines be realized as individual source files so they may be linked in on an as-needed basis.

For example, depending on the target, two types of optimization can be done:

- Some routines can be replaced by another routine using integer promotion.
- Some routines can be replaced by the combination of a limiting routine and a routine with a different signature.

142

8.5 Routine definitions

8.5.1 Floating point to Fixed-Point Conversion

[SWS_Mfl_00005] ⌈

Service Name	Mfl_Cvrt_f32_<OutTypeMn>	
Syntax	`<OutType> Mfl_Cvrt_f32_<OutTypeMn> ( float32 ValFloat, sint16 ValFixedExponent )`	
Service ID [hex]	0x01 to 0x04	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValFloat	Floating-point quantity to be converted.
	ValFixedExponent	Exponent of the fixed-point result of the conversion.
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Returns the integer value of the fixed-point result
Description	Returns the integer value of the fixed point result of the conversion, determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00006] ⌈

Result = ValFloat * 2ValFixedExponent

⌋*()*

[SWS_Mfl_00007] ⌈

The return value shall be saturated to the return type boundary values in the event of overflow or underflow.

⌋*()*

[SWS_Mfl_00008] ⌈

If it is necessary to round the result of this equation, it is rounded toward zero.

⌋*()*

Function ID and prototypes

[SWS_Mfl_00009] ⌈

Function ID[hex]	Function prototype
0x01	uint16 Mfl_Cvrt_f32_u16(float32, sint16)
0x02	sint16 Mfl_Cvrt_f32_s16(float32, sint16)
0x03	uint32 Mfl_Cvrt_f32_u32(float32, sint16)
0x04	sint32 Mfl_Cvrt_f32_s32(float32, sint16)

⌋()

8.5.2 Fixed-Point to Floating-Point Conversion

[SWS_Mfl_00010] ⌈

Service Name	Mfl_Cvrt_<InTypeMn>_f32	

Syntax	`float32 Mfl_Cvrt_<InTypeMn>_f32 (`
	`  <InType> ValFixedInteger,`
	`  sint16 ValFixedExponent`
	`)`	

Service ID [hex]	0x05 to 0x08	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValFixedInteger	Integer value of the fixed-point quantity to be converted
	ValFixedExponent	Exponent of the fixed-point quantity to be converted.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	The floating-point result of the conversion.
Description	Returns the floating-point result of the conversion, determined according to the following equation.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00011] ⌈

Result = ValFixedInteger * 2-ValFixedExponent

⌋()

Function ID and prototypes

[SWS_Mfl_00012] ⌈

Function ID[hex]	Function prototype
0x05	float32 Mfl_Cvrt_u16_f32(uint16, sint16)
0x06	float32 Mfl_Cvrt_s16_f32(sint16, sint16)
0x07	float32 Mfl_Cvrt_u32_f32(uint32, sint16)
0x08	float32 Mfl_Cvrt_s32_f32(sint32, sint16)

⌋()

8.5.3 Rounding

[SWS_Mfl_00013] ⌈

Service Name	Mfl_Trunc_f32	
Syntax	`float32 Mfl_Trunc_f32 ( float32 ValValue )`	
Service ID [hex]	0x09	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Truncated value
Description	Returns the integer value determined by rounding the argument toward zero.	
Available via	Mfl.h	

⌋*()* For example:

36.56 will be truncated to 36.00

[SWS_Mfl_00015] ⌈

Service Name	Mfl_Round_f32	
Syntax	`float32 Mfl_Round_f32 ( float32 ValValue )`	
Service ID [hex]	0x0A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Rounded value of operand.
Description	Returns the integer value determined by rounding the argument toward the nearest whole number.	
Available via	Mfl.h	

⌋*()* For example:

36.56 will be rounded to 37.00

[SWS_Mfl_00017] ⌈

If the argument is halfway between two integers, it is rounded away from zero.

For example:

36.5 will be rounded to 37.00

⌋*()*

[SWS_Mfl_00018] ⌈

Service Name	Mfl_Ceil_f32	
Syntax	`float32 Mfl_Ceil_f32 ( float32 ValValue )`	
Service ID [hex]	0x0B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Ceiling of the ValValue.
Description	Returns the integer value determined by rounding the argument toward positive infinity.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00020] ⌈

Service Name	Mfl_Floor_f32	
Syntax	`float32 Mfl_Floor_f32 ( float32 ValValue )`	
Service ID [hex]	0x0C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Operand rounded to floor.
Description	Returns the natural number value determined by rounding the argument toward negative infinity.	
Available via	Mfl.h	

⌋*()*

8.5.4 Controller routines

Controller routines includes P, PT1, DT1, PD, I, PI, PID governors used in control system applications. For these controllers, the required parameters are derived using Laplace-Z transformation. The following parameters are required to calculate the new controller output yn and can be represented in the following equation.

Yn = a1 * Yn-1 + b0 * Xn + b1 * Xn-1 + b2 * Xn-2 + ... + bn-1 * X1 + bn * X0

In the equation, the following symbols are used

Symbols	Description
Yn	Actual output to calculate
Yn-1	Output value, one time step before
Xn	Actual input, given from the input
Xn-1	Input, one time step before
Xn-2	Input, two time steps before
X1	Input, n-1 time steps before
X0	Input, n time steps before
a1, b0, b1, b2, bn-1, bn	Controller dependent proportional parameters are used to describe the weight of the states.

8.5.4.1 Structure definitions for controller routines

System parameters are separated from time or time equivalent parameters. The system parameters are grouped in controller dependent structures Mfl_Param<controller>_Type, whereas the time (equivalent) parameters are assigned directly. Systems states are grouped in a structure Mfl_State<controller>_Type except the actual input value Xn which is assigned directly.

The System parameters, used in the equations are given by:

K : Amplification factor, the description of the semantic is given in

T1 : Decay time constant

Tv : Lead time

Tn : Follow-up time

The time & time equivalent parameters in the equation / implementation are given by:

dT : Time step = sampling interval

Analogous to the abbreviations above, the following abbreviations are used in the implementation:

K_<size>, K_C : Amplification factor

T1rec_<size> : Reciprocal delay time constant = 1/ T1

Tv _<size>, Tv _C : Lead time

Tnrec _<size>, Tnrec _C : Reciprocal follow-up time = 1/ Tn.

dT_<size> : Time step = sampling interval

TeQ_<size> : Time equivalent = exp (-dT/ T1).

Herein "<size>" denotes the size of the variable, e.g _f32 stand for a float32 bit variable.

Following C-structures are specially defined for the controller routines.

[SWS_Mfl_00025] ⌈

Name	Mfl_StatePT1_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for PT1 controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00823] ⌈

Name	Mfl_StateDT1Typ1_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	X2	
	Type	float32
	Comment	Input value, two time steps before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for DT1-Type1 controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00824] ⌈

Name	Mfl_StateDT1Typ2_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for DT1-Type2 controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00825] ⌈

Name	Mfl_StatePD_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for PD controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00826] ⌈

Name	Mfl_ParamPD_Type	
Kind	Structure	
Elements	K_C	
	Type	float32
	Comment	Amplification factor
	Tv_C	
	Type	float32
	Comment	Lead time
Description	System and Time equivalent parameter Structure for PD controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00827] ⌈

Name	Mfl_StateI_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for I controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00828] ⌈

Name	Mfl_StatePI_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for PI additive (*Type1 and Type 2*) controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00829] ⌈

Name	Mfl_ParamPI_Type	
Kind	Structure	
Elements	K_C	
	Type	float32
	Comment	Amplification factor
	Tnrec_C	
	Type	float32
	Comment	Reciprocal follow up time (1/Tn)
Description	System and Time equivalent parameter Structure for PI additive (*Type1 and Type 2*) controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00830] ⌈

Name	Mfl_StatePID_Type	
Kind	Structure	
Elements	X1	
	Type	float32
	Comment	Input value, one time step before
	X2	
	Type	float32
	Comment	Input value, two time step before
	Y1	
	Type	float32
	Comment	Output value, one time step before
Description	System State Structure for PID additive (*Type1 and Type 2*) controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00831] ⌈

Name	Mfl_ParamPID_Type	
Kind	Structure	
Elements	K_C	
	Type	float32
	Comment	Amplification factor
	Tv_C	
	Type	float32
	Comment	Lead time
	Tnrec_C	
	Type	float32
	Comment	Reciprocal follow up time (1/Tn)
Description	System and Time equivalent parameter Structure for PID additive (*Type1 and Type 2*) controller routine	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00832] ⌈

Name	Mfl_Limits_Type	
Kind	Structure	
Elements	Min_C	
	Type	float32
	Comment	Minimum limit value
	Max_C	
	Type	float32
	Comment	Maximum limit value
Description	Controller limit value structure	
Available via	Mfl.h	

⌋*()*

8.5.4.2 Proportional Controller

Proportional component calculates Y(x) = Kp * X.

1. 'P' Controller

[SWS_Mfl_00026] ⌈

Service Name	Mfl_PCalc	
Syntax	`void Mfl_PCalc ( float32 X_f32, float32* P_pf32, float32 K_f32 )`	
Service ID [hex]	0x10	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	input value
	K_f32	Amplification factor
Parameters (inout)	P_pf32	Pointer to the calculated state
Parameters (out)	None	
Return value	None	
Description	Differential equation: Y = K * X	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00027] ⌈

Implemented difference equation:

*P_pf32 = K_f32 * X_f32

1. Get 'P' output

This routine can be realised using inline function.

⌋*()*

[SWS_Mfl_00030] ⌈

Service Name	Mfl_POut_f32	
Syntax	`float32 Mfl_POut_f32 ( const float32* P_pf32 )`	
Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	P_pf32	Pointer to the calculated state
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'P' controller output value

▽

△

Description	This routine returns 'P' controllers output value limited by the return data type
Available via	Mfl.h

⌋()

[SWS_Mfl_00031] ⌈

Output value = *P_pf32

⌋()

8.5.4.3 Proportional controller with first order time constant

This routine calculates proportional element with first order time constant.

Routine Mfl_CalcTeQ_f32, given in [REF], shall be used for Mfl_PT1Calc function to calculate the time equivalent TeQ_f32.

1. 'PT1' Controller

[SWS_Mfl_00032] ⌈

Service Name	Mfl_PT1Calc	

Syntax	`void Mfl_PT1Calc (`
	`  float32 X_f32,`
	`  Mfl_StatePT1_Type* State_cpst,`
	`  float32 K_f32,`
	`  float32 TeQ_f32`
	`)`	

Service ID [hex]	0x1A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the PT1 element
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
Parameters (inout)	State_cpst	Pointer to PT1 state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes PT1 controller output value using below difference equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00033] ⌈

Yn= exp(-dT/T1) * Yn-1+ K(1- exp(-dT/T1)) * Xn-1

This derives implementation:

Output_value = (TeQ_f32 * State_cpst->Y1) + K_f32 * (1 - TeQ_f32) * State_cpst->X1 where TeQ_f32 = exp (-dT/T1)

⌋()

[SWS_Mfl_00035] ⌈

If (TeQ_f32 = 0) then PT1 controller follows Input value,

State_cpst->Y1 = K_f32 * X_f32

⌋()

[SWS_Mfl_00036] ⌈

calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.

