

<b>Document Title</b>	Specification of Floating Point Math Routines
<b>Document Owner</b>	AUTOSAR
<b>Document Responsibility</b>	AUTOSAR
<b>Document Identification No</b>	397

<b>Document Status</b>	published
<b>Part of AUTOSAR Standard</b>	Classic Platform
<b>Part of Standard Release</b>	R23-11

<b>Document Change History</b>			
<b>Date</b>	<b>Release</b>	<b>Changed by</b>	<b>Description</b>
2023-11-23	R23-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Updated SWS_Mfl_00305.</li> </ul>
2022-11-24	R22-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Updated SWS_Mfl_00285 and SWS_Mfl_00289.</li> <li>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.</li> </ul>
2021-11-25	R21-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>New requirements SWS_Mfl_00843 and SWS_Mfl_00848 added for "Mfl_FloatToIntCvrt_f32" function.</li> <li>Editorial changes.</li> </ul>
2020-11-30	R20-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>New function added Mfl_DivLim_f32 with the requirements SWS_Mfl_00844, SWS_Mfl_00845, SWS_Mfl_00846 and SWS_Mfl_00847.</li> <li>Updated Chapter 7.1 Error classification.</li> </ul>
2019-11-28	R19-11	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Editorial changes</li> <li>Changed Document Status from Final to published</li> </ul>





2019-07-29	4.4.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• New function description for Mfl_Mod_f32 function</li> <li>• Added new 64 bit variants in conversion function - SWS_Mfl_00836, SWS_Mfl_00839, SWS_Mfl_00838, SWS_Mfl_00842</li> </ul>
2018-10-31	4.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Added description for Mfl_Pow_f32 function</li> <li>• Updated name of parameter dT_f32 in the requirements SWS_Mfl_00045, SWS_Mfl_00047, SWS_Mfl_00301 &amp; SWS_Mfl_00303</li> </ul>
2017-12-08	4.3.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• A note has been added in Section 8.1 of MFL specification to provide clarity in usage of mnemonic for Boolean data types.</li> <li>• Inclusion of Pointer to Constant (P2CONST) for SWS_Mfl_00260, SWS_Mfl_00246, SWS_Mfl_00225 &amp; SWS_Mfl_00223 and proper categorization of Parameters as Out/InOut for SWS_Mfl_00266, SWS_Mfl_00285 &amp; SWS_Mfl_00037.</li> </ul>
2016-11-30	4.3.0	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>• Section 2 has been revisited to update Default Error Tracer instead of Development Error tracer.</li> <li>• SWS_Mfl_00362 has been updated to provide clarity in requirements.</li> <li>• SWS_Mfl_00363 has been modified to provide clear requirements.</li> <li>• Updated the parameters in SWS_Mfl_00360 for Mfl_ArcTan2_f32 service to be in sync with standard C library.</li> <li>• Updated SWS_Mfl_00122 to provide better clarity on the input parameter limits.</li> <li>• Verified that the spec SWS_Mfl_00122 has been updated to provide better clarity on input parameter limits.</li> </ul>





			<p style="text-align: center;">△</p> <ul style="list-style-type: none"> <li>• Updated MFL document to support MISRA 2012 standard. (Removed Reference related to MISRA 2004 from chapter 3.2 and redundant statements in SWS_Mfl_00809 which already exist in SWS_BSW document and SWS_SRS document)</li> <li>• Modified the reference to SRS_BSW_General (SRS_BSW_00437) &amp; (SRS_BSW_00448) for SWS_Mfl_00810 &amp; SWS_Mfl_00822 requirements.</li> </ul>
<p>2015-07-31</p>	<p>4.2.2</p>	<p>AUTOSAR Release Management</p>	<ul style="list-style-type: none"> <li>• BSWUML Model for "Mfl_HystCenterHalfDelta_f32_u8", "Mfl_HystLeftRight_f32_u8", "Mfl_HystDeltaRight_f32_u8" &amp; "Mfl_HystLeftDelta_f32_u8" functions were updated in the Word Document.</li> <li>• Statement has been updated for Mfl_DT1Typ1Calc and Mfl_DT1Typ2Calc to clearly mention the data type for the Time Equivalent parameter.</li> <li>• Description field has been updated/rectified for Tv_C and Tnrec_C parameters in Mfl_ParamPID_Type.</li> <li>• Updated naming convention for TeQ_f32 Parameter.</li> <li>• Corrected the description for TeQ_&lt;Size&gt; in section 8.5.4.1 and statement in section 8.5.4.4.</li> <li>• Naming convention followed for Tnrec Parameter in Mfl_PISetParam function.</li> <li>• Statement has been updated to correct naming convention for TeQ_f32.</li> <li>• Updated SWS_Mfl_00001 for naming convention under Section 5.1, File Structure</li> <li>• BSWUML Model for "Mfl_ArrayAverage_f32_f32" function was updated to include</li> </ul> <p style="text-align: center;">▽</p>





			<p style="text-align: center;">△</p> <p>pointer to constant to avoid MISRA violation/warning. (SWS_Mfl_00192)</p> <ul style="list-style-type: none"> <li>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</li> <li>Removed the requirements SWS_Mfl_00240, SWS_Mfl_00245, SWS_Mfl_00250 &amp; SWS_Mfl_00255</li> <li>Removed redundant requirements SWS_Mfl_00034, SWS_Mfl_00046 &amp; SWS_Mfl_00302, which were covered as part of section 8.5.4.4.</li> </ul>
2014-10-31	4.2.1	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>New Functions are added to convert values between Float and Integer. (SWS_Mfl_00837, SWS_Mfl_838, SWS_Mfl_840, SWS_Mfl_841 &amp; SWS_Mfl_842)</li> <li>BSWUML Model was updated for "Mfl_FloatToIntCvrt_f32" &amp; "Mfl_IntToFloatCvrt" functions. (SWS_Mfl_00836 &amp; SWS_Mfl_839)</li> <li>Updated usage of const in a consistent manner.</li> </ul>
2014-03-31	4.1.3	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Removed SWS_Mfl_00206, SWS_Mfl_00207 and SWS_Mfl_00281 from Mfl_RampCalc &amp; Mfl_RampCalcJump functions.</li> </ul>
2013-10-31	4.1.2	AUTOSAR Release Management	<ul style="list-style-type: none"> <li>Deprecated: Mfl_DeadTime function</li> <li>Removed: SWS_Mfl_00197 from Mfl_Hypot function</li> <li>Added: SWS_Mfl_00835 for Mfl_RampCalc function, a note for Mfl_RampGetSwitchPos function</li> <li>Modified: Description for Mfl_RampSetParam function, Parameter (in) definition for Mfl_RateLimiter_f32</li> <li>Editorial changes</li> </ul>



△

2013-03-15	4.1.1	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Description and requirements are modified for Mfl_RampCalcJump, Mfl_RampCalc</li> <li>• Formatting error in superscripts are corrected</li> <li>• Corrected "DT1" to "I" in I-Controller functions</li> <li>• Description of the parameter "State" is corrected in Mfl_Debounce and Mfl_DebounceInit functions</li> <li>• Corrected for 'DependencyOnArtifact'</li> </ul>
2011-12-22	4.0.3	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Removal of 'Accumulator routine'</li> <li>• Revised 'Trigonometric routines' names</li> <li>• Added 'Median Sort Routines'</li> </ul>
2010-09-30	3.1.5	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Introduction of additional LIMITED Functions for controllers</li> <li>• Ramp functions optimised for effective usage</li> <li>• Separation of DT1 Type 1 and Type 2 Controller functions</li> <li>• Introduction of additional approximative function for calculatio of TeQ</li> </ul>
2010-02-02	3.1.4	AUTOSAR Administration	<ul style="list-style-type: none"> <li>• Initial Release</li> </ul>

## Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

## Contents

1	Introduction and functional overview	10
2	Acronyms and Abbreviations	12
3	Related documentation	13
3.1	Input documents & related standards and norms	13
3.2	Related specification	13
4	Constraints and assumptions	14
4.1	Limitations	14
4.2	Applicability to car domains	14
5	Dependencies to other modules	15
5.1	File structure	15
6	Requirements Tracing	16
7	Functional specification	18
7.1	Error Classification	18
7.1.1	Development Errors	18
7.1.2	Runtime Errors	18
7.1.3	Transient Faults	18
7.1.4	Production Errors	18
7.1.5	Extended Production Errors	18
7.2	Error detection	18
7.3	Error notification	19
7.4	Initialization and shutdown	19
7.5	Using Library API	19
7.6	Library implementation	20
8	Routine specification	22
8.1	Imported types	22
8.2	Type definitions	22
8.3	Comment about rounding	23
8.4	Comment about routines optimized for target	23
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