State_cpst->Y1 = Output_value

State_cpst->X1 = X_f32

1. 'PT1' Set State Value

This routine can be realised using inline function.

⌋()

[SWS_Mfl_00037] ⌈

Service Name	Mfl_PT1SetState	
Syntax	`void Mfl_PT1SetState ( Mfl_StatePT1_Type* State_cpst, float32 X1_f32, float32 Y1_f32 )`	
Service ID [hex]	0x1B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a PT1 element.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00038] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋()

[SWS_Mfl_00039] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32.

1. Calculate time equivalent Value

This routine can be realised using inline function.

⌋*()*

[SWS_Mfl_00040] ⌈

Service Name	Mfl_CalcTeQ_f32	
Syntax	`float32 Mfl_CalcTeQ_f32 ( float32 T1rec_f32, float32 dT_f32 )`	
Service ID [hex]	0x1C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	T1rec_f32	Reciprocal delay time
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Time Equivalent TeQ_f32
Description	This routine calculates time equivalent factor	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00041] ⌈

TeQ_f32 = exp(-T1rec_f32 * dT_f32)

1. Calculate an approximate time equivalent Value

This routine calculates approximate time equivalent and can be realised using inline function⌋*()*

[SWS_Mfl_00315] ⌈

Service Name	Mfl_CalcTeQApp_f32	
Syntax	`float32 Mfl_CalcTeQApp_f32 ( float32 T1rec_f32, float32 dT_f32 )`	
Service ID [hex]	0x1E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	T1rec_f32	Reciprocal delay time
	dT_f32	Sample Time
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	float32	Time Equivalent TeQApp_f32
Description	This routine calculates time equivalent factor	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00316] ⌈

TeQApp_f32 = 1 - (T1rec_f32 * dT_f32)

1. Get 'PT1' output

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00042] ⌈

Service Name	Mfl_PT1Out_f32	

Syntax	`float32 Mfl_PT1Out_f32 (`
	`  const Mfl_StatePT1_Type* State_cpst`
	`)`	

Service ID [hex]	0x1D	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'PT1' controller output value
Description	This routine returns 'PT1' controllers output value	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00043] ⌈

Output value = State_cpst->Y1

⌋*()*

8.5.4.4 Differential component with time delay : DT1

This routine calculates differential element with first order time constant.

Routine Mfl_CalcTeQ_f32, given in [REF], shall be used for Mfl_DT1Typ1Calc and Mfl_DT1Typ2Calc functions to calculate the time equivalent TeQ_f32.

1. 'DT1' Controller - Type1

[SWS_Mfl_00044] ⌈

Service Name	Mfl_DT1Typ1Calc	
Syntax	`void Mfl_DT1Typ1Calc ( float32 X_f32, Mfl_StateDT1Typ1_Type* State_cpst, float32 K_f32, float32 TeQ_f32, float32 dT_f32 )`	
Service ID [hex]	0x20	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the DT1 controller
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes DT1 controller output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00045] ⌈

Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn-1 - Xn-2) / dT)

This derives implementation:

Output_value = (TeQ_f32 * State_cpst->Y1) + K_f32 * (1 - TeQ_f32) * ((State_cpst->X1 - State_cpst->X2) / dT_f32)

where TeQ_f32 = exp(-dT_f32/T1)

⌋*()*

[SWS_Mfl_00047] ⌈

If (TeQ_f32 = 0) then DT1 controller follows Input value,

Output_value = K_f32 * (X_f32 - State_cpst->X1) / dT_f32

⌋*()*

[SWS_Mfl_00048] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00049] ⌈

Old input value State_cpst->X1 shall be stored to State_cpst->X2.

State_cpst->X2 = State_cpst->X1

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. 'DT1' Controller - Type2

⌋()

[SWS_Mfl_00300] ⌈

Service Name	Mfl_DT1Typ2Calc	
Syntax	`void Mfl_DT1Typ2Calc ( float32 X_f32, Mfl_StateDT1Typ2_Type* State_cpst, float32 K_f32, float32 TeQ_f32, float32 dT_f32 )`	
Service ID [hex]	0xC0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the DT1 controller
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes DT1 controller output value using differential equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00301] ⌈

Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn - Xn-1) / dT)

This derives implementation:

Output_value = (TeQ_f32 * State_cpst->Y1) + K_f32 * (1 - TeQ_f32) * ((X_f32 - State_cpst->X1) / dT_f32)

where TeQ_f32 = exp(-dT_f32/T1)

⌋()

[SWS_Mfl_00303] ⌈

If (TeQ_f32 = 0) then DT1 controller follows Input value,

Output_value = K_f32 * (X_f32 - State_cpst->X1) / dT_f32

⌋()

[SWS_Mfl_00304] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋()

[SWS_Mfl_00305] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32x

1. Set 'DT1' State Value - Type1

This routine can be realised using inline function.⌋()

[SWS_Mfl_00050] ⌈

Service Name	Mfl_DT1Typ1SetState	
Syntax	`void Mfl_DT1Typ1SetState ( Mfl_StateDT1Typ1_Type* State_cpst, float32 X1_f32, float32 X2_f32, float32 Y1_f32 )`	
Service ID [hex]	0x22	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for the input state X1
	X2_f32	Initial value for the input state X2
	Y1_f32	Initial value for the output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a DT1 element.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00051] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋()

[SWS_Mfl_00052] ⌈

Initialisation of input state variables X1 and X2.

State_cpst->X1 = X1_f32

State_cpst->X2 = X2_f32

1. Set 'DT1' State Value - Type2

This routine can be realised using inline function.⌋()

[SWS_Mfl_00306] ⌈

Service Name	Mfl_DT1Typ2SetState	
Syntax	`void Mfl_DT1Typ2SetState ( Mfl_StateDT1Typ2_Type* State_cpst, float32 X1_f32, float32 Y1_f32 )`	
Service ID [hex]	0xC1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for the input state
	Y1_f32	Initial value for the output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a DT1 element.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00307] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋()

[SWS_Mfl_00308] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

1. Get 'DT1' output - Type1

This routine can be realised using inline function.⌋()

[SWS_Mfl_00053] ⌈

Service Name	Mfl_DT1Typ1Out_f32	
Syntax	`float32 Mfl_DT1Typ1Out_f32 ( const Mfl_StateDT1Typ1_Type* State_cpst )`	
Service ID [hex]	0x23	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'DT1' controller output value
Description	This routine returns 'DT1' controller's output value	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00054] ⌈

Output value = State_cpst->Y1

1. Get 'DT1' output - Type2

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00310] ⌈

Service Name	Mfl_DT1Typ2Out_f32	
Syntax	`float32 Mfl_DT1Typ2Out_f32 ( const Mfl_StateDT1Typ2_Type* State_cpst )`	
Service ID [hex]	0xC2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'DT1' controller output value
Description	This routine returns 'DT1' controller's output value	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00311] ⌈

Output value = State_cpst->Y1

⌋*()*

8.5.4.5 Proportional & Differential controller

This routine is a combination of proportional & differential controller.

1. PD Controller

[SWS_Mfl_00055] ⌈

Service Name	Mfl_PDCalc	
Syntax	`void Mfl_PDCalc ( float32 X_f32, Mfl_StatePD_Type* State_cpst, const Mfl_ParamPD_Type* Param_cpst, float32 dT_f32 )`	
Service ID [hex]	0x2A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the PD controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes proportional plus derivative controller output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00056] ⌈

Yn= K(1+Tv/dT) * Xn- K(Tv/dT) * Xn-1

This derives implementation:

Output_value = (Param_cpst->K_C * (1+ Param_cpst->Tv_C/dT_f32) * X_f32) - (Param_cpst->K_C * (Param_cpst->Tv_C/dT_f32) * State_cpst->X1)

⌋*()*

[SWS_Mfl_00057] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00058] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. PD Set State Value

This routine can be realised using inline function.⌋()

[SWS_Mfl_00059] ⌈

Service Name	Mfl_PDSetState	
Syntax	`void Mfl_PDSetState ( Mfl_StatePD_Type* State_cpst, float32 X1_f32, float32 Y1_f32 )`	
Service ID [hex]	0x2B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a PD element.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00060] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋()

[SWS_Mfl_00061] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

1. Set 'PD' Parameters

This routine can be realised using inline function.⌋()

[SWS_Mfl_00062] ⌈

Service Name	Mfl_PDSetParam	
Syntax	`void Mfl_PDSetParam ( Mfl_ParamPD_Type* Param_cpst, float32 K_f32, float32 Tv_f32 )`	
Service ID [hex]	0x2C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_f32	Amplification factor
	Tv_f32	Lead time

▽

△

Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	None	
Description	The routine sets the parameter structure of a PD element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00063] ⌈

Initialisation of amplification factor.

Param_cpst->K_C = K_f32

⌋*()*

[SWS_Mfl_00064] ⌈

Initialisation of lead time state variable

Param_cpst->Tv_C = Tv_f32

1. Get 'PD' output

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00066] ⌈

Service Name	Mfl_PDOut_f32	

Syntax	`float32 Mfl_PDOut_f32 (`
	`  const Mfl_StatePD_Type* State_cpst`
	`)`	

Service ID [hex]	0x2D	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'PD' controller output value
Description	This routine returns 'PD' controllers output value.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00067] ⌈

Output value = State_cpst->Y1

⌋*()*

8.5.4.6 Integral component

This routine calculates Integration element.

1. 'I' Controller

[SWS_Mfl_00068] ⌈

Service Name	Mfl_ICalc	
Syntax	`void Mfl_ICalc ( float32 X_f32, Mfl_StateI_Type* State_cpst, float32 K_f32, float32 dT_f32 )`	
Service ID [hex]	0x30	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'I' controller
	K_f32	Amplification factor
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to state variable.
Return value	None	
Description	This routine computes I controller output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00069] ⌈

Yn= Yn-1 + K * dT * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + K_f32 * dT_f32 * State_cpst->X1

⌋*()*

[SWS_Mfl_00070] ⌈

Calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.

State_cpst->Y1 = Output_value

State_cpst->X1 = X_f32

1. 'I' Controller with limitation

⌋*()*

[SWS_Mfl_00320] ⌈

Service Name	Mfl_ILimCalc	
Syntax	`void Mfl_ILimCalc ( float32 X_f32, Mfl_StateI_Type* State_cpst, float32 K_f32, const Mfl_Limits_Type* Limit_cpst, float32 dT_f32 )`	
Service ID [hex]	0x32	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'I' controller
	K_f32	Amplification factor
	Limit_cpst	Pointer to limit structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to state variable
Parameters (out)	None	
Return value	None	
Description	This routine computes I controller output value using differential equation	
Available via	Mfl.h	

⌋ *()*

[SWS_Mfl_00321] ⌈

Yn= Yn-1 + K * dT * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + K_f32 * dT_f32 * State_cpst->X1

⌋ *()*

[SWS_Mfl_00322] ⌈

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

⌋ *()*

[SWS_Mfl_00323] ⌈

Calculated Output_value and current input value shall be stored to State_cpst->Y1 and State_cpst->X1 respectively.