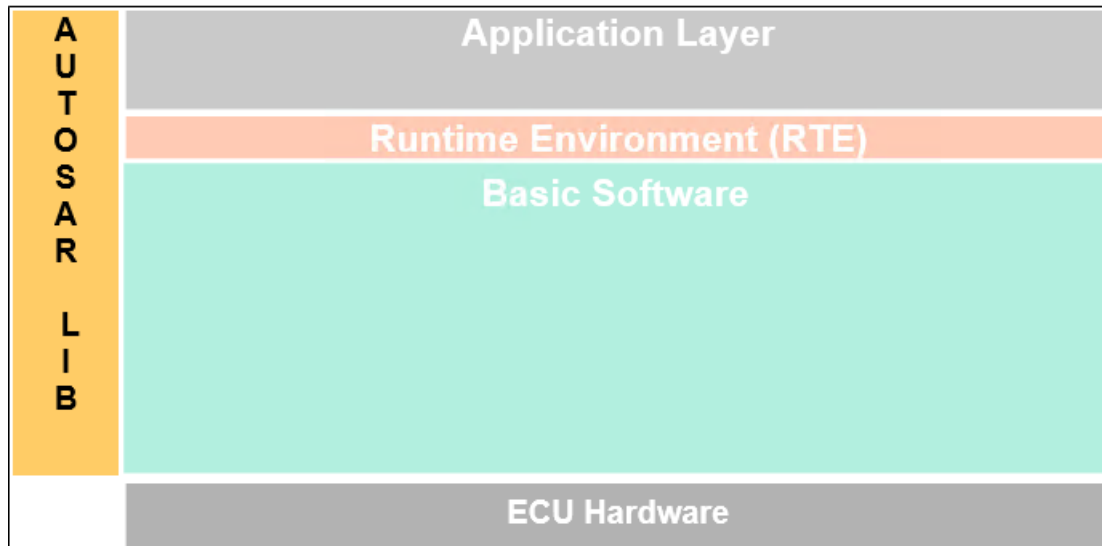
8.5.4.6	Integral component . . . . .	47
8.5.4.7	Proportional & Integral controller . . . . .	51
8.5.4.8	Proportional, Integral & Differential controller . . . . .	58
8.5.5	Magnitude and Sign . . . . .	66
8.5.6	Limiting . . . . .	67
8.5.7	Logarithms and Exponentials . . . . .	70
8.5.8	Trigonometry . . . . .	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	Bit pattern . . . . .	109
8.5.20	Fast inverse square root . . . . .	109
8.5.21	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
B History of Specification Items	116
B.1 Specification Item History of this document compared to AUTOSAR R22-11.	116
B.1.1 Added Specification Items in R23-11	116
B.1.2 Changed Specification Items in R23-11	116
B.1.3 Deleted Specification Items in R23-11	118

# 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  
<https://www.iso.org>

[2] General Specification of Basic Software Modules  
AUTOSAR\_CP\_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_Mfi_00815]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Mfi_00809]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Mfi_00812]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Mfi_00813]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_Mfi_00815]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Mfi_00815]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Mfi_00811]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Mfi_00814]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Mfi_00812]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Mfi_00814]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_Mfi_00814]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Mfi_00815] [SWS_Mfi_00816]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Mfi_00816]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_Mfi_00810]
[SRS_BSW_00448]	Module SWS shall not contain requirements from other modules	[SWS_Mfi_00822]
[SRS_LIBS_00001]	The functional behavior of each library functions shall not be configurable	[SWS_Mfi_00818]
[SRS_LIBS_00002]	A library shall be operational before all BSW modules and application SW-Cs	[SWS_Mfi_00800]
[SRS_LIBS_00003]	A library shall be operational until the shutdown	[SWS_Mfi_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: Requirements Tracing**

## 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:<sup>[10]</sup>(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.

**[SWS\_Mfl\_91002] Definition of datatype Mfl\_Mod\_St\_Type [**

<b>Name</b>	Mfl_Mod_St_Type		
<b>Kind</b>	Enumeration		
<b>Range</b>	E_SUCCESS	–	Mod operation success
	E_INVALID	–	Invalid Operation
<b>Description</b>	Specifies the status of the Mod operation		
<b>Available via</b>	Mfl.h		

]

**[SWS\_Mfl\_91005] Definition of datatype Mfl\_Div\_st [**

<b>Name</b>	Mfl_Div_st		
<b>Kind</b>	Enumeration		
<b>Range</b>	E_SUCCESS	–	Division success
	E_DIVBYZERO	–	Divide by Zero
	E_INVALID	–	Invalid Operation
<b>Description</b>	–		
<b>Available via</b>	Mfl_Types.h		

]

### 8.3 Comment about rounding

Two types of rounding can be applied:

Results are 'rounded off', it means:

- $0 \leq X < 0.5$  rounded to 0
- $0.5 \leq X < 1$  rounded to 1
- $-0.5 < X \leq 0$  rounded to 0
- $-1 < X \leq -0.5$  rounded to -1

Results are rounded towards zero.

- $0 \leq X < 1$  rounded to 0
- $-1 < X \leq 0$  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.



## 8.5 Routine definitions

### 8.5.1 Floating point to Fixed-Point Conversion

#### [SWS\_Mfl\_00005] Definition of API function Mfl\_Cvrt\_f32\_<OutTypeMn> [

<b>Service Name</b>	Mfl_Cvrt_f32_<OutTypeMn>	
<b>Syntax</b>	<OutType> Mfl_Cvrt_f32_<OutTypeMn> ( float32 ValFloat, sint16 ValFixedExponent )	
<b>Service ID [hex]</b>	0x01 to 0x04	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValFloat	Floating-point quantity to be converted.
	ValFixedExponent	Exponent of the fixed-point result of the conversion.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	<OutType>	Returns the integer value of the fixed-point result
<b>Description</b>	Returns the integer value of the fixed point result of the conversion, determined according to the following equation.	
<b>Available via</b>	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] Definition of API function Mfl\_Cvrt\_<InTypeMn>\_f32 ⌈

<b>Service Name</b>	Mfl_Cvrt_<InTypeMn>_f32	
<b>Syntax</b>	float32 Mfl_Cvrt_<InTypeMn>_f32 ( <InType> ValFixedInteger, sint16 ValFixedExponent )	
<b>Service ID [hex]</b>	0x05 to 0x08	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValFixedInteger	Integer value of the fixed-point quantity to be converted
	ValFixedExponent	Exponent of the fixed-point quantity to be converted.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	The floating-point result of the conversion.
<b>Description</b>	Returns the floating-point result of the conversion, determined according to the following equation.	
<b>Available via</b>	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] Definition of API function Mfl\_Trunc\_f32 [

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

]() For example:

36.56 will be truncated to 36.00

#### [SWS\_Mfl\_00015] Definition of API function Mfl\_Round\_f32 [

<b>Service Name</b>	Mfl_Round_f32	
<b>Syntax</b>	float32 Mfl_Round_f32 ( float32 ValValue )	
<b>Service ID [hex]</b>	0x0A	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValValue	Floating-point operand.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Rounded value of operand.
<b>Description</b>	Returns the integer value determined by rounding the argument toward the nearest whole number.	
<b>Available via</b>	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] Definition of API function Mfl\_Ceil\_f32 [**

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

]|()

**[SWS\_Mfl\_00020] Definition of API function Mfl\_Floor\_f32 [**

<b>Service Name</b>	Mfl_Floor_f32	
<b>Syntax</b>	float32 Mfl_Floor_f32 ( float32 ValValue )	
<b>Service ID [hex]</b>	0x0C	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValValue	Floating-point operand.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Operand rounded to floor.
<b>Description</b>	Returns the natural number value determined by rounding the argument toward negative infinity.	
<b>Available via</b>	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  $y_n$  and can be represented in the following equation.

$$Y_n = a_1 * Y_{n-1} + b_0 * X_n + b_1 * X_{n-1} + b_2 * X_{n-2} + \dots + b_{n-1} * X_1 + b_n * X_0$$

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] Definition of datatype Mfl\_StatePT1\_Type [**

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

]()

**[SWS\_Mfl\_00823] Definition of datatype Mfl\_StateDT1Typ1\_Type [**

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

]()

**[SWS\_Mfl\_00824] Definition of datatype Mfl\_StateDT1Typ2\_Type [**

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

]()

**[SWS\_Mfl\_00825] Definition of datatype Mfl\_StatePD\_Type [**

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

]()

**[SWS\_Mfl\_00826] Definition of datatype Mfl\_ParamPD\_Type [**

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

]()

**[SWS\_Mfl\_00827] Definition of datatype Mfl\_StateI\_Type [**

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

]()

[SWS\_Mfl\_00828] Definition of datatype Mfl\_StatePI\_Type [

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

]()

[SWS\_Mfl\_00829] Definition of datatype Mfl\_ParamPI\_Type [

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

]()

[SWS\_Mfl\_00830] Definition of datatype Mfl\_StatePID\_Type [

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

]()



**[SWS\_Mfl\_00831] Definition of datatype Mfl\_ParamPID\_Type [**

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

]()

**[SWS\_Mfl\_00832] Definition of datatype Mfl\_Limits\_Type [**

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

]()

### 8.5.4.2 Proportional Controller

Proportional component calculates  $Y(x) = K_p * X$ .