State_cpst->Y1 = Output_value

State_cpst->X1 = X_f32

1. Set limits for controllers

⌋()

[SWS_Mfl_00324] ⌈

Service Name	Mfl_CtrlSetLimit

Syntax	`void Mfl_CtrlSetLimit (`
	`  float32 Min_f32,`
	`  float32 Max_f32,`
	`  Mfl_Limits_Type* Limit_cpst`
	`)`

Service ID [hex]	0x34
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	Min_f32 — Minimum limit
	Max_f32 — Maximum limit
Parameters (inout)	Limit_cpst — Pointer to limit structure
Parameters (out)	None
Return value	None
Description	Update limit structure
Available via	Mfl.h

⌋()

[SWS_Mfl_00325] ⌈

Update limit structure

Limit_cpst->Min_C = Min_f32

Limit_cpst->Max_C = Max_f32

⌋()

Note : "This routine (Mfl_CtrlSetLimit) is depreciated and will not be supported in future release

Replacement routine : Mfl_CtrlSetLimits "

[SWS_Mfl_00367] ⌈

Service Name	Mfl_CtrlSetLimits	
Syntax	`void Mfl_CtrlSetLimits ( Mfl_Limits_Type* Limit_cpst, float32 Min_f32, float32 Max_f32 )`	
Service ID [hex]	0xC9	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Min_f32	Minimum limit
	Max_f32	Maximum limit
Parameters (inout)	Limit_cpst	Pointer to limit structure
Parameters (out)	None	
Return value	None	
Description	Update limit structure	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00368] ⌈

Update limit structure

Limit_cpst->Min_C = Min_f32

Limit_cpst->Max_C = Max_f32

1. Set 'I' State Value

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00071] ⌈

Service Name	Mfl_ISetState	
Syntax	`void Mfl_ISetState ( Mfl_StateI_Type* State_cpst, float32 X1_f32, float32 Y1_f32 )`	
Service ID [hex]	0x31	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of an I element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00072] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋()

[SWS_Mfl_00073] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

1. Get 'I' output

This routine can be realised using inline function.⌋()

[SWS_Mfl_00074] ⌈

Service Name	Mfl_IOut_f32	
Syntax	`float32 Mfl_IOut_f32 ( const Mfl_StateI_Type* State_cpst )`	
Service ID [hex]	0x33	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'I' controller output value
Description	This routine returns 'I' controllers output value.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00075] ⌈

Output value = State_cpst->Y1

⌋()

8.5.4.7 Proportional & Integral controller

This routine is a combination of Proportional & Integral controller.

8.5.4.7.1 'PI' Controller - Type1 (Implicit type)

[SWS_Mfl_00076] ⌈

Service Name	Mfl_PITyp1Calc	

Syntax	`void Mfl_PITyp1Calc (`
	`float32 X_f32,`
	`Mfl_StatePI_Type* State_cpst,`
	`const Mfl_ParamPI_Type* Param_cpst,`
	`float32 dT_f32`
	`)`	

Service ID [hex]	0x35	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to the internal state structure.
Return value	None	
Description	This routine computes Proportional plus integral controller (implicit type) output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00077] ⌈

Yn= Yn-1+ K * Xn- K * (1 - dT/Tn) * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_f32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_f32) * State_cpst->X1)

⌋*()*

[SWS_Mfl_00078] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00079] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

⌋*()*

8.5.4.7.2 'PI' Controller - Type1 with limitation (Implicit type)

[SWS_Mfl_00326] ⌈

Service Name	Mfl_PITyp1LimCalc	
Syntax	`void Mfl_PITyp1LimCalc ( float32 X_f32, Mfl_StatePI_Type* State_cpst, const Mfl_ParamPI_Type* Param_cpst, const Mfl_Limits_Type* Limit_cpst, float32 dT_f32 )`	
Service ID [hex]	0xC3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes Proportional plus integral controller (implicit type) output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00327] ⌈

Yn= Yn-1+ K * Xn- K * (1 - dT/Tn) * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_f32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_f32) * State_cpst->X1)

⌋*()*

[SWS_Mfl_00328] ⌈

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

⌋*()*

[SWS_Mfl_00329] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋()

[SWS_Mfl_00330] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

⌋()

8.5.4.7.3 'PI' Controller - Type2 (Explicit type)

[SWS_Mfl_00080] ⌈

Service Name	Mfl_PITyp2Calc	
Syntax	`void Mfl_PITyp2Calc ( float32 X_f32, Mfl_StatePI_Type* State_cpst, const Mfl_ParamPI_Type* Param_cpst, float32 dT_f32 )`	
Service ID [hex]	0x36	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to the internal state structure.
Return value	None	
Description	This routine computes Proportional plus integral controller (explicit type) output value using differential equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00081] ⌈

Yn= Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_f32) * X_f32) - (Param_cpst->K_C * State_cpst->X1)

⌋()

[SWS_Mfl_00082] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋()

[SWS_Mfl_00083] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

⌋()

8.5.4.7.4 'PI' Controller - Type2 with limitation (Explicit type)

[SWS_Mfl_00331] ⌈

Service Name	Mfl_PITyp2LimCalc	
Syntax	`void Mfl_PITyp2LimCalc ( float32 X_f32, Mfl_StatePI_Type* State_cpst, const Mfl_ParamPI_Type* Param_cpst, const Mfl_Limits_Type* Limit_cpst, float32 dT_f32 )`	
Service ID [hex]	0xC4	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes Proportional plus integral controller (explicit type) output value using differential equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00332] ⌈

Yn= Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1

This derives implementation:

Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_f32) * X_f32) - (Param_cpst->K_C * State_cpst->X1)

⌋()

[SWS_Mfl_00333] ⌈

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

⌋*()*

[SWS_Mfl_00334] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00335] ⌈

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. Set 'PI' State Value

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00084] ⌈

Service Name	Mfl_PISetState	
Syntax	`void Mfl_PISetState ( Mfl_StatePI_Type* State_cpst, float32 X1_f32, float32 Y1_f32 )`	
Service ID [hex]	0x37	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a PI element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00085] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋*()*

[SWS_Mfl_00086] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

1. Set 'PI' Parameters

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00087] ⌈

Service Name	Mfl_PISetParam	
Syntax	`void Mfl_PISetParam ( Mfl_ParamPI_Type* Param_cpst, float32 K_f32, float32 Tnrec_f32 )`	
Service ID [hex]	0x38	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_f32	Amplification factor
	Tnrec_f32	Reciprocal follow-up time
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	None	
Description	The routine sets the parameter structure of a PI element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00088] ⌈

Initialisation of amplification factor.

Param_cpst->K_C = K_f32

⌋*()*

[SWS_Mfl_00089] ⌈

Initialisation of reciprocal follow up time state variable

Param_cpst->Tnrec_C = Tnrec_f32

1. Get 'PI' output

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00090] ⌈

Service Name	Mfl_PIOut_f32	
Syntax	`float32 Mfl_PIOut_f32 ( const Mfl_StatePI_Type* State_cpst )`	
Service ID [hex]	0x39	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'PI' controller output value
Description	This routine returns 'PI' controllers output value.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00091] ⌈

Output value = State_cpst->Y1

⌋*()*

8.5.4.8 Proportional, Integral & Differential controller

This routine is a combination of Proportional, integral & differential controller

1. 'PID' Controller - Type1 (Implicit type)

[SWS_Mfl_00092] ⌈

Service Name	Mfl_PIDTyp1Calc	
Syntax	`void Mfl_PIDTyp1Calc ( float32 X_f32, Mfl_StatePID_Type* State_cpst, const Mfl_ParamPID_Type* Param_cpst, float32 dT_f32 )`	
Service ID [hex]	0x3A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to the internal state structure.
Return value	None	

▽

△

Description	This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation
Available via	Mfl.h

⌋*()*

[SWS_Mfl_00093] ⌈

Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2

This derives implementation:

calc1 = Param_cpst->K_C * (1 + t_val) * X_f32

calc2 = Param_cpst->K_C * (1 - dT_f32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1

calc3 = Param_cpst->K_C * t_val * State_cpst->X2

Output_value = State_cpst->Y1 + calc1 - calc2 + calc3

Where t_val = Param_cpst->Tv_C / dT_f32

⌋*()*

[SWS_Mfl_00094] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00095] ⌈

Old input value State_cpst->X1 shall be stored to State_cpst->X2

State_cpst->X2 = State_cpst->X1

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. ’PID’ Controller - Type1 with limitation (Implicit type)

⌋*()*

[SWS_Mfl_00340] ⌈

Service Name	Mfl_PIDTyp1LimCalc	
Syntax	`void Mfl_PIDTyp1LimCalc ( float32 X_f32, Mfl_StatePID_Type* State_cpst, const Mfl_ParamPID_Type* Param_cpst, const Mfl_Limits_Type* Limit_cpst, float32 dT_f32 )`	
Service ID [hex]	0xC5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00341] ⌈

Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2

This derives implementation:

calc1 = Param_cpst->K_C * (1 + t_val) * X_f32

calc2 = Param_cpst->K_C * (1 - dT_f32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1

calc3 = Param_cpst->K_C * t_val * State_cpst->X2

Output_value = State_cpst->Y1 + calc1 - calc2 + calc3

Where t_val = Param_cpst->Tv_C / dT_f32

⌋*()*

[SWS_Mfl_00342] ⌈

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then, Output_value = Limit_cpst->Max_C

⌋()

[SWS_Mfl_00343] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋()

[SWS_Mfl_00344] ⌈

Old input value State_cpst->X1 shall be stored to State_cpst->X2

State_cpst->X2 = State_cpst->X1

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. 'PID' Controller - Type2 (Explicit type)

⌋()

[SWS_Mfl_00096] ⌈

Service Name	Mfl_PIDTyp2Calc	
Syntax	`void Mfl_PIDTyp2Calc ( float32 X_f32, Mfl_StatePID_Type* State_cpst, const Mfl_ParamPID_Type* Param_cpst, float32 dT_f32 )`	
Service ID [hex]	0x3B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to the internal state structure
Return value	None	
Description	This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00097] ⌈

Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2

This derives implementation:

calc1 = Param_cpst->K_C * (1 + dT_f32 * Param_cpst->Tnrec_C + t_val) * X_f32

calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1

calc3 = Param_cpst->K_C * t_val * State_cpst->X2

Output_value = State_cpst->Y1 + calc1 - calc2 + calc3

Where t_val = Param_cpst->Tv_C / dT_f32

⌋*()*

[SWS_Mfl_00098] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00099] ⌈

Old input value State_cpst->X1 shall be stored to State_cpst->X2

State_cpst->X2 = State_cpst->X1

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. 'PID' Controller - Type2 with limitation (Explicit type)

⌋*()*

[SWS_Mfl_00345] ⌈

Service Name	Mfl_PIDTyp2LimCalc	
Syntax	`void Mfl_PIDTyp2LimCalc ( float32 X_f32, Mfl_StatePID_Type* State_cpst, const Mfl_ParamPID_Type* Param_cpst, const Mfl_Limits_Type* Limit_cpst, float32 dT_f32 )`	
Service ID [hex]	0xC6	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	Limit_cpst	Pointer to limit structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to the internal state structure
Parameters (out)	None	
Return value	None	
Description	This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00346] ⌈

Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2

This derives implementation:

calc1 = Param_cpst->K_C * (1 + dT_f32 * Param_cpst->Tnrec_C + t_val) * X_f32

calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1

calc3 = Param_cpst->K_C * t_val * State_cpst->X2

Output_value = State_cpst->Y1 + calc1 - calc2 + calc3

Where t_val = Param_cpst->Tv_C / dT_f32

⌋*()*

[SWS_Mfl_00347] ⌈

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

⌋*()*

[SWS_Mfl_00348] ⌈

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

⌋*()*

[SWS_Mfl_00349] ⌈

Old input value State_cpst->X1 shall be stored to State_cpst->X2

State_cpst->X2 = State_cpst->X1

Current input value X_f32 shall be stored to State_cpst->X1.