1. 'P' Controller

#### [SWS\_Mfl\_00026] Definition of API function Mfl\_PCalc [

<b>Service Name</b>	Mfl_PCalc	
<b>Syntax</b>	<pre>void Mfl_PCalc (     float32 X_f32,     float32* P_pf32,     float32 K_f32 )</pre>	
<b>Service ID [hex]</b>	0x10	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	input value
	K_f32	Amplification factor
<b>Parameters (inout)</b>	P_pf32	Pointer to the calculated state
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Differential equation: $Y = K * X$	
<b>Available via</b>	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] Definition of API function Mfl\_POut\_f32 [

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



△

<b>Description</b>	This routine returns 'P' controllers output value limited by the return data type
<b>Available via</b>	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] Definition of API function Mfl\_PT1Calc** [

<b>Service Name</b>	Mfl_PT1Calc	
<b>Syntax</b>	<pre>void Mfl_PT1Calc (     float32 X_f32,     Mfl_StatePT1_Type* State_cpst,     float32 K_f32,     float32 TeQ_f32 )</pre>	
<b>Service ID [hex]</b>	0x1A	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	Input value for the PT1 element
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
<b>Parameters (inout)</b>	State_cpst	Pointer to PT1 state structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	This routine computes PT1 controller output value using below difference equation	
<b>Available via</b>	Mfl.h	

}]()

**[SWS\_Mfl\_00033]** [

$$Y_n = \exp(-dT/T_1) * Y_{n-1} + K(1 - \exp(-dT/T_1)) * X_n - 1$$

This derives implementation:

$$\text{Output\_value} = (\text{TeQ\_f32} * \text{State\_cpst} \rightarrow Y_1) + K\_f32 * (1 - \text{TeQ\_f32}) * \text{State\_cpst} \rightarrow X_1$$

where  $TeQ\_f32 = \exp(-dT/T1)$

}]()

**[SWS\_Mfl\_00035]** [

If ( $TeQ\_f32 = 0$ ) then PT1 controller follows Input value,

$State\_cpst \rightarrow Y1 = K\_f32 * X\_f32$

}]()

**[SWS\_Mfl\_00036]** [

calculated Output\_value and current input value shall be stored to  $State\_cpst \rightarrow Y1$  and  $State\_cpst \rightarrow X1$  respectively.

$State\_cpst \rightarrow Y1 = Output\_value$

$State\_cpst \rightarrow X1 = X\_f32$

1. 'PT1' Set State Value

This routine can be realised using inline function.

}]()

**[SWS\_Mfl\_00037] Definition of API function Mfl\_PT1SetState** [

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

}]()

**[SWS\_Mfl\_00038]** [

Initialisation of output state variable Y1.

$State\_cpst \rightarrow Y1 = Y1\_f32$

}]()

**[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] Definition of API function Mfl\_CalcTeQ\_f32 [**

<b>Service Name</b>	Mfl_CalcTeQ_f32	
<b>Syntax</b>	float32 Mfl_CalcTeQ_f32 ( float32 T1rec_f32, float32 dT_f32 )	
<b>Service ID [hex]</b>	0x1C	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	T1rec_f32	Reciprocal delay time
	dT_f32	Sample Time
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Time Equivalent TeQ_f32
<b>Description</b>	This routine calculates time equivalent factor	
<b>Available via</b>	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] Definition of API function Mfl\_CalcTeQApp\_f32 [**

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



△

<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Time Equivalent TeQApp_f32
<b>Description</b>	This routine calculates time equivalent factor	
<b>Available via</b>	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] Definition of API function Mfl\_PT1Out\_f32 [

<b>Service Name</b>	Mfl_PT1Out_f32	
<b>Syntax</b>	float32 Mfl_PT1Out_f32 ( const Mfl_StatePT1_Type* State_cpst )	
<b>Service ID [hex]</b>	0x1D	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'PT1' controller output value
<b>Description</b>	This routine returns 'PT1' controllers output value	
<b>Available via</b>	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] Definition of API function Mfl\_DT1Typ1Calc** [

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

]()

**[SWS\_Mfl\_00045]** [

$$Y_n = \exp(-dT/T1) * Y_{n-1} + K * (1 - \exp(-dT/T1)) * ((X_{n-1} - X_{n-2}) / dT)$$

This derives implementation:

$$\text{Output\_value} = (\text{TeQ\_f32} * \text{State\_cpst} \rightarrow Y1) + K\_f32 * (1 - \text{TeQ\_f32}) * ((\text{State\_cpst} \rightarrow X1 - \text{State\_cpst} \rightarrow X2) / dT\_f32)$$

where  $\text{TeQ\_f32} = \exp(-dT\_f32/T1)$

]()

**[SWS\_Mfl\_00047]** [

If (TeQ\_f32 = 0) then DT1 controller follows Input value,

$$\text{Output\_value} = K\_f32 * (X\_f32 - \text{State\_cpst} \rightarrow X1) / dT\_f32$$

]()

**[SWS\_Mfl\_00048]** [

Calculated Output\_value shall be stored to State\_cpst->Y1.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

]()

**[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] Definition of API function Mfl\_DT1Typ2Calc [**

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

}]()

**[SWS\_Mfl\_00301] [**

$$Y_n = \exp(-dT/T1) * Y_{n-1} + K * (1 - \exp(-dT/T1)) * ((X_n - X_{n-1}) / dT)$$

This derives implementation:

$$\text{Output\_value} = (\text{TeQ\_f32} * \text{State\_cpst->Y1}) + \text{K\_f32} * (1 - \text{TeQ\_f32}) * ((\text{X\_f32} - \text{State\_cpst->X1}) / \text{dT\_f32})$$

where  $\text{TeQ\_f32} = \exp(-dT\_f32/T1)$

}]()

**[SWS\_Mfl\_00303] [**

If (TeQ\_f32 = 0) then DT1 controller follows Input value,

$$\text{Output\_value} = \text{K\_f32} * (\text{X\_f32} - \text{State\_cpst->X1}) / \text{dT\_f32}$$

}]()

**[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\_f32

1. Set 'DT1' State Value - Type1

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

**[SWS\_Mfl\_00050] Definition of API function Mfl\_DT1Typ1SetState** [

<b>Service Name</b>	Mfl_DT1Typ1SetState	
<b>Syntax</b>	<pre>void Mfl_DT1Typ1SetState (     Mfl_StateDT1Typ1_Type* State_cpst,     float32 X1_f32,     float32 X2_f32,     float32 Y1_f32 )</pre>	
<b>Service ID [hex]</b>	0x22	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	State_cpst	Pointer to internal state structure
<b>Return value</b>	None	
<b>Description</b>	The routine initialises internal state variables of a DT1 element.	
<b>Available via</b>	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] Definition of API function Mfl\_DT1Typ2SetState [**

<b>Service Name</b>	Mfl_DT1Typ2SetState	
<b>Syntax</b>	<pre>void Mfl_DT1Typ2SetState (     Mfl_StateDT1Typ2_Type* State_cpst,     float32 X1_f32,     float32 Y1_f32 )</pre>	
<b>Service ID [hex]</b>	0xC1	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X1_f32	Initial value for the input state
	Y1_f32	Initial value for the output state
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	State_cpst	Pointer to internal state structure
<b>Return value</b>	None	
<b>Description</b>	The routine initialises internal state variables of a DT1 element.	
<b>Available via</b>	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] Definition of API function Mfl\_DT1Typ1Out\_f32** [

<b>Service Name</b>	Mfl_DT1Typ1Out_f32	
<b>Syntax</b>	<pre>float32 Mfl_DT1Typ1Out_f32 (     const Mfl_StateDT1Typ1_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0x23	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'DT1' controller output value
<b>Description</b>	This routine returns 'DT1' controller's output value	
<b>Available via</b>	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] Definition of API function Mfl\_DT1Typ2Out\_f32** [

<b>Service Name</b>	Mfl_DT1Typ2Out_f32	
<b>Syntax</b>	<pre>float32 Mfl_DT1Typ2Out_f32 (     const Mfl_StateDT1Typ2_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0xC2	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'DT1' controller output value
<b>Description</b>	This routine returns 'DT1' controller's output value	
<b>Available via</b>	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] Definition of API function Mfl\_PDCalc [

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

}]()

#### [SWS\_Mfl\_00056] [

$$Y_n = K(1 + T_v/dT) * X_n - K(T_v/dT) * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = (\text{Param\_cpst} \rightarrow K\_C * (1 + \text{Param\_cpst} \rightarrow T_v\_C/dT\_f32) * X\_f32) - (\text{Param\_cpst} \rightarrow K\_C * (\text{Param\_cpst} \rightarrow T_v\_C/dT\_f32) * \text{State\_cpst} \rightarrow X1)$$

}]()

#### [SWS\_Mfl\_00057] [

Calculated Output\_value shall be stored to State\_cpst->Y1.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

}]()

#### [SWS\_Mfl\_00058] [

Current input value X\_f32 shall be stored to State\_cpst->X1.