State_cpst->X1 = X_f32

1. Set 'PID' State Value

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00100] ⌈

Service Name	Mfl_PIDSetState	
Syntax	`void Mfl_PIDSetState ( Mfl_StatePID_Type* State_cpst, float32 X1_f32, float32 X2_f32, float32 Y1_f32 )`	
Service ID [hex]	0x3C	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Initial value for input state
	X2_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (inout)	None	
Parameters (out)	State_cpst	Pointer to internal state structure
Return value	None	
Description	The routine initialises internal state variables of a PID element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00101] ⌈

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

⌋*()*

[SWS_Mfl_00102] ⌈

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

Initialisation of input state variable X2.

State_cpst->X2 = X2_f32

1. Set 'PID' Parameters

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00103] ⌈

Service Name	Mfl_PIDSetParam
Syntax	`void Mfl_PIDSetParam ( Mfl_ParamPID_Type* Param_cpst, float32 K_f32, float32 Tv_f32, float32 Tnrec_f32 )`

▽

△

Service ID [hex]	0x3D	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	K_f32	Amplification factor
	Tv_f32	Lead Time
	Tnrec_f32	Reciprocal follow-up timer
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to internal parameter structure
Return value	None	
Description	The routine sets the parameter structure of a PID element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00104] ⌈

Initialisation of amplification factor.

Param_cpst->K_C = K_f32⌋*()*

[SWS_Mfl_00105] ⌈Initialisation of lead time state variable

Param_cpst->Tv_C = Tv_f32

⌋*()*

[SWS_Mfl_00106] ⌈

Initialisation of reciprocal follow up time state variable

Param_cpst->Tnrec_C = Tnrec_f32

1. Get 'PID' output

This routine can be realised using inline function.⌋*()*

[SWS_Mfl_00107] ⌈

Service Name	Mfl_PIDOut_f32	
Syntax	`float32 Mfl_PIDOut_f32 ( const Mfl_StatePID_Type* State_cpst )`	
Service ID [hex]	0x3E	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return 'PID' controller output value
Description	This routine returns 'PID' controllers output value.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00108] ⌈

Output value = State_cpst->Y1

⌋*()*

8.5.5 Magnitude and Sign

[SWS_Mfl_00110] ⌈

Service Name	Mfl_Abs_f32	
Syntax	`float32 Mfl_Abs_f32 ( float32 ValValue )`	
Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Absolute value of operand.
Description	Returns the absolute value of the argument (ValAbs), determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00111] ⌈

ValAbs = | ValValue |

⌋*()*

[SWS_Mfl_00112] ⌈

Service Name	Mfl_Sign_f32	
Syntax	`sint8 Mfl_Sign_f32 ( float32 ValValue )`	
Service ID [hex]	0x41	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	sint8	Integer representing the sign of the operand.
Description	Returns the sign of the argument (ValSign), determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00113] ⌈

ValSign = 1, ValValue > 0.0

⌋()

[SWS_Mfl_00114] ⌈

ValSign = 0, ValValue == 0.0

⌋()

[SWS_Mfl_00115] ⌈

ValSign = -1, ValValue < 0.0

⌋()

8.5.6 Limiting

[SWS_Mfl_00116] ⌈

Service Name	Mfl_Max_f32	
Syntax	`float32 Mfl_Max_f32 ( float32 ValValue1, float32 ValValue2 )`	
Service ID [hex]	0x45	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue1	Floating-point operand.
	ValValue2	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Maximum value of two arguments.
Description	Returns the value of the larger of the two arguments (ValMax), determined according to the following equation.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00117] ⌈

ValMax = ValValue1, ValValue1 $\geq$ ValValue2

ValMax = ValValue2, ValValue1 < ValValue2

⌋()

[SWS_Mfl_00118] ⌈

Service Name	Mfl_Min_f32	
Syntax	`float32 Mfl_Min_f32 ( float32 Value1, float32 Value2 )`	
Service ID [hex]	0x46	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Value1	Floating-point operand.
	Value2	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Minimum value of two arguments.
Description	Returns the value of the smaller of the two arguments (Min), determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00119] ⌈

Min = Value1, Value1 $\leq$ Value2

Min = Value2, Value1 > Value2

⌋*()*

[SWS_Mfl_00120] ⌈

Service Name	Mfl_RateLimiter_f32	
Syntax	`float32 Mfl_RateLimiter_f32 ( float32 newval, float32 oldval, float32 maxdif )`	
Service ID [hex]	0x47	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	newval	Variable to be limited.
	oldval	Previous value of newval.
	maxdif	Absolute maximum difference allowed between previous value (oldval) and the current value (newval).
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Limited value.
Description	An increasing value and decreasing value is rate limited by maxdif	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00121] ⌈

if (newval > oldval) and ((newval - oldval) > maxdif)

Result = oldval + maxdif

else if (newval < oldval) and ((oldval - newval) > maxdif)

Result = oldval - maxdif

else

Result = newval

⌋()

[SWS_Mfl_00122] ⌈

Service Name	Mfl_Limit_f32	
Syntax	`float32 Mfl_Limit_f32 ( float32 val, float32 lowLim, float32 upLim )`	
Service ID [hex]	0x48	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	val	Quantity to be bounded.
	lowLim	Lower bound. lowLim shall not be strictly greater than upLim.
	upLim	Upper bound. upLim shall not be strictly lower than lowLim.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Limited value.
Description	Returns the bounded value (newVal), determined according to the following equation.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00123] ⌈

newVal = lowLim, val $\leq$ lowLim

newVal = upLim, val $\geq$ upLim

newVal = val, lowLim < val < upLim

⌋()

8.5.7 Logarithms and Exponentials

[SWS_Mfl_00130] ⌈

Service Name	Mfl_Pow_f32	
Syntax	`float32 Mfl_Pow_f32 ( float32 ValBase, float32 ValExp )`	
Service ID [hex]	0x50	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValBase	Base to be raised to an exponent. Valid range:ValBase > 0.0
	ValExp	Exponent by which to raise the base.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	ValBase raised to ValExp power.
Description	Returns the ValBase raised to ValExp power, determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00131] ⌈

ValResult = ValBaseValExp

⌋*()*

[SWS_Mfl_00132] ⌈

If ValExp = 0, and ValBase = 0, ValResult = 1, (00 = 1)

If ValBase = 0 and ValExp <> 0, ValResult = 0, (0ValExp = 0)

⌋*()*

[SWS_Mfl_00133] ⌈

If ValBase and ValExp are having maximum value of type float32, the return value will be toward positive infinity.

⌋*()*

[SWS_Mfl_00135] ⌈

Service Name	Mfl_Sqrt_f32	
Syntax	`float32 Mfl_Sqrt_f32 ( float32 ValValue )`	
Service ID [hex]	0x51	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Square root of ValValue
Description	Returns the square root of the operand (ValSqrt), determined according to the following equation	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00136] ⌈

ValSqrt = ValValue1/2

⌋()

[SWS_Mfl_00137] ⌈

ValValue shall be passed as positive value. (ValValue $\geq$ 0)

⌋()

[SWS_Mfl_00140] ⌈

Service Name	Mfl_Exp_f32	
Syntax	`float32 Mfl_Exp_f32 ( float32 ValValue )`	
Service ID [hex]	0x53	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	e raised to ValValue power
Description	Returns the exponential of the operand (ValExp), determined according to the following equation.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00141] ⌈

ValExp = eValValue

⌋()

[SWS_Mfl_00142] ⌈

Call the function with input value in the range [-24PI, +24PI]

⌋*()*

[SWS_Mfl_00145] ⌈

Service Name	Mfl_Log_f32	
Syntax	`float32 Mfl_Log_f32 ( float32 ValValue )`	
Service ID [hex]	0x54	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValValue	Floating-point operand. Valid range: ValValue > 0.0
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Natural log of ValValue
Description	Returns the natural (base-e) logarithm of the operand (ValLog), determined according to the following equation.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00146] ⌈

ValLog = loge(ValValue)

⌋*()*

[SWS_Mfl_00147] ⌈

ValValue shall be passed as > 0 value.

⌋*()*

8.5.8 Trigonometry

[SWS_Mfl_00150] ⌈

Service Name	Mfl_Sin_f32	
Syntax	`float32 Mfl_Sin_f32 ( float32 value )`	
Service ID [hex]	0x55	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	angle in radians
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	result = sine (value)
Description	Calculates the sine of the argument.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00151] ⌈

Result: result = sine (value)

⌋()

[SWS_Mfl_00152] ⌈

Call the function with input value in the range [-24PI, +24PI]

⌋()

[SWS_Mfl_00155] ⌈

Service Name	Mfl_Cos_f32	
Syntax	`float32 Mfl_Cos_f32 ( float32 value )`	
Service ID [hex]	0x56	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	angle in radians
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	result = cosine (value)
Description	Calculates the cosine of the argument.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00156] ⌈

Result: result = cosine (value)

⌋()

[SWS_Mfl_00157] ⌈

Call the function with input value in the range [-24PI, +24PI]

⌋()

[SWS_Mfl_00160] ⌈

Service Name	Mfl_Tan_f32	
Syntax	`float32 Mfl_Tan_f32 ( float32 value )`	
Service ID [hex]	0x57	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	angle in radians
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	result = tangent(value)

▽

△

Description	Calculates the tangent of the argument.
Available via	Mfl.h

⌋*()*

[SWS_Mfl_00161] ⌈

Result: result = tangent(value)

⌋*()*

[SWS_Mfl_00163] ⌈

Call the function with input value in the range [-24PI, +24PI]

⌋*()*

[SWS_Mfl_00165] ⌈

Service Name	Mfl_arcSin_f32	
Syntax	`float32 Mfl_arcSin_f32 ( float32 value )`	
Service ID [hex]	0x58	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc sine is to be returned
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	The arc sine of the argument, in radians
Description	Returns the arc sine of an angle, in the range of -pi/2 through pi/2.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00167] ⌈

If the argument is zero, then the result is a zero.