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$

#### 1. PD Set State Value

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

**[SWS\_Mfl\_00059] Definition of API function Mfl\_PDSetState [**

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

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



△

<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Param_cpst	Pointer to internal parameter structure
<b>Return value</b>	None	
<b>Description</b>	The routine sets the parameter structure of a PD element.	
<b>Available via</b>	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] Definition of API function Mfl\_PDOut\_f32** [

<b>Service Name</b>	Mfl_PDOut_f32	
<b>Syntax</b>	float32 Mfl_PDOut_f32 ( const Mfl_StatePD_Type* State_cpst )	
<b>Service ID [hex]</b>	0x2D	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'PD' controller output value
<b>Description</b>	This routine returns 'PD' controllers output value.	
<b>Available via</b>	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] Definition of API function Mfl\_ICalc [

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

]()

#### [SWS\_Mfl\_00069] [

$$Y_n = Y_{n-1} + K * dT * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + K\_f32 * dT\_f32 * \text{State\_cpst} \rightarrow X1$$

]()

#### [SWS\_Mfl\_00070] [

Calculated Output\_value and current input value shall be stored to State\_cpst->Y1 and State\_cpst->X1 respectively.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$

1. 'I' Controller with limitation

]()

**[SWS\_Mfl\_00320] Definition of API function Mfl\_ILimCalc** [

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

]()

**[SWS\_Mfl\_00321]** [

$$Y_n = Y_{n-1} + K * dT * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + K\_f32 * dT\_f32 * \text{State\_cpst} \rightarrow X1$$

]()

**[SWS\_Mfl\_00322]** [

Limit output value with maximum and minimum controller limits.

If (Output\_value < Limit\_cpst->Min\_C) Then,

$$\text{Output\_value} = \text{Limit\_cpst} \rightarrow \text{Min\_C}$$

If (Output\_value > Limit\_cpst->Max\_C) Then,

$$\text{Output\_value} = \text{Limit\_cpst} \rightarrow \text{Max\_C}$$

]()

**[SWS\_Mfl\_00323]** [

Calculated Output\_value and current input value shall be stored to State\_cpst->Y1 and State\_cpst->X1 respectively.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$



1. Set limits for controllers

]()

**[SWS\_Mfl\_00324] Definition of API function Mfl\_CtrlSetLimit [**

<b>Service Name</b>	Mfl_CtrlSetLimit	
<b>Syntax</b>	<pre>void Mfl_CtrlSetLimit (     float32 Min_f32,     float32 Max_f32,     Mfl_Limits_Type* Limit_cpst )</pre>	
<b>Service ID [hex]</b>	0x34	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Min_f32	Minimum limit
	Max_f32	Maximum limit
<b>Parameters (inout)</b>	Limit_cpst	Pointer to limit structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Update limit structure	
<b>Available via</b>	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] Definition of API function Mfl\_CtrlSetLimits [**

<b>Service Name</b>	Mfl_CtrlSetLimits	
<b>Syntax</b>	<pre>void Mfl_CtrlSetLimits (     Mfl_Limits_Type* Limit_cpst,     float32 Min_f32,     float32 Max_f32 )</pre>	
<b>Service ID [hex]</b>	0xC9	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Min_f32	Minimum limit
	Max_f32	Maximum limit
<b>Parameters (inout)</b>	Limit_cpst	Pointer to limit structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Update limit structure	
<b>Available via</b>	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] Definition of API function Mfl\_ISetState [**

<b>Service Name</b>	Mfl_ISetState	
<b>Syntax</b>	<pre>void Mfl_ISetState (     Mfl_StateI_Type* State_cpst,     float32 X1_f32,     float32 Y1_f32 )</pre>	
<b>Service ID [hex]</b>	0x31	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	State_cpst	Pointer to internal state structure
<b>Return value</b>	None	
<b>Description</b>	The routine initialises internal state variables of an I element.	
<b>Available via</b>	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] Definition of API function Mfl\_IOut\_f32** [

<b>Service Name</b>	Mfl_IOut_f32	
<b>Syntax</b>	float32 Mfl_IOut_f32 ( const Mfl_StateI_Type* State_cpst )	
<b>Service ID [hex]</b>	0x33	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'I' controller output value
<b>Description</b>	This routine returns 'I' controllers output value.	
<b>Available via</b>	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] Definition of API function Mfl\_PITyp1Calc [

<b>Service Name</b>	Mfl_PITyp1Calc	
<b>Syntax</b>	<pre>void Mfl_PITyp1Calc (     float32 X_f32,     Mfl_StatePI_Type* State_cpst,     const Mfl_ParamPI_Type* Param_cpst,     float32 dT_f32 )</pre>	
<b>Service ID [hex]</b>	0x35	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	State_cpst	Pointer to the internal state structure.
<b>Return value</b>	None	
<b>Description</b>	This routine computes Proportional plus integral controller (implicit type) output value using differential equation	
<b>Available via</b>	Mfl.h	

}]()

#### [SWS\_Mfl\_00077] [

$$Y_n = Y_{n-1} + K * X_n - K * (1 - dT/T_n) * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + (\text{Param\_cpst} \rightarrow K\_C * X\_f32) - (\text{Param\_cpst} \rightarrow K\_C * (1 - \text{Param\_cpst} \rightarrow Tnrec\_C * dT\_f32) * \text{State\_cpst} \rightarrow X1)$$

}]()

#### [SWS\_Mfl\_00078] [

Calculated Output\_value shall be stored to State\_cpst->Y1.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

}]()

#### [SWS\_Mfl\_00079] [

Current input value X\_f32 shall be stored to State\_cpst->X1.

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$

}]()

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

#### [SWS\_Mfl\_00326] Definition of API function Mfl\_PITyp1LimCalc [

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

]()

#### [SWS\_Mfl\_00327] [

$$Y_n = Y_{n-1} + K * X_n - K * (1 - dT/T_n) * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + (\text{Param\_cpst} \rightarrow K\_C * X\_f32) - (\text{Param\_cpst} \rightarrow K\_C * (1 - \text{Param\_cpst} \rightarrow Tnrec\_C * dT\_f32) * \text{State\_cpst} \rightarrow X1)$$

]()

#### [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] Definition of API function Mfl\_PITyp2Calc** [

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

}]()

**[SWS\_Mfl\_00081]** [

$$Y_n = Y_{n-1} + K * (1 + dT/T_n) * X_n - K * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst}\text{->Y1} + (\text{Param\_cpst}\text{->K\_C} * (1 + \text{Param\_cpst}\text{->Tnrec\_C} * dT\_f32) * X\_f32) - (\text{Param\_cpst}\text{->K\_C} * \text{State\_cpst}\text{->X1})$$

}]()

**[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] Definition of API function Mfl\_PITyp2LimCalc [

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

}]()

[SWS\_Mfl\_00332] [

$$Y_n = Y_{n-1} + K * (1 + dT/T_n) * X_n - K * X_{n-1}$$

This derives implementation:

$$\text{Output\_value} = \text{State\_cpst}\text{->Y1} + (\text{Param\_cpst}\text{->K\_C} * (1 + \text{Param\_cpst}\text{->Tnrec\_C} * dT\_f32) * X\_f32) - (\text{Param\_cpst}\text{->K\_C} * \text{State\_cpst}\text{->X1})$$

}]()

[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] Definition of API function Mfl\_PISetState** [

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

<b>Service Name</b>	Mfl_PISetParam	
<b>Syntax</b>	<pre>void Mfl_PISetParam (     Mfl_ParamPI_Type* Param_cpst,     float32 K_f32,     float32 Tnrec_f32 )</pre>	
<b>Service ID [hex]</b>	0x38	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	K_f32	Amplification factor
	Tnrec_f32	Reciprocal follow-up time
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Param_cpst	Pointer to internal parameter structure
<b>Return value</b>	None	
<b>Description</b>	The routine sets the parameter structure of a PI element.	
<b>Available via</b>	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] Definition of API function Mfl\_PIOut\_f32 [**

<b>Service Name</b>	Mfl_PIOut_f32	
<b>Syntax</b>	<pre>float32 Mfl_PIOut_f32 (     const Mfl_StatePI_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0x39	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return 'PI' controller output value
<b>Description</b>	This routine returns 'PI' controllers output value.	
<b>Available via</b>	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] Definition of API function Mfl\_PIDTyp1Calc [**

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



△

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

]()

**[SWS\_Mfl\_00093]** [

$$Y_n = Y_{n-1} + K * (1 + T_v/dT) * X_n - K * (1 - dT/T_n + 2T_v/dT) * X_{n-1} + K * (T_v/dT) * X_{n-2}$$

This derives implementation:

$$\text{calc1} = \text{Param\_cpst} \rightarrow K\_C * (1 + t\_val) * X\_f32$$

$$\text{calc2} = \text{Param\_cpst} \rightarrow K\_C * (1 - dT\_f32 * \text{Param\_cpst} \rightarrow T_{nrec\_C} + 2 * t\_val) * \text{State\_cpst} \rightarrow X1$$

$$\text{calc3} = \text{Param\_cpst} \rightarrow K\_C * t\_val * \text{State\_cpst} \rightarrow X2$$

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + \text{calc1} - \text{calc2} + \text{calc3}$$

$$\text{Where } t\_val = \text{Param\_cpst} \rightarrow T_v\_C / dT\_f32$$

]()

**[SWS\_Mfl\_00094]** [

Calculated Output\_value shall be stored to State\_cpst->Y1.

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

]()

**[SWS\_Mfl\_00095]** [

Old input value State\_cpst->X1 shall be stored to State\_cpst->X2

$$\text{State\_cpst} \rightarrow X2 = \text{State\_cpst} \rightarrow X1$$

Current input value X\_f32 shall be stored to State\_cpst->X1.

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$

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

]()

**[SWS\_Mfl\_00340] Definition of API function Mfl\_PIDTyp1LimCalc [**

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

]()

**[SWS\_Mfl\_00341] [**

$$Y_n = Y_{n-1} + K * (1 + T_v/dT) * X_n - K * (1 - dT/T_n + 2T_v/dT) * X_{n-1} + K * (T_v/dT) * X_{n-2}$$

This derives implementation:

$$calc1 = Param\_cpst \rightarrow K\_C * (1 + t\_val) * X\_f32$$

$$calc2 = Param\_cpst \rightarrow K\_C * (1 - dT\_f32 * Param\_cpst \rightarrow T_{nrec\_C} + 2 * t\_val) * State\_cpst \rightarrow X1$$

$$calc3 = Param\_cpst \rightarrow K\_C * t\_val * State\_cpst \rightarrow X2$$

$$Output\_value = State\_cpst \rightarrow Y1 + calc1 - calc2 + calc3$$

$$\text{Where } t\_val = Param\_cpst \rightarrow T_v\_C / dT\_f32$$

]()

**[SWS\_Mfl\_00342] [**

Limit output value with maximum and minimum controller limits.

If (Output\_value < Limit\_cpst->Min\_C) Then,

$$Output\_value = Limit\_cpst \rightarrow Min\_C$$

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] Definition of API function Mfl\_PIDTyp2Calc** [

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

}|()

**[SWS\_Mfl\_00097]** [

$$Y_n = Y_{n-1} + K * (1 + dT/T_n + T_v/dT) * X_n - K * (1 + 2T_v/dT) * X_{n-1} + K * (T_v/dT) * X_{n-2}$$

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] Definition of API function Mfl\_PIDTyp2LimCalc** [

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

}]()

**[SWS\_Mfl\_00346]** [

$$Y_n = Y_{n-1} + K * (1 + dT/T_n + Tv/dT) * X_n - K * (1 + 2Tv/dT) * X_{n-1} + K * (Tv/dT) * X_{n-2}$$

This derives implementation:

$$\text{calc1} = \text{Param\_cpst} \rightarrow K\_C * (1 + dT\_f32 * \text{Param\_cpst} \rightarrow Tnrec\_C + t\_val) * X\_f32$$

$$\text{calc2} = \text{Param\_cpst} \rightarrow K\_C * (1 + 2 * t\_val) * \text{State\_cpst} \rightarrow X1$$

$$\text{calc3} = \text{Param\_cpst} \rightarrow K\_C * t\_val * \text{State\_cpst} \rightarrow X2$$

$$\text{Output\_value} = \text{State\_cpst} \rightarrow Y1 + \text{calc1} - \text{calc2} + \text{calc3}$$

$$\text{Where } t\_val = \text{Param\_cpst} \rightarrow Tv\_C / dT\_f32$$

]()

**[SWS\_Mfl\_00347]** [

Limit output value with maximum and minimum controller limits.

If ( $\text{Output\_value} < \text{Limit\_cpst} \rightarrow \text{Min\_C}$ ) Then,

$$\text{Output\_value} = \text{Limit\_cpst} \rightarrow \text{Min\_C}$$

If ( $\text{Output\_value} > \text{Limit\_cpst} \rightarrow \text{Max\_C}$ ) Then,

$$\text{Output\_value} = \text{Limit\_cpst} \rightarrow \text{Max\_C}$$

]()

**[SWS\_Mfl\_00348]** [

Calculated  $\text{Output\_value}$  shall be stored to  $\text{State\_cpst} \rightarrow Y1$ .

$$\text{State\_cpst} \rightarrow Y1 = \text{Output\_value}$$

]()

**[SWS\_Mfl\_00349]** [

Old input value  $\text{State\_cpst} \rightarrow X1$  shall be stored to  $\text{State\_cpst} \rightarrow X2$

$$\text{State\_cpst} \rightarrow X2 = \text{State\_cpst} \rightarrow X1$$

Current input value  $X\_f32$  shall be stored to  $\text{State\_cpst} \rightarrow X1$ .

$$\text{State\_cpst} \rightarrow X1 = X\_f32$$

1. Set 'PID' State Value

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

**[SWS\_Mfl\_00100] Definition of API function Mfl\_PIDSetState [**

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

<b>Service Name</b>	Mfl_PIDSetParam	
<b>Syntax</b>	<pre>void Mfl_PIDSetParam (     Mfl_ParamPID_Type* Param_cpst,     float32 K_f32,     float32 Tv_f32,     float32 Tnrec_f32 )</pre>	





△

<b>Service ID [hex]</b>	0x3D	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	K_f32	Amplification factor
	Tv_f32	Lead Time
	Tnrec_f32	Reciprocal follow-up timer
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Param_cpst	Pointer to internal parameter structure
<b>Return value</b>	None	
<b>Description</b>	The routine sets the parameter structure of a PID element.	
<b>Available via</b>	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] Definition of API function Mfl\_PIDOut\_f32** [

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

]()

[SWS\_Mfl\_00108] [

Output value = State\_cpst->Y1

]()

### 8.5.5 Magnitude and Sign

[SWS\_Mfl\_00110] Definition of API function Mfl\_Abs\_f32 [

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

]()

[SWS\_Mfl\_00111] [

ValAbs = | ValValue |

]()

[SWS\_Mfl\_00112] Definition of API function Mfl\_Sign\_f32 [

<b>Service Name</b>	Mfl_Sign_f32	
<b>Syntax</b>	sint8 Mfl_Sign_f32 ( float32 ValValue )	
<b>Service ID [hex]</b>	0x41	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValValue	Floating-point operand.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	sint8	Integer representing the sign of the operand.
<b>Description</b>	Returns the sign of the argument (ValSign), determined according to the following equation.	
<b>Available via</b>	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] Definition of API function Mfl\_Max\_f32** [

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

}]()

**[SWS\_Mfl\_00117]** [

ValMax = ValValue1, ValValue1 ≥ ValValue2

ValMax = ValValue2, ValValue1 < ValValue2

}]()

**[SWS\_Mfl\_00118] Definition of API function Mfl\_Min\_f32 [**

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

}]()

**[SWS\_Mfl\_00119] [**

Min = Value1, Value1 ≤ Value2

Min = Value2, Value1 > Value2

}]()

**[SWS\_Mfl\_00120] Definition of API function Mfl\_RateLimiter\_f32 [**

<b>Service Name</b>	Mfl_RateLimiter_f32	
<b>Syntax</b>	float32 Mfl_RateLimiter_f32 ( float32 newval, float32 oldval, float32 maxdif )	
<b>Service ID [hex]</b>	0x47	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	newval	Variable to be limited.
	oldval	Previous value of newval.
	maxdif	Absolute maximum difference allowed between previous value (oldval) and the current value (newval).
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Limited value.
<b>Description</b>	An increasing value and decreasing value is rate limited by maxdif	
<b>Available via</b>	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] Definition of API function Mfl\_Limit\_f32** [

<b>Service Name</b>	Mfl_Limit_f32	
<b>Syntax</b>	float32 Mfl_Limit_f32 ( float32 val, float32 lowLim, float32 upLim )	
<b>Service ID [hex]</b>	0x48	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Limited value.
<b>Description</b>	Returns the bounded value (newVal), determined according to the following equation.	
<b>Available via</b>	Mfl.h	

}]()

**[SWS\_Mfl\_00123]** [

newVal = lowLim, val ≤ lowLim

newVal = upLim, val ≥ upLim

newVal = val, lowLim < val < upLim

}]()

### 8.5.7 Logarithms and Exponentials

#### [SWS\_Mfl\_00130] Definition of API function Mfl\_Pow\_f32 [

<b>Service Name</b>	Mfl_Pow_f32	
<b>Syntax</b>	<pre>float32 Mfl_Pow_f32 (     float32 ValBase,     float32 ValExp )</pre>	
<b>Service ID [hex]</b>	0x50	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValBase	Base to be raised to an exponent. Valid range:ValBase > 0.0
	ValExp	Exponent by which to raise the base.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	ValBase raised to ValExp power.
<b>Description</b>	Returns the ValBase raised to ValExp power, determined according to the following equation.	
<b>Available via</b>	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] Definition of API function Mfl\_Sqrt\_f32** [

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

]()

**[SWS\_Mfl\_00136]** [

ValSqrt = ValValue<sup>1/2</sup>

]()

**[SWS\_Mfl\_00137]** [

ValValue shall be passed as positive value. (ValValue ≥ 0)

]()

**[SWS\_Mfl\_00140] Definition of API function Mfl\_Exp\_f32** [

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

]()

**[SWS\_Mfl\_00141]** [

ValExp = eValValue

]()

**[SWS\_Mfl\_00142]** [

Call the function with input value in the range  $[-24\text{PI}, +24\text{PI}]$

]0



**[SWS\_Mfl\_00145] Definition of API function Mfl\_Log\_f32** [

<b>Service Name</b>	Mfl_Log_f32	
<b>Syntax</b>	float32 Mfl_Log_f32 ( float32 ValValue )	
<b>Service ID [hex]</b>	0x54	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValValue	Floating-point operand. Valid range: ValValue > 0.0
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Natural log of ValValue
<b>Description</b>	Returns the natural (base-e) logarithm of the operand (ValLog), determined according to the following equation.	
<b>Available via</b>	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] Definition of API function Mfl\_Sin\_f32** [