⌋*()*

[SWS_Mfl_00168] ⌈

Range of the value shall be [-1, +1]

⌋*()*

Note : "This routine (Mfl_arcSin_f32) is depreciated and will not be supported in future release

Replacement routine : Mfl_ArcSin_f32"

[SWS_Mfl_00350] ⌈

Service Name	Mfl_ArcSin_f32	
Syntax	`float32 Mfl_ArcSin_f32 ( float32 value )`	
Service ID [hex]	0xBC	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc sine is to be returned
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	The arc sine of the argument, in radians
Description	Returns the arc sine of an angle, in the range of -pi/2 through pi/2.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00352] ⌈

If the argument is zero, then the result is a zero.

⌋()

[SWS_Mfl_00353] ⌈

Range of the value shall be [-1, +1]

⌋()

[SWS_Mfl_00170] ⌈

Service Name	Mfl_arcCos_f32	
Syntax	`float32 Mfl_arcCos_f32 ( float32 value )`	
Service ID [hex]	0x59	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc cosine is to be returned
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	The arc cosine of the argument, in radians
Description	Returns the arc cosine of an angle, in the range of 0.0 through pi.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00172] ⌈

Range of the value shall be [-1, +1]

⌋()

Note : "This routine (Mfl_arcCos_f32) is depreciated and will not be supported in future release

Replacement routine : Mfl_ArcCos_f32"

[SWS_Mfl_00354] ⌈

Service Name	Mfl_ArcCos_f32	
Syntax	`float32 Mfl_ArcCos_f32 ( float32 value )`	
Service ID [hex]	0xBD	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc cosine is to be returned
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	The arc cosine of the argument, in radians

▽

△

Description	Returns the arc cosine of an angle, in the range of 0.0 through pi.
Available via	Mfl.h

⌋()

[SWS_Mfl_00356] ⌈

Range of the value shall be [-1, +1]

⌋()

[SWS_Mfl_00175] ⌈

Service Name	Mfl_arcTan_f32	
Syntax	`float32 Mfl_arcTan_f32 ( float32 value )`	
Service ID [hex]	0x5A	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc tan is to be returned.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	the arc tan of the argument, in radians
Description	Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00177] ⌈

If the argument is zero, then the result is a zero with the same sign as the argument.

⌋()

Note : "This routine (Mfl_arcTan_f32) is depreciated and will not be supported in future release

Replacement routine : Mfl_ArcTan_f32"

[SWS_Mfl_00357] ⌈

Service Name	Mfl_ArcTan_f32	
Syntax	`float32 Mfl_ArcTan_f32 ( float32 value )`	
Service ID [hex]	0xBE	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value	The value whose arc tan is to be returned.
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	float32	the arc tan of the argument, in radians
Description	Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00359] ⌈

If the argument is zero, then the result is a zero with the same sign as the argument.

⌋*()*

[SWS_Mfl_00180] ⌈

Service Name	Mfl_arcTan2_f32	
Syntax	`float32 Mfl_arcTan2_f32 ( float32 X1_f32, float32 X2_f32 )`	
Service ID [hex]	0x5B	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X1_f32	Input value 1
	X2_f32	Input value 2
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Returns arctan for inputs X1_f32 & X2_f32
Description	Returns the arc tangent of an angle, in the range of [-pi to pi]	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00182] ⌈

If the argument is zero, then the result is a zero with the same sign as the argument.

⌋*()*

[SWS_Mfl_00183] ⌈

Z = X2_f32 / X1_f32

if (Z > 1) Then

Result = Z / (1.0 + (0.28 * Zˆ2))

if (Z < 1) Then

Result = (pi / 2) - (Z / (Zˆ2 + 0.28))

⌋*()*

Note : "This routine (Mfl_arcTan2_f32) is depreciated and will not be supported in future release

Replacement routine : Mfl_ArcTan2_f32"

[SWS_Mfl_00360] ⌈

Service Name	Mfl_ArcTan2_f32	
Syntax	`float32 Mfl_ArcTan2_f32 ( float32 y, float32 x )`	
Service ID [hex]	0xBF	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	y	y coordinate
	x	x coordinate
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Returns arctan for inputs y and x
Description	Returns the arc tangent of an angle, in the range of [-pi to pi]	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00362] ⌈

If the x coordinate is zero, then check

if(y > 0.0) then

Return PI/2

if(y = 0.0) then

Return Zero

if(y < 0.0) then

Return -PI/2

⌋*()*

[SWS_Mfl_00363] ⌈

Z = y / x

if (|Z| < 1) Then

Result = Z / (1.0 + (0.28 * Zˆ2))

if (x < 0.0f) Then

Result = (y < 0.0f) ? Result - PI : Result + PI

Else

Result = (pi / 2) - (Z / (Zˆ2 + 0.28))

if (y < 0.0f) Result = Result - PI;

⌋()

8.5.9 Average

[SWS_Mfl_00190] ⌈

Service Name	Mfl_Average_f32_f32	
Syntax	`float32 Mfl_Average_f32_f32 ( float32 value1, float32 value2 )`	
Service ID [hex]	0x61	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	value1	Input value1
	value2	Input value2
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return value of the function
Description	The routine returns average value.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00191] ⌈

Output = (Value1 + Value2) / 2

⌋()

8.5.10 Array Average

[SWS_Mfl_00192] ⌈

Service Name	Mfl_ArrayAverage_f32_f32	
Syntax	`float32 Mfl_ArrayAverage_f32_f32 ( const float32* Array, uint32 Count )`	
Service ID [hex]	0x65	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Array	Pointer to an array
	Count	Number of array elements
Parameters (inout)	None	

▽

△

Parameters (out)	None	
Return value	float32	Return value of the function
Description	The routine returns average value of an array.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00193] ⌈

Output = (Array[0] + Array[1]+_ _ Array[N-1]) / N

⌋()

8.5.11 Hypotenuse

[SWS_Mfl_00195] ⌈

Service Name	Mfl_Hypot_f32f32_f32	

Syntax	`float32 Mfl_Hypot_f32f32_f32 (`
	`  float32 x_value,`
	`  float32 y_value`
	`)`	

Service ID [hex]	0x70	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	First argument Recommended input range: [-24PI, +24PI]
	y_value	Second argument Recommended input range [-24PI, +24PI]
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Return value of the function
Description	This service computes the length of a vector	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00196] ⌈

This service computes the length of a vector:

Result = square_root (x_value * x_value + y_value * y_value)

⌋()

8.5.12 Ramp routines

In case of a change of the input value, the ramp output value follows the input value with a specified limited slope.

Mfl_ParamRamp_Type and Mfl_StateRamp_Type are the data types for storing ramp parameters. Usage of Switch-Routine and Jump-Routine is optional based on the functionality requirement. Usage of Switch-Routine, Jump-Routine, Calc-Routine and Out-Method have the following precondition concerning the sequence of the calls.

- Mfl_RampCalcSwitch
- Mfl_RampCalcJump
- Mfl_RampCalc
- Mfl_RampOut_f32

Structure definition for function argument

[SWS_Mfl_00200] ⌈

Name	Mfl_ParamRamp_Type	
Kind	Structure	
Elements	SlopePos_f32	
	Type	float32
	Comment	Positive slope for ramp in absolute value
	SlopeNeg_f32	
	Type	float32
	Comment	Negative slope for ramp in absolute value
Description	Structure definition for Ramp routine	
Available via	Mfl.h	

⌋ *()*

[SWS_Mfl_00833] ⌈

Name	Mfl_StateRamp_Type	
Kind	Structure	
Elements	State_f32	
	Type	float32
	Comment	State of the ramp
	Dir_s8	
	Type	sint8
	Comment	Ramp direction
	Switch_s8	
	Type	sint8
	Comment	Position of switch
Description	Structure definition for Ramp routine	
Available via	Mfl.h	

⌋ *()*

8.5.12.1 Ramp routine

[SWS_Mfl_00201] ⌈

Service Name	Mfl_RampCalc	
Syntax	`void Mfl_RampCalc ( float32 X_f32, Mfl_StateRamp_Type* State_cpst, const Mfl_ParamRamp_Type* Param_cpcst, float32 dT_f32 )`	
Service ID [hex]	0x90	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Target value for the ramp to reach
	Param_cpcst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to state structure
Parameters (out)	None	
Return value	None	
Description	The ramp output value increases or decreases a value with slope * dT_f32 depending if (State_cpst->State_f32 > X_f32) or (State_cpst->State_f32 < X_f32).	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00835] ⌈

If the ramp state State_cpst->State_f32 has reached or crossed the target value X_f32 while the direction of the ramp had been RISING/FALLING, then set State_cpst->State_f32 = X_f32.

⌋*()*

[SWS_Mfl_00202] ⌈

If ramp direction is rising then ramp increases a value with slope * dT_f32

if (State_cpst->Dir_s8 == RISING)

State_cpst->State_f32 = State_cpst->State_f32 + (Param_cpcst->SlopePos_f32 * dT_f32)

⌋*()*

[SWS_Mfl_00203] ⌈

If ramp direction is falling then ramp decreases a value with slope * dT_f32

if (State_cpst->Dir_s8 == FALLING)

State_cpst->State_f32 = State_cpst->State_f32 - (Param_cpcst->SlopeNeg_f32 * dT_f32)

⌋*()*

[SWS_Mfl_00204] ⌈

Direction of the ramp is stored so that a change of the target can be recognized and the output will follow immediately to the new target value.

State_cpst->Dir_s8 states are: RISING, FALLING, END.

⌋*()*

[SWS_Mfl_00205] ⌈

Comparison of State and Target decides ramp direction.

If(State_cpst->State_f32 > X_f32) then State_cpst->Dir_s8 = FALLING

If(State_cpst->State_f32 < X_f32) then State_cpst->Dir_s8 = RISING

If(State_cpst->State_f32 == X_f32) then State_cpst->Dir_s8 = END

⌋*()*

8.5.12.2 Ramp Initialisation

[SWS_Mfl_00208] ⌈

Service Name	Mfl_RampInitState	

Syntax	`void Mfl_RampInitState (`
	`  Mfl_StateRamp_Type* State_cpst,`
	`  float32 Val_f32`
	`)`	

Service ID [hex]	0x91	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Val_f32	Initial value for state variable
Parameters (inout)	State_cpst	Pointer to the state structure
Parameters (out)	None	
Return value	None	
Description	Initializes the state, direction and switch parameters for the ramp.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00209] ⌈

Ramp direction is initialised with END value. User has no possibility to change or modify ramp direction.

State_cpst->Dir_s8 = END

For example:

ramp direction states: RISING = 1, FALLING = -1, END = 0⌋*()*

[SWS_Mfl_00275] ⌈

Initialisation of state variable

State_cpst ->State_f32 = Val_f32

⌋()

[SWS_Mfl_00276] ⌈

Initialisation of switch variable. User has no possibility to change or modify switch initialization value.