<b>Service Name</b>	Mfl_Sin_f32	
<b>Syntax</b>	float32 Mfl_Sin_f32 ( float32 value )	
<b>Service ID [hex]</b>	0x55	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	angle in radians
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	result = sine ( value )
<b>Description</b>	Calculates the sine of the argument.	
<b>Available via</b>	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] Definition of API function Mfl\_Cos\_f32** [

<b>Service Name</b>	Mfl_Cos_f32	
<b>Syntax</b>	float32 Mfl_Cos_f32 ( float32 value )	
<b>Service ID [hex]</b>	0x56	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	angle in radians
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	result = cosine ( value )
<b>Description</b>	Calculates the cosine of the argument.	
<b>Available via</b>	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] Definition of API function Mfl\_Tan\_f32** [

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



△

<b>Description</b>	Calculates the tangent of the argument.
<b>Available via</b>	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] Definition of API function Mfl\_arcSin\_f32** [

<b>Service Name</b>	Mfl_arcSin_f32	
<b>Syntax</b>	float32 Mfl_arcSin_f32 ( float32 value )	
<b>Service ID [hex]</b>	0x58	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	The value whose arc sine is to be returned
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	The arc sine of the argument, in radians
<b>Description</b>	Returns the arc sine of an angle, in the range of -pi/2 through pi/2.	
<b>Available via</b>	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] Definition of API function Mfl\_ArcSin\_f32** [

<b>Service Name</b>	Mfl_ArcSin_f32	
<b>Syntax</b>	float32 Mfl_ArcSin_f32 ( float32 value )	
<b>Service ID [hex]</b>	0xBC	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	The value whose arc sine is to be returned
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	The arc sine of the argument, in radians
<b>Description</b>	Returns the arc sine of an angle, in the range of -pi/2 through pi/2.	
<b>Available via</b>	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] Definition of API function Mfl\_arcCos\_f32** [

<b>Service Name</b>	Mfl_arcCos_f32	
<b>Syntax</b>	float32 Mfl_arcCos_f32 ( float32 value )	
<b>Service ID [hex]</b>	0x59	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	The value whose arc cosine is to be returned
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	The arc cosine of the argument, in radians
<b>Description</b>	Returns the arc cosine of an angle, in the range of 0.0 through pi.	
<b>Available via</b>	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] Definition of API function Mfl\_ArcCos\_f32** [

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





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

}]()

**[SWS\_Mfl\_00356]** [

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

}]()

**[SWS\_Mfl\_00175] Definition of API function Mfl\_arcTan\_f32** [

<b>Service Name</b>	Mfl_arcTan_f32	
<b>Syntax</b>	float32 Mfl_arcTan_f32 ( float32 value )	
<b>Service ID [hex]</b>	0x5A	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	value	The value whose arc tan is to be returned.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	the arc tan of the argument, in radians
<b>Description</b>	Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.	
<b>Available via</b>	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] Definition of API function Mfl\_ArcTan\_f32** [

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



△

<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	the arc tan of the argument, in radians
<b>Description</b>	Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.	
<b>Available via</b>	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] Definition of API function Mfl\_arcTan2\_f32** [

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

 $Result = Z / (1.0 + (0.28 * Z^2))$ 

 if ( $Z < 1$ ) Then

 $Result = (\pi / 2) - (Z / (Z^2 + 0.28))$ 

]()

Note : "This routine (Mfl\_arcTan2\_f32) is depreciated and will not be supported in future release

Replacement routine : Mfl\_ArcTan2\_f32"

**[SWS\_Mfl\_00360] Definition of API function Mfl\_ArcTan2\_f32** [

<b>Service Name</b>	Mfl_ArcTan2_f32	
<b>Syntax</b>	float32 Mfl_ArcTan2_f32 ( float32 y, float32 x )	
<b>Service ID [hex]</b>	0xBF	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	y	y coordinate
	x	x coordinate
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Returns arctan for inputs y and x
<b>Description</b>	Returns the arc tangent of an angle, in the range of [-pi to pi]	
<b>Available via</b>	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 \* Z<sup>2</sup>))

if (x < 0.0f) Then

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

Else

Result = (pi / 2) - (Z / (Z<sup>2</sup> + 0.28))



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

}|()

### 8.5.9 Average

#### [SWS\_Mfl\_00190] Definition of API function Mfl\_Average\_f32\_f32 [

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

}|()

#### [SWS\_Mfl\_00191] [

Output = (Value1 + Value2) / 2

}|()

### 8.5.10 Array Average

#### [SWS\_Mfl\_00192] Definition of API function Mfl\_ArrayAverage\_f32\_f32 [

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



△

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

]()

**[SWS\_Mfl\_00193]** [

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

]()

### 8.5.11 Hypotenuse

**[SWS\_Mfl\_00195] Definition of API function Mfl\_Hypot\_f32f32\_f32** [

<b>Service Name</b>	Mfl_Hypot_f32f32_f32	
<b>Syntax</b>	<pre>float32 Mfl_Hypot_f32f32_f32 (     float32 x_value,     float32 y_value )</pre>	
<b>Service ID [hex]</b>	0x70	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	x_value	First argument Recommended input range: [-24PI, +24PI]
	y_value	Second argument Recommended input range [-24PI, +24PI]
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return value of the function
<b>Description</b>	This service computes the length of a vector	
<b>Available via</b>	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] Definition of datatype Mfl\_ParamRamp\_Type [**

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

]()

**[SWS\_Mfl\_00833] Definition of datatype Mfl\_StateRamp\_Type [**

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

]()

### 8.5.12.1 Ramp routine

#### [SWS\_Mfl\_00201] Definition of API function Mfl\_RampCalc [

<b>Service Name</b>	Mfl_RampCalc	
<b>Syntax</b>	<pre>void Mfl_RampCalc (     float32 X_f32,     Mfl_StateRamp_Type* State_cpst,     const Mfl_ParamRamp_Type* Param_cpst,     float32 dT_f32 )</pre>	
<b>Service ID [hex]</b>	0x90	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	Target value for the ramp to reach
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
<b>Parameters (inout)</b>	State_cpst	Pointer to state structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	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).	
<b>Available via</b>	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\_cpst->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\_cpst->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] Definition of API function Mfl\_RampInitState** [

<b>Service Name</b>	Mfl_RampInitState	
<b>Syntax</b>	<pre>void Mfl_RampInitState (     Mfl_StateRamp_Type* State_cpst,     float32 Val_f32 )</pre>	
<b>Service ID [hex]</b>	0x91	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Val_f32	Initial value for state variable
<b>Parameters (inout)</b>	State_cpst	Pointer to the state structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	Initializes the state, direction and switch parameters for the ramp.	
<b>Available via</b>	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] Definition of API function Mfl\_RampSetParam** [

<b>Service Name</b>	Mfl_RampSetParam	
<b>Syntax</b>	<pre>void Mfl_RampSetParam (     Mfl_ParamRamp_Type* Param_cpst,     float32 SlopePosVal_f32,     float32 SlopeNegVal_f32 )</pre>	
<b>Service ID [hex]</b>	0x92	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	SlopePosVal_f32	Positive slope value
	SlopeNegVal_f32	Negative slope value
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Param_cpst	Pointer to parameter structure
<b>Return value</b>	None	
<b>Description</b>	Sets the slope parameter for the ramp provided by the structure Mfl_ParamRamp_Type.	
<b>Available via</b>	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] Definition of API function Mfl\_RampOut\_f32 [

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

]()

#### [SWS\_Mfl\_00213] [

Return Value = State\_cpcst->State\_f32

]()

### 8.5.12.5 Ramp Jump routine

#### [SWS\_Mfl\_00214] Definition of API function Mfl\_RampCalcJump [

<b>Service Name</b>	Mfl_RampCalcJump	
<b>Syntax</b>	<pre>void Mfl_RampCalcJump (     float32 X_f32,     Mfl_StateRamp_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0x94	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	Target value for ramp to jump
<b>Parameters (inout)</b>	State_cpst	Pointer to the state value
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	<p>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.</p> <p>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.</p>	
<b>Available via</b>	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] Definition of API function Mfl\_RampCalcSwitch\_f32 [

<b>Service Name</b>	Mfl_RampCalcSwitch_f32	
<b>Syntax</b>	<pre>float32 Mfl_RampCalcSwitch_f32 (     float32 Xa_f32,     float32 Xb_f32,     Mfl_StateRamp_Type* State_cpst,     const Mfl_ParamRamp_Type* Param_cpst,     float32 dT_f32 )</pre>	
<b>Service ID [hex]</b>	0x95	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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_cpst	Pointer to the parameter structure which contains the positive and negative slope of the ramp
	dT_f32	Sample Time
<b>Parameters (inout)</b>	State_cpst	Pointer to actual value of the ramp
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Returns the actual state of the ramp
<b>Description</b>	This routine switches ramp between two target values based on the Switch value.	
<b>Available via</b>	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\_cpst->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] Definition of API function Mfl\_RampCalcSwitch** [