State_cpst->Switch_s8 = OFF

For example:

switch states: TARGET_A = 1, TARGET_B = -1, OFF = 0⌋()

8.5.12.3 Ramp Set Slope

[SWS_Mfl_00210] ⌈

Service Name	Mfl_RampSetParam	
Syntax	`void Mfl_RampSetParam ( Mfl_ParamRamp_Type* Param_cpst, float32 SlopePosVal_f32, float32 SlopeNegVal_f32 )`	
Service ID [hex]	0x92	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	SlopePosVal_f32	Positive slope value
	SlopeNegVal_f32	Negative slope value
Parameters (inout)	None	
Parameters (out)	Param_cpst	Pointer to parameter structure
Return value	None	
Description	Sets the slope parameter for the ramp provided by the structure Mfl_ParamRamp_Type.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00211] ⌈

Sets positive and negative ramp slopes.

Param_cpst->SlopePos_f32 = SlopePosVal_f32

Param_cpst->SlopeNeg_f32 = SlopeNegVal_f32

⌋()

8.5.12.4 Ramp Out routine

[SWS_Mfl_00212] ⌈

Service Name	Mfl_RampOut_f32	
Syntax	`float32 Mfl_RampOut_f32 ( const Mfl_StateRamp_Type* State_cpcst )`	
Service ID [hex]	0x93	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpcst	Pointer to the state value
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Internal state of the ramp element
Description	Returns the internal state of the ramp element.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00213] ⌈

Return Value = State_cpcst->State_f32

⌋*()*

8.5.12.5 Ramp Jump routine

[SWS_Mfl_00214] ⌈

Service Name	Mfl_RampCalcJump	
Syntax	`void Mfl_RampCalcJump ( float32 X_f32, Mfl_StateRamp_Type* State_cpst )`	
Service ID [hex]	0x94	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_f32	Target value for ramp to jump
Parameters (inout)	State_cpst	Pointer to the state value
Parameters (out)	None	
Return value	None	
Description	This routine works in addition to main ramp function Mfl_RampCalc to provide a faster adaption to target value. If ramp is still rising (or falling) and target value is not reached, then input value of ramp jumps to a lower (or higher) value of current ramp state, ramp will jump to that value immediately. This functionality is helpful if input target value of ramp changes its direction often and significantly and ramp should reach target value faster than without that functionality. If the target is reached or the target does not change its direction, the standard behaviour of ramp functionality is untouched. In general, this routine decides whether a jump has to be done or not, if there is a change in the target. After a call to this function, Mfl_RampCalc function shall be called to execute the standard ramp behaviour.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00215] ⌈

If target value changes to a value contrary to current ramp direction and ramp has not reached its old target value then ramp state jumps to new target value immediately.

State_cpst->State_f32 = X_f32

State_cpst->Dir_s8 = END

Otherwise the previous values of State_cpst->Dir_s8 and State_cpst->State_f32 should be kept.

⌋*()*

8.5.12.6 Ramp switch routine

[SWS_Mfl_00216] ⌈

Service Name	Mfl_RampCalcSwitch_f32	

Syntax	`float32 Mfl_RampCalcSwitch_f32 (`
	`  float32 Xa_f32,`
	`  float32 Xb_f32,`
	`  Mfl_StateRamp_Type* State_cpst,`
	`  const Mfl_ParamRamp_Type* Param_cpcst,`
	`  float32 dT_f32`
	`)`	

Service ID [hex]	0x95	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Xa_f32	Target value for the ramp to reach if switch is in position 'A'
	Xb_f32	Target value for the ramp to reach if switch is in position 'B'
	Param_cpcst	Pointer to the parameter structure which contains the positive and negative slope of the ramp
	dT_f32	Sample Time
Parameters (inout)	State_cpst	Pointer to actual value of the ramp
Parameters (out)	None	
Return value	float32	Returns the actual state of the ramp
Description	This routine switches ramp between two target values based on the Switch value.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00217] ⌈

Switch decides target to select.

If (State_cpst->Switch_s8 == TARGET_A), target = Xa_f32

If (State_cpst->Switch_s8 == TARGET_B), target = Xb_f32

⌋*()*

[SWS_Mfl_00218] ⌈

State_cpst->Dir_s8 holds direction information

Ramp direction status: RISING, FALLING, END

⌋*()*

[SWS_Mfl_00219] ⌈

If ramp is active then ramp will change to reach selected target with defined slope.

if (State_cpst->Dir_s8 == RISING)

then State_cpst->State_f32 = State_cpst->State_f32 + (Param_cpcst->SlopePos_f32 * dT_f32)

else if (State_cpst->Dir_s8 == FALLING)

then State_cpst->State_f32 = State_cpst->State_f32 - (Param_cpcst->SlopeNeg_f32 * dT_f32)

else if (State_cpst->Dir_s8 == END)

State_cpst->State_f32 = target value which is decided by State_cpst->Switch_s8.

⌋*()*

[SWS_Mfl_00220] ⌈

Once ramp value reaches the selected target value, the ramp direction status is switched to END.

State_cpst->Dir_s8 == END

⌋*()*

[SWS_Mfl_00221] ⌈

If the ramp has reached its destination and no change of switch occurs, the output value follows the actual target value.

If(State_cpst->State_f32 == target value)

Return_value = Xa_f32 (if State_cpst->Switch_s8 is TARGET_A)

Return_value = Xb_f32 (if State_cpst->Switch_s8 is TARGET_B)

⌋*()*

[SWS_Mfl_00222] ⌈

Calculated ramp value shall be stored to State_cpst->State_f32 variable.

⌋*()*

Note : "This routine (Mfl_RampCalcSwitch_f32) is depreciated and will not be supported in future release.

Replacement routine : Mfl_RampCalcSwitch "

[SWS_Mfl_00369] ⌈

Service Name	Mfl_RampCalcSwitch	

Syntax	`float32 Mfl_RampCalcSwitch (`
	`float32 Xa_f32,`
	`float32 Xb_f32,`
	`boolean Switch,`
	`Mfl_StateRamp_Type* State_cpst`
	`)`	

Service ID [hex]	0xCA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Xa_f32	Target value for the ramp to reach if switch is in position ’A’

▽

△

	Xb_f32	Target value for the ramp to reach if switch is in position 'B'
	Switch	Switch to decide target value
Parameters (inout)	State_cpst	Pointer to StateRamp structure
Parameters (out)	None	
Return value	float32	Returns the selected target value
Description	This routine switches between two target values for a ramp service based on a Switch parameter.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00370] ⌈

Parameter Switch decides which target value is selected.

If Switch = TRUE, then Xa_f32 is selected.

State_cpst->Switch_s8 is set to TARGET_A

Return value = Xa_f32

If Swtich = FALSE, then Xb_f32 is selected.

State_cpst->Switch_s8 is set to TARGET_B

Return value = Xb_f32

⌋*()*

[SWS_Mfl_00371] ⌈

State_cpst->Dir_s8 hold direction information

State_cpst->Dir_s8 shall be set to END to reset direction information in case of target switch.

⌋*()*

[SWS_Mfl_00372] ⌈

Mfl_RampCalcSwitch has to be called before Mfl_RampCalc routine

⌋*()*

8.5.12.7 Get Ramp Switch position

[SWS_Mfl_00223] ⌈

Service Name	Mfl_RampGetSwitchPos	

Syntax	`boolean Mfl_RampGetSwitchPos (`
	`const Mfl_StateRamp_Type* State_cpst`
	`)`	

Service ID [hex]	0x96	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to the state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	return value TRUE or FALSE
Description	Gets the current switch position of ramp switch function.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00224] ⌈

Return value = TRUE if Switch position State_cpst->Switch_s8 = TARGET_A

Return value = FALSE if Switch position State_cpst->Switch_s8 = TARGET_B

⌋*()*

Note: The function "Mfl_RampGetSwitchPos" should be called only after calling the function "Mfl_RampCalcSwitch" or "Mfl_RampCalc".

8.5.12.8 Check Ramp Activity

[SWS_Mfl_00225] ⌈

Service Name	Mfl_RampCheckActivity	

Syntax	`boolean Mfl_RampCheckActivity (`
	`const Mfl_StateRamp_Type* State_cpst`
	`)`	

Service ID [hex]	0x97	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	State_cpst	Pointer to the state structure
Parameters (inout)	None	
Parameters (out)	None	
Return value	boolean	return value TRUE or FALSE
Description	This routine checks the status of the ramp and returns a TRUE if the ramp is active, otherwise it returns FALSE.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00226] ⌈

return value = TRUE, if Ramp is active (State_cpst->Dir_s8 != END)

return value = FALSE, if Ramp is inactive (State_cpst->Dir_s8 == END)

⌋*()*

8.5.13 Hysteresis routines

8.5.13.1 Hysteresis center half delta

[SWS_Mfl_00236] ⌈

Service Name	Mfl_HystCenterHalfDelta_f32_u8	
Syntax	`boolean Mfl_HystCenterHalfDelta_f32_u8 ( float32 X, float32 center, float32 halfDelta, uint8* State )`	
Service ID [hex]	0xA0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	center	Center of hysteresis range
	halfDelta	Half width of hysteresis range
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with center and left and right side halfDelta switching point.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00237] ⌈

Return value is TRUE if input is greater then center plus halfDelta switching point.

⌋*()*

[SWS_Mfl_00238] ⌈

Return value is FALSE if input is less then center minus halfDelta switching point.

⌋*()*

[SWS_Mfl_00239] ⌈

Return value is former state value if input is in the range of halfDelta around the center switching point

⌋*()*

8.5.13.2 Hysteresis left right

[SWS_Mfl_00241] ⌈

Service Name	Mfl_HystLeftRight_f32_u8	
Syntax	`boolean Mfl_HystLeftRight_f32_u8 ( float32 X, float32 Lsp, float32 Rsp, uint8* State )`	
Service ID [hex]	0xA3	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Lsp	Left switching point
	Rsp	Right switching point
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with left and right switching point.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00242] ⌈

Return value is TRUE if input is greater then right switching point.

⌋*()*

[SWS_Mfl_00243] ⌈

Return value is FALSE if input is less then left switching point.

⌋*()*

[SWS_Mfl_00244] ⌈

Return value is former state value if input is between left and right switching points

⌋*()*

8.5.13.3 Hysteresis delta right

[SWS_Mfl_00246] ⌈

Service Name	Mfl_HystDeltaRight_f32_u8	
Syntax	`boolean Mfl_HystDeltaRight_f32_u8 ( float32 X, float32 Delta, float32 Rsp, uint8* State )`	
Service ID [hex]	0xA5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Delta	Left switching point = rsp - delta
	Rsp	Right switching point
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with right switching point and delta to left switching point	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00247] ⌈

Return value is TRUE if input is greater then right switching point.

⌋*()*

[SWS_Mfl_00248] ⌈

Return value is FALSE if input is less then right switching point minus delta.

⌋*()*

[SWS_Mfl_00249] ⌈

Return value is former state value if input is between right switching points and right minus delta.

⌋*()*

8.5.13.4 Hysteresis left delta

[SWS_Mfl_00251] ⌈

Service Name	Mfl_HystLeftDelta_f32_u8	
Syntax	`boolean Mfl_HystLeftDelta_f32_u8 ( float32 X, float32 Lsp, float32 Delta, uint8* State )`	
Service ID [hex]	0xA7	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Lsp	Left switching point
	Delta	Right switching point = lsp + delta
Parameters (inout)	State	Pointer to state value
Parameters (out)	None	
Return value	boolean	Returns TRUE or FALSE depending of input value and state value
Description	Hysteresis with left switching point and delta to right switching point.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00252] ⌈

Return value is TRUE if input is greater then left switching point plus delta.

⌋*()*

[SWS_Mfl_00253] ⌈

Return value is FALSE if input is less then left switching point.

⌋*()*

[SWS_Mfl_00254] ⌈

Return value is former state value if input is between left switching points and left plus delta.

⌋*()*

8.5.14 Mfl_DeadTime

[SWS_Mfl_00256] ⌈

Service Name	Mfl_DeadTime_f32_f32	
Syntax	`float32 Mfl_DeadTime_f32_f32 ( float32 X, float32 DelayTime, float32 StepTime, Mfl_DeadTimeParam_Type* Param )`	
Service ID [hex]	0xAA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	DelayTime	Time to be delayed
	StepTime	Sample time
Parameters (inout)	Param	Pointer to parameter structure of type Mfl_DeadTimeParam_Type
Parameters (out)	None	
Return value	float32	Returns the actual state of the dead time element as sint16 value
Description	This routine returns input value with specified delay time.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00257] ⌈

Buffer data stores input samples hence reproduced output signal will reduce samples in case high delay time.

⌋*()*

[SWS_Mfl_00258] ⌈

Buffer size shall be configured as per the delay time range requirement.

⌋*()*

Structure definition for function argument

[SWS_Mfl_00259] ⌈

Name	Mfl_DeadTimeParam_Type	
Kind	Structure	
Elements	dsintStatic	
	Type	float32
	Comment	Time since the last pack was written
	*lszStatic	
	Type	float32
	Comment	Pointer to actual buffer position
	*dtbufBegStatic	

▽

△

	Type	float32
	Comment	Pointer to begin of buffer
	*dtbufEndStatic	
	Type	float32
	Comment	Pointer to end of buffer
Description	Structure definition for Dead Time routine	
Available via	Mfl.h	

⌋*()* "Note: This routine (Mfl_DeadTime_f32_f32) is depreciated and will not be supported in future release."

8.5.15 Debounce routines

8.5.15.1 Mfl_Debounce

[SWS_Mfl_00260] ⌈

Service Name	Mfl_Debounce_u8_u8	
Syntax	`boolean Mfl_Debounce_u8_u8 ( boolean X, Mfl_DebounceState_Type* State, const Mfl_DebounceParam_Type* Param, float32 dT )`	
Service ID [hex]	0xB0	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Input value
	Param	Pointer to state structure of type Mfl_DebounceState_Type
	dT	Sample Time
Parameters (inout)	State	Pointer to structure for debouncing state variables
Parameters (out)	None	
Return value	boolean	Returns the debounced input value
Description	This routine debounces a digital input signal and returns the state of the signal as a boolean value.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00261] ⌈

If(X != State->XOld) then check start debouncing.

⌋*()*

[SWS_Mfl_00262] ⌈

If transition is from Low to High, then use Param->TimeLowHigh as debouncing time otherwise use Param->TimeHighLow

⌋*()*

[SWS_Mfl_00263] ⌈

State->Timer is incremented with sample time for debouncing input signal.

Once reached to the set period, old state is updated with X.

State->Timer += dT;

If(State ->Timer $\geq$ TimePeriod)

State->XOld = X, and stop the timer, State->Timer = 0

where TimePeriod = Param->TimeLowHigh or Param->TimeHighLow

⌋*()*

[SWS_Mfl_00264] ⌈

Old value shall be returned as a output value. Current input is stored to old state.

Return value = State->XOld

State->XOld = X

⌋*()*

Structure definition for function argument

[SWS_Mfl_00265] ⌈

Name	Mfl_DebounceParam_Type	
Kind	Structure	
Elements	TimeHighLow	
	Type	float32
	Comment	Time for a High to Low transition, given in 10ms steps
	TimeLowHigh	
	Type	float32
	Comment	Time for a Low to High transition, given in 10ms steps
Description	Structure definition for Debouncing parameters	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00834] ⌈

Name	Mfl_DebounceState_Type
Kind	Structure
Elements	XOld

▽

△

	Type	boolean
	Comment	Old input value from last call
	Timer	
	Type	float32
	Comment	Timer for internal state
Description	Structure definition for Debouncing state variables	
Available via	Mfl.h	

⌋()

8.5.15.2 Mfl_DebounceInit

[SWS_Mfl_00266] ⌈

Service Name	Mfl_DebounceInit	
Syntax	`void Mfl_DebounceInit ( Mfl_DebounceState_Type* State, boolean X )`	
Service ID [hex]	0xB1	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X	Initial value for the input state
Parameters (inout)	None	
Parameters (out)	State	Pointer to structure for debouncing state variables
Return value	None	
Description	This routine call shall stop the debouncing timer.	
Available via	Mfl.h	

⌋()

[SWS_Mfl_00267] ⌈

State->Timer = 0

⌋()

[SWS_Mfl_00268] ⌈

Sets the input state to the given init value.

State->XOld = X

⌋()

8.5.15.3 Mfl_DebounceSetParam

[SWS_Mfl_00269] ⌈

Service Name	Mfl_DebounceSetparam	
Syntax	`void Mfl_DebounceSetparam ( Mfl_DebounceParam_Type* Param, float32 THighLow, float32 TLowHigh )`	
Service ID [hex]	0xB2	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (inout)	None	
Parameters (out)	Param	Pointer to state structure of type Mfl_DebounceParam_Type
Return value	None	
Description	This routine sets timing parameters, time for high to low transition and time for low to high for debouncing.	
Available via	Mfl.h	

⌋ *()*

[SWS_Mfl_00270] ⌈

Param-> TimeHighLow = THighLow

Param-> TimeLowHigh = TLowHigh

⌋ *()*

Note : "This routine (Mfl_DebounceSetparam) is depreciated and will not be supported in future release

Replacement routine : Mfl_DebounceSetParam "

[SWS_Mfl_00365] ⌈

Service Name	Mfl_DebounceSetParam	
Syntax	`void Mfl_DebounceSetParam ( Mfl_DebounceParam_Type* Param, float32 THighLow, float32 TLowHigh )`	
Service ID [hex]	0xC8	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (inout)	None	
Parameters (out)	Param	Pointer to state structure of type Mfl_DebounceParam_Type

▽

△

Return value	None
Description	This routine sets timing parameters, time for high to low transition and time for low to high for debouncing.
Available via	Mfl.h

⌋*()*

[SWS_Mfl_00366] ⌈

Param-> TimeHighLow = THighLow

Param-> TimeLowHigh = TLowHigh

⌋*()*

8.5.16 Ascending Sort Routine

[SWS_Mfl_00271] ⌈

Service Name	Mfl_SortAscend_f32	
Syntax	`void Mfl_SortAscend_f32 ( float32* Array, uint16 Num )`	
Service ID [hex]	0xB5	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Num	Size of an data array
Parameters (inout)	Array	Pointer to an data array
Parameters (out)	None	
Return value	None	
Description	The sorting algorithm modifies the given input array in ascending order & returns sorted array result via pointer	
Available via	Mfl.h	

⌋*()* Example for signed array:

Input array : float32 Array [5] = {-42.0, -10.0, 88.0, 8.0, 15.0};

Result : Array will be sorted to [-42.0, -10.0, 8.0, 15.0, 88.0]

8.5.17 Descending Sort Routine

[SWS_Mfl_00273] ⌈

Service Name	Mfl_SortDescend_f32	
Syntax	`void Mfl_SortDescend_f32 ( float32* Array, uint16 Num )`	
Service ID [hex]	0xBA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Num	Size of an data array
Parameters (inout)	Array	Pointer to an data array
Parameters (out)	None	
Return value	None	
Description	The sorting algorithm modifies the given input array in descending order & returns sorted array result via pointer	
Available via	Mfl.h	

⌋*()* Example for signed array:

Input array : float32 Array [5] = {-42.0, -10.0, 88.0, 8.0, 15.0};

Result : Array will be sorted to [88.0, 15.0, 8.0, -10.0, -42.0]

8.5.18 Median sort routine

[SWS_Mfl_00285] ⌈

Service Name	Mfl_MedianSort_f32_f32	
Syntax	`float32 Mfl_MedianSort_f32_f32 ( float32* Array, uint8 N )`	
Service ID [hex]	0xBB	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	N	Size of an array
Parameters (inout)	Array	Pointer to an array
Parameters (out)	None	
Return value	float32	Return value of the function
Description	This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call. If N is set incorrectly there is a security risk of buffer overrun. Memory corruption is possible, if parameter N is greater than the array size. Therefore N is recommended to be a constant, not a variable.	
Available via	Mfl.h	

⌋*()* For example:

Input array [5] = [42.0, 10.0, 88.0, 8.0, 15.0]

Sorted array[5] = [8.0, 10.0, 15.0, 42.0, 88.0]

[SWS_Mfl_00287] ⌈

Returns the median value of sorted array in case of N is even.

Result = (Sorted_array[N/2] + Sorted_array[(N/2) - 1]) / 2

For example:

Sorted_array[4] = [8.0, 10.0, 15.0, 42.0]

Result = (15.0 + 10.0) / 2.0 = 12.5⌋*()*

[SWS_Mfl_00288] ⌈

Returns the median value of sorted array in case of N is odd.

Return_Value = Sorted_array [N/2] = 15

For example:

Sorted_array[5] = [8.0, 10.0, 15.0, 42.0, 88.0]

Result = 15.0⌋*()*

[SWS_Mfl_00289] ⌈

In above calculation, N/2 shall be rounded off towards 0.

If N is set incorrectly there is a security risk of buffer overrun. Memory corruption is possible, if parameter N is greater than the array size. Therefore N is recommended to be a constant, not a variable.