<b>Service Name</b>	Mfl_RampCalcSwitch	
<b>Syntax</b>	<pre>float32 Mfl_RampCalcSwitch (     float32 Xa_f32,     float32 Xb_f32,     boolean Switch,     Mfl_StateRamp_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0xCA	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	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
<b>Parameters (inout)</b>	State_cpst	Pointer to StateRamp structure
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Returns the selected target value
<b>Description</b>	This routine switches between two target values for a ramp service based on a Switch parameter.	
<b>Available via</b>	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 Switch = 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] Definition of API function Mfl\_RampGetSwitchPos [

<b>Service Name</b>	Mfl_RampGetSwitchPos	
<b>Syntax</b>	<pre>boolean Mfl_RampGetSwitchPos (     const Mfl_StateRamp_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0x96	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to the state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	return value TRUE or FALSE
<b>Description</b>	Gets the current switch position of ramp switch function.	
<b>Available via</b>	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] Definition of API function Mfl\_RampCheckActivity [

<b>Service Name</b>	Mfl_RampCheckActivity	
<b>Syntax</b>	<pre>boolean Mfl_RampCheckActivity (     const Mfl_StateRamp_Type* State_cpst )</pre>	
<b>Service ID [hex]</b>	0x97	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	State_cpst	Pointer to the state structure
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	return value TRUE or FALSE
<b>Description</b>	This routine checks the status of the ramp and returns a TRUE if the ramp is active, otherwise it returns FALSE.	
<b>Available via</b>	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] Definition of API function Mfl\_HystCenterHalfDelta\_f32\_u8** [

<b>Service Name</b>	Mfl_HystCenterHalfDelta_f32_u8	
<b>Syntax</b>	<pre>boolean Mfl_HystCenterHalfDelta_f32_u8 (     float32 X,     float32 center,     float32 halfDelta,     uint8* State )</pre>	
<b>Service ID [hex]</b>	0xA0	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X	Input value
	center	Center of hysteresis range
	halfDelta	Half width of hysteresis range
<b>Parameters (inout)</b>	State	Pointer to state value
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	Returns TRUE or FALSE depending of input value and state value
<b>Description</b>	Hysteresis with center and left and right side halfDelta switching point.	
<b>Available via</b>	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] Definition of API function Mfl\_HystLeftRight\_f32\_u8 [

<b>Service Name</b>	Mfl_HystLeftRight_f32_u8	
<b>Syntax</b>	<pre>boolean Mfl_HystLeftRight_f32_u8 (     float32 X,     float32 Lsp,     float32 Rsp,     uint8* State )</pre>	
<b>Service ID [hex]</b>	0xA3	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X	Input value
	Lsp	Left switching point
	Rsp	Right switching point
<b>Parameters (inout)</b>	State	Pointer to state value
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	Returns TRUE or FALSE depending of input value and state value
<b>Description</b>	Hysteresis with left and right switching point.	
<b>Available via</b>	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] Definition of API function Mfl\_HystDeltaRight\_f32\_u8 [

<b>Service Name</b>	Mfl_HystDeltaRight_f32_u8	
<b>Syntax</b>	<pre>boolean Mfl_HystDeltaRight_f32_u8 (     float32 X,     float32 Delta,     float32 Rsp,     uint8* State )</pre>	
<b>Service ID [hex]</b>	0xA5	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X	Input value
	Delta	Left switching point = rsp - delta
	Rsp	Right switching point
<b>Parameters (inout)</b>	State	Pointer to state value
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	Returns TRUE or FALSE depending of input value and state value
<b>Description</b>	Hysteresis with right switching point and delta to left switching point	
<b>Available via</b>	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] Definition of API function Mfl\_HystLeftDelta\_f32\_u8 [

<b>Service Name</b>	Mfl_HystLeftDelta_f32_u8	
<b>Syntax</b>	<pre>boolean Mfl_HystLeftDelta_f32_u8 (     float32 X,     float32 Lsp,     float32 Delta,     uint8* State )</pre>	
<b>Service ID [hex]</b>	0xA7	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X	Input value
	Lsp	Left switching point
	Delta	Right switching point = lsp + delta
<b>Parameters (inout)</b>	State	Pointer to state value
<b>Parameters (out)</b>	None	
<b>Return value</b>	boolean	Returns TRUE or FALSE depending of input value and state value
<b>Description</b>	Hysteresis with left switching point and delta to right switching point.	
<b>Available via</b>	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] Definition of API function Mfl\_DeadTime\_f32\_f32 [

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

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



△

	<b>Type</b>	float32
	<b>Comment</b>	Pointer to begin of buffer
	*dtbufEndStatic	
	<b>Type</b>	float32
	<b>Comment</b>	Pointer to end of buffer
<b>Description</b>	Structure definition for Dead Time routine	
<b>Available via</b>	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] Definition of API function Mfl\_Debounce\_u8\_u8 [

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

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

}]()

**[SWS\_Mfl\_00834] Definition of datatype Mfl\_DebounceState\_Type** [

<b>Name</b>	Mfl_DebounceState_Type
<b>Kind</b>	Structure
<b>Elements</b>	XOld



△

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

}]()

### 8.5.15.2 Mfl\_DebounceInit

#### [SWS\_Mfl\_00266] Definition of API function Mfl\_DebounceInit [

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

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

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



△

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

⌋()

[SWS\_Mfl\_00366] ⌈

Param-> TimeHighLow = THighLow

Param-> TimeLowHigh = TLowHigh

⌋()

### 8.5.16 Ascending Sort Routine

[SWS\_Mfl\_00271] Definition of API function Mfl\_SortAscend\_f32 ⌈

<b>Service Name</b>	Mfl_SortAscend_f32	
<b>Syntax</b>	<pre>void Mfl_SortAscend_f32 (     float32* Array,     uint16 Num )</pre>	
<b>Service ID [hex]</b>	0xB5	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Num	Size of an data array
<b>Parameters (inout)</b>	Array	Pointer to an data array
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The sorting algorithm modifies the given input array in ascending order & returns sorted array result via pointer	
<b>Available via</b>	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] Definition of API function Mfl\_SortDescend\_f32 [

<b>Service Name</b>	Mfl_SortDescend_f32	
<b>Syntax</b>	<pre>void Mfl_SortDescend_f32 (     float32* Array,     uint16 Num )</pre>	
<b>Service ID [hex]</b>	0xBA	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	Num	Size of an data array
<b>Parameters (inout)</b>	Array	Pointer to an data array
<b>Parameters (out)</b>	None	
<b>Return value</b>	None	
<b>Description</b>	The sorting algorithm modifies the given input array in descending order & returns sorted array result via pointer	
<b>Available via</b>	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] Definition of API function Mfl\_MedianSort\_f32\_f32 [

<b>Service Name</b>	Mfl_MedianSort_f32_f32	
<b>Syntax</b>	<pre>float32 Mfl_MedianSort_f32_f32 (     float32* Array,     uint8 N )</pre>	
<b>Service ID [hex]</b>	0xBB	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	N	Size of an array
<b>Parameters (inout)</b>	Array	Pointer to an array
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Return value of the function
<b>Description</b>	<p>This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call.</p> <p>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.</p>	
<b>Available via</b>	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] Definition of API function Mfl\_IntToFloatCvrt\_<InTypeMn>\_f32**

[

<b>Service Name</b>	Mfl_IntToFloatCvrt_<InTypeMn>_f32	
<b>Syntax</b>	float32 Mfl_IntToFloatCvrt_<InTypeMn>_f32 ( <InType> ValInteger )	
<b>Service ID [hex]</b>	0xD1 to 0xD6, 0xD9 to 0xDA	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	ValInteger	Integer value to be converted
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Returns the float value
<b>Description</b>	Returns the Float value for the corresponding Integer input.	
<b>Available via</b>	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] Definition of API function Mfl [

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

]()

**[SWS\_Mfl\_00850] Definition of API function Mfl\_Mod\_f32** [

<b>Service Name</b>	Mfl_Mod_f32	
<b>Syntax</b>	<pre>Mfl_Mod_St_Type Mfl_Mod_f32 (     float32 x_f32,     float32 y_f32,     float32* Result )</pre>	
<b>Service ID [hex]</b>	0x100	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	x_f32	dividend
	y_f32	divisor
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Result	Pointer to the Result
	<b>Return value</b>	Returns status of modulus operation E_SUCCESS: Division success E_DIVBYZERO: Divide by Zero E_INVALID: Invalid Operation
<b>Description</b>	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 $y\_f32$ .	
<b>Available via</b>		

]()

**[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 dividend.

- 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

-  $\text{mod}(0, 0)$  or  $\text{mod}(+\infty, +\infty)$

]()

**[SWS\_Mfl\_00844] Definition of API function Mfl\_DivLim\_f32 [**

<b>Service Name</b>	Mfl_DivLim_f32	
<b>Syntax</b>	<pre>Mfl_Div_st Mfl_DivLim_f32 (     float32 X_f32,     float32 Y_f32,     float32 min_f32,     float32 max_f32,     float32* Result )</pre>	
<b>Service ID [hex]</b>	0xDC	
<b>Sync/Async</b>	Synchronous	
<b>Reentrancy</b>	Reentrant	
<b>Parameters (in)</b>	X_f32	X_f32 Dividend
	Y_f32	Y_f32 Divisor
	min_f32	min_f32 minimum limit, min_f32 shall not be strictly greater than max_f32
	max_f32	max_f32 maximum limit, max_f32 shall not be strictly lower than min_f32
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	Result	Result Pointer to the Result
<b>Return value</b>	Mfl_Div_st	Returns status of division E_SUCCESS: Division success E_DIVBYZERO: Divide by Zero E_INVALID: Invalid Operation
<b>Description</b>	Divides X_f32 by Y_f32 and limits the result within min_f32 and max_f32 value.	
<b>Available via</b>	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  $(+/- \infty)/(+/- \infty)$  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.