⌋*()*

[SWS_Mfl_00836] ⌈

Service Name	Mfl_IntToFloatCvrt_<InTypeMn>_f32	

Syntax	`float32 Mfl_IntToFloatCvrt_<InTypeMn>_f32 (`
	`  <InType> ValInteger`
	`)`	

Service ID [hex]	0xD1 to 0xD6, 0xD9 to 0xDA	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValInteger	Integer value to be converted
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Returns the float value
Description	Returns the Float value for the corresponding Integer input.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00837] ⌈The result shall be round ties to even.⌋*()*

Function ID and prototypes

[SWS_Mfl_00838] ⌈

Function ID[hex]	Function prototype
0xD1	float32 Mfl_IntToFloatCvrt_u8_f32(uint8)
0xD2	float32 Mfl_IntToFloatCvrt_s8_f32(sint8)
0xD3	float32 Mfl_IntToFloatCvrt_u16_f32(uint16)
0xD4	float32 Mfl_IntToFloatCvrt_s16_f32(sint16)
0xD5	float32 Mfl_IntToFloatCvrt_u32_f32(uint32)
0xD6	float32 Mfl_IntToFloatCvrt_s32_f32(sint32)
0xD9	float32 Mfl_IntToFloatCvrt_u64_f32(uint64)
0xDA	float32 Mfl_IntToFloatCvrt_s64_f32(sint64)

⌋*()*

[SWS_Mfl_00839] ⌈

Service Name	Mfl	
Syntax	`<OutType> Mfl ( float32 ValFloat )`	
Service ID [hex]	0xCB to 0xD0, 0xD7 to 0xD8	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	ValFloat	Floating-point value to be converted
Parameters (inout)	None	
Parameters (out)	None	
Return value	<OutType>	Returns the integer value
Description	Returns the Integer value for the corresponding floating point input.	
Available via	Mfl.h	

⌋*()*

[SWS_Mfl_00840] ⌈

The return value shall be saturated to the return type boundary values in the event of overflow or underflow.

⌋*()*

[SWS_Mfl_00841] ⌈

The result shall be rounded toward zero.

⌋*()*

[SWS_Mfl_00843] ⌈

If the input is +/- Infinity, then the output shall be limited to target type boundaries (max/min) after the conversion.

⌋*()*

[SWS_Mfl_00848] ⌈

If the input is NaN, then the output of the conversion shall be zero.

⌋*()*

[SWS_Mfl_00842] ⌈

Function ID[hex]	Function prototype
0xCB	uint8 Mfl_FloatToIntCvrt_f32_u8(float32)
0xCC	sint8 Mfl_FloatToIntCvrt_f32_s8(float32)
0xCD	uint16 Mfl_FloatToIntCvrt_f32_u16(float32)
0xCE	sint16 Mfl_FloatToIntCvrt_f32_s16(float32)
0xCF	uint32 Mfl_FloatToIntCvrt_f32_u32(float32)
0xD0	sint32 Mfl_FloatToIntCvrt_f32_s32(float32)
0xD7	uint64 Mfl_FloatToIntCvrt_f32_u64(float32)
0xD8	sint64 Mfl_FloatToIntCvrt_f32_s64(float32)

⌋*()*

8.5.19 Modulus

[SWS_Mfl_00849] ⌈

Service name:	Mfl_Mod_f32	
Syntax:	Mfl_Mod_St_Type Mfl_Mod_f32(float32 x_f32, float32 y_f32, float32* Result)	
Service ID[hex]:	0xDB	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_f32	dividend
	y_f32	divisor
Parameters (inout):	None	
Parameters (out):	Result	Pointer to the Result
Return value:	Mfl_Mod_St_Type	Returns status of modulus operation E_SUCCESS: Mod operation success E_INVALID: Invalid Operation
Description:	This routine returns the value x_f32 - (n*y_f32), for some integer n such that, if y_f32 is nonzero, the result has the same sign as x_f32 and magnitude less than the magnitude of y_f32.	
Available via:	Mfl.h	

⌋*()*

[SWS_Mfl_00851] ⌈

Returns E_SUCCESS, in case of the following scenarios,

if the dividend and divisor is finite then,

*Result = x_f32 % y_f32 and the sign of result shall be same as sign of divdend.

- If the dividend is +/-0 and the divisor is finite number then the result shall be +/-0.

- If the dividend is finite number and divisor is +/-Infinity then the dividend shall be return as the result and the sign of result shall be same as that of the dividend.

⌋*()*

[SWS_Mfl_00852] ⌈

Returns E_INVALID, if there is an invalid operation and the result of the operation shall be NaN (not a number).

The operations considered as invalid in the following scenarios:

- If the divisor is zero

- If dividend is +/- infinity

- If dividend or divisor is NaN

- mod(0, 0) or mod(+/∞, +/∞)

⌋*()*

8.5.20 Division with limitation

[SWS_Mfl_00844] ⌈

Service name:	Mfl_DivLim_f32	
Syntax:	Mfl_Div_St Mfl_DivLim_f32(float32 x_f32, float32 y_f32, float32 min_f32, float32 max_f32, float32* Result)	
Service ID[hex]:	0xDC	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_f32	dividend
	y_f32	divisor
	min_f32	minimum limit, min_f32 shall not be strictly greater than max_f32
	max_f32	maximum limit, max_f32 shall not be strictly lower than min_f32
Parameters (inout):	None	
Parameters (out):	Result	Pointer to the result
Return value:	Mfl_Div_St	Returns status of division E_SUCCESS: Division operation successful E_DIVBYZERO: Divide by Zero E_INVALID: Invalid operation
Description:	Divides x_f32 by y_f32 and limits the result within the min_f32 and max_f32 value.	
Available via:	Mfl.h	

⌋ *()*

[SWS_Mfl_00845] ⌈

Returns E_SUCCESS, in case of the following scenarios,if the dividend and divisor are finite then,Result = X_f32 / Y_f32 and the sign of result is the exclusive OR of the operands' signs.

IF(*Result > max_f32)

*Result = max_f32

ELSE IF(*Result < min_f32)

*Result = min_f32

If the dividend is +/-Infinity and the divisor is finite number then the result shall be saturated to max/min based on the sign of the result which is the exclusive OR of the operands' signs.

If the divisor is Infinity and dividend is finite number then the result shall be zero and the sign of result is the exclusive OR of the operands' signs.

⌋ *()*

[SWS_Mfl_00846] ⌈

Returns E_INVALID, if there is an invalid operation and the result of the IEEE754 division operation is NaN (not a number).

According to IEEE 754 the results of 0/0 and (+/- ∞)/(+/- ∞) are invalid. In these cases the function result shall remain at its initial value.

⌋*()*

[SWS_Mfl_00847] ⌈

Returns E_DIVBYZERO, if the divisor is zero and the dividend is a finite non-zero number, the result shall be max/min based on the sign of the result which is the exclusive OR of the operands' signs.

⌋*()*

8.5.21 Bit pattern

[SWS_Mfl_91000] ⌈

Service Name	Mfl_Bitpat_f32_u32	
Syntax	`uint32 Mfl_Bitpat_f32_u32 ( float32 Value )`	
Service ID [hex]	0x101	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Value	Floating-point value to be represented
Parameters (inout)	None	
Parameters (out)	None	
Return value	uint32	Bit representation of the single precision floating point value
Description	Return the bit representation of the single point precision float value	
Available via	Mfl.h	

⌋*()*

8.5.22 Fast inverse square root

[SWS_Mfl_91001] ⌈

Service Name	Mfl_RSqrt_f32
Syntax	`float32 Mfl_RSqrt_f32 ( float32 ValValue )`
Service ID [hex]	0x104
Sync/Async	Synchronous
Reentrancy	Reentrant

▽

△

Parameters (in)	ValValue	Floating-point operand.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Reciprocal square root of ValValue
Description	Fast reciprocal square root of the operand (ValSqrt)	
Available via	Mfl.h	

⌋(*SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011*)

[SWS_Mfl_00854] ⌈

Calculate an approximation of the reciprocal of the square root of ValValue (1/sqrt(ValValue)). The accuracy of the result is no less than 6.75 bits, and therefore always within +/- 1 % of the accurate result.

This instruction can be used to implement a floating-point square root function in software using the Newton-Raphson iterative method.⌋(*SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011*)

8.5.23 Trigonometric routines

[SWS_Mfl_91003] ⌈

Service Name	Mfl_Cos_s32_f32	
Syntax	`float32 Mfl_Cos_s32_f32 ( sint32 x_value )`	
Service ID [hex]	0x102	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Physical range: [-PI, PI[Resolution: $2^*PI/(2^{32})$
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	result = cosine (x_value), range: [-1.0 ... 1.0]
Description	Fast cosine calculation	
Available via	Mfl.h	

⌋(*SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011*)

Algorithm hint (no requirement) for x_value [-PI/2...PI/2]:

Mfl_Cos_s32_f32 (x_value) = c1 + c2 * x_valueˆ 2 + c3 * x_valueˆ 4

which is the same as:

Mfl_Cos_s32_f32 (x_value) = c1 + x_valueˆ 2 * (c2 + c3 * x_valueˆ 2)

[SWS_Mfl_91004] ⌈

Service Name	Mfl_Sin_s32_f32	
Syntax	`float32 Mfl_Sin_s32_f32 ( sint32 x_value )`	
Service ID [hex]	0x103	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	x_value	Physical range: [-PI, PI[Resolution: 2*PI/(2^{32})
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	result = sine (x_value), range: [-1.0 ... 1.0]
Description	Fast sine calculation	
Available via	Mfl.h	

⌋*(SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011)*

Algorithm hint (no requirement):

Mfl_Sin_s32_f32(x_value) = Mfl_Cos_s32_f32(1073741824 (physical: pi/2) - x_value)

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Mfl_GetVersionInfo

[SWS_Mfl_00815] ⌈

Service Name	Mfl_GetVersionInfo	
Syntax	`void Mfl_GetVersionInfo ( Std_VersionInfoType* versioninfo )`	
Service ID [hex]	0xff	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	None	
Parameters (out)	versioninfo	Pointer to where to store the version information of this module. Format according [BSW00321]
Return value	None	
Description	Returns the version information of this library.	
Available via	Mfl.h	

⌋*(SRS_BSW_00407, SRS_BSW_00003, SRS_BSW_00318, SRS_BSW_00321)*

The version information of a BSW module generally contains:

- Module Id
- Vendor Id
- Vendor specific version numbers (SRS_BSW_00407).

[SWS_Mfl_00816] ⌈If source code for caller and callee of Mfl_GetVersionInfo is available, the Mfl library should realize Mfl_GetVersionInfo as a macro defined in the module's header file.⌋*(SRS_BSW_00407, SRS_BSW_00411)*

8.8 Callback notifications

None

8.9 Scheduled functions

The Mfl library does not have scheduled functions.

8.10 Expected interfaces

None

8.10.1 Mandatory interfaces

None

8.10.2 Optional interfaces

None

8.10.3 Configurable interfaces

None

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published Information

[SWS_Mfl_00814] ⌈The standardized common published parameters as required by [SRS_BSW_00402] in the General Requirements on Basic Software Modules [REF] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [REF].⌋*(SRS_BSW_00402, SRS_BSW_00374, SRS_-BSW_00379)*

Additional module-specific published parameters are listed below if applicable.

10.2 Configuration option

[SWS_Mfl_00818] ⌈The Mfl library shall not have any configuration options that may affect the functional behavior of the routines. I.e. for a given set of input parameters, the outputs shall be always the same. For example, the returned value in case of error shall not be configurable.⌋*(SRS_LIBS_00001)*

However, a library vendor is allowed to add specific configuration options concerning library implementation, e.g. for resources consumption optimization.

A Not applicable requirements

[SWS_Mfl_00822] ⌈These requirements are not applicable to this specification.⌋ *(SRS_BSW_00448)*