}]()

**8.5.19 Bit pattern**

**[SWS\_Mfl\_91000] Definition of API function Mfl\_Bitpat\_f32\_u32** [

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

}]()

**8.5.20 Fast inverse square root**

**[SWS\_Mfl\_91001] Definition of API function Mfl\_RSqrt\_f32** [

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



△

<b>Parameters (in)</b>	ValValue	Floating-point operand.
<b>Parameters (inout)</b>	None	
<b>Parameters (out)</b>	None	
<b>Return value</b>	float32	Reciprocal square root of ValValue
<b>Description</b>	Fast reciprocal square root of the operand (ValSqrt)	
<b>Available via</b>	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](#))

## 8.5.21 Trigonometric routines

[SWS\_Mfl\_91003] Definition of API function Mfl\_Cos\_s32\_f32 [

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

]([SRS\\_LIBS\\_00005](#), [SRS\\_LIBS\\_00009](#), [SRS\\_LIBS\\_00011](#))

Algorithm hint (no requirement) for x\_value [-PI/2...PI/2]:

$$\text{Mfl\_Cos\_s32\_f32} ( x\_value ) = c1 + c2 * x\_value^2 + c3 * x\_value^4$$

which is the same as:

$$\text{Mfl\_Cos\_s32\_f32} ( x\_value ) = c1 + x\_value^2 * ( c2 + c3 * x\_value^2 )$$

**[SWS\_Mfl\_91004] Definition of API function Mfl\_Sin\_s32\_f32 [**

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

]([SRS\\_LIBS\\_00005](#), [SRS\\_LIBS\\_00009](#), [SRS\\_LIBS\\_00011](#))

Algorithm hint (no requirement):

$\text{Mfl\_Sin\_s32\_f32}(x\_value) = \text{Mfl\_Cos\_s32\_f32}(1073741824 \text{ (physical: } \pi/2) - x\_value)$

## 8.6 Examples of use of functions

None

## 8.7 Version API

### 8.7.1 Mfl\_GetVersionInfo

**[SWS\_Mfl\_00815] Definition of API function Mfl\_GetVersionInfo [**

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

## B History of Specification Items

Please note that the lists in this chapter also include specification items that have been removed from the specification in a later version. These specification items do not appear as hyperlinks in the document.

### B.1 Specification Item History of this document compared to AUTOSAR R22-11.

#### B.1.1 Added Specification Items in R23-11

Number	Heading
[SWS_Mfl_00850]	Definition of API function Mfl_Mod_f32
[SWS_Mfl_91002]	Definition of datatype Mfl_Mod_St_Type
[SWS_Mfl_91005]	Definition of datatype Mfl_Div_st

**Table B.1: Added Specification Items in R23-11**

#### B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Mfl_00010]	Definition of API function Mfl_Cvrt_<InTypeMn>_f32
[SWS_Mfl_00013]	Definition of API function Mfl_Trunc_f32
[SWS_Mfl_00015]	Definition of API function Mfl_Round_f32
[SWS_Mfl_00018]	Definition of API function Mfl_Ceil_f32
[SWS_Mfl_00020]	Definition of API function Mfl_Floor_f32
[SWS_Mfl_00025]	Definition of datatype Mfl_StatePT1_Type
[SWS_Mfl_00030]	Definition of API function Mfl_POut_f32
[SWS_Mfl_00040]	Definition of API function Mfl_CalcTeQ_f32
[SWS_Mfl_00042]	Definition of API function Mfl_PT1Out_f32
[SWS_Mfl_00045]	
[SWS_Mfl_00053]	Definition of API function Mfl_DT1Typ1Out_f32
[SWS_Mfl_00066]	Definition of API function Mfl_PDOut_f32
[SWS_Mfl_00074]	Definition of API function Mfl_IOut_f32
[SWS_Mfl_00090]	Definition of API function Mfl_PIOut_f32
[SWS_Mfl_00107]	Definition of API function Mfl_PIDOut_f32
[SWS_Mfl_00110]	Definition of API function Mfl_Abs_f32
[SWS_Mfl_00112]	Definition of API function Mfl_Sign_f32





Number	Heading
[SWS_Mfl_00116]	Definition of API function Mfl_Max_f32
[SWS_Mfl_00118]	Definition of API function Mfl_Min_f32
[SWS_Mfl_00120]	Definition of API function Mfl_RateLimiter_f32
[SWS_Mfl_00122]	Definition of API function Mfl_Limit_f32
[SWS_Mfl_00130]	Definition of API function Mfl_Pow_f32
[SWS_Mfl_00135]	Definition of API function Mfl_Sqrt_f32
[SWS_Mfl_00140]	Definition of API function Mfl_Exp_f32
[SWS_Mfl_00145]	Definition of API function Mfl_Log_f32
[SWS_Mfl_00150]	Definition of API function Mfl_Sin_f32
[SWS_Mfl_00155]	Definition of API function Mfl_Cos_f32
[SWS_Mfl_00160]	Definition of API function Mfl_Tan_f32
[SWS_Mfl_00165]	Definition of API function Mfl_arcSin_f32
[SWS_Mfl_00170]	Definition of API function Mfl_arcCos_f32
[SWS_Mfl_00175]	Definition of API function Mfl_arcTan_f32
[SWS_Mfl_00180]	Definition of API function Mfl_arcTan2_f32
[SWS_Mfl_00190]	Definition of API function Mfl_Average_f32_f32
[SWS_Mfl_00192]	Definition of API function Mfl_ArrayAverage_f32_f32
[SWS_Mfl_00195]	Definition of API function Mfl_Hypot_f32f32_f32
[SWS_Mfl_00200]	Definition of datatype Mfl_ParamRamp_Type
[SWS_Mfl_00212]	Definition of API function Mfl_RampOut_f32
[SWS_Mfl_00216]	Definition of API function Mfl_RampCalcSwitch_f32
[SWS_Mfl_00223]	Definition of API function Mfl_RampGetSwitchPos
[SWS_Mfl_00225]	Definition of API function Mfl_RampCheckActivity
[SWS_Mfl_00236]	Definition of API function Mfl_HystCenterHalfDelta_f32_u8
[SWS_Mfl_00241]	Definition of API function Mfl_HystLeftRight_f32_u8
[SWS_Mfl_00246]	Definition of API function Mfl_HystDeltaRight_f32_u8
[SWS_Mfl_00251]	Definition of API function Mfl_HystLeftDelta_f32_u8
[SWS_Mfl_00256]	Definition of API function Mfl_DeadTime_f32_f32
[SWS_Mfl_00259]	Definition of datatype Mfl_DeadTimeParam_Type
[SWS_Mfl_00260]	Definition of API function Mfl_Debounce_u8_u8
[SWS_Mfl_00265]	Definition of datatype Mfl_DebounceParam_Type
[SWS_Mfl_00285]	Definition of API function Mfl_MedianSort_f32_f32
[SWS_Mfl_00305]	
[SWS_Mfl_00310]	Definition of API function Mfl_DT1Typ2Out_f32
[SWS_Mfl_00315]	Definition of API function Mfl_CalcTeQApp_f32
[SWS_Mfl_00350]	Definition of API function Mfl_ArcSin_f32
[SWS_Mfl_00354]	Definition of API function Mfl_ArcCos_f32
[SWS_Mfl_00357]	Definition of API function Mfl_ArcTan_f32
[SWS_Mfl_00360]	Definition of API function Mfl_ArcTan2_f32



△

Number	Heading
[SWS_Mfl_00369]	Definition of API function Mfl_RampCalcSwitch
[SWS_Mfl_00823]	Definition of datatype Mfl_StateDT1Typ1_Type
[SWS_Mfl_00824]	Definition of datatype Mfl_StateDT1Typ2_Type
[SWS_Mfl_00825]	Definition of datatype Mfl_StatePD_Type
[SWS_Mfl_00826]	Definition of datatype Mfl_ParamPD_Type
[SWS_Mfl_00827]	Definition of datatype Mfl_StateI_Type
[SWS_Mfl_00828]	Definition of datatype Mfl_StatePI_Type
[SWS_Mfl_00829]	Definition of datatype Mfl_ParamPI_Type
[SWS_Mfl_00830]	Definition of datatype Mfl_StatePID_Type
[SWS_Mfl_00831]	Definition of datatype Mfl_ParamPID_Type
[SWS_Mfl_00832]	Definition of datatype Mfl_Limits_Type
[SWS_Mfl_00833]	Definition of datatype Mfl_StateRamp_Type
[SWS_Mfl_00834]	Definition of datatype Mfl_DebounceState_Type
[SWS_Mfl_00836]	Definition of API function Mfl_IntToFloatCvrt_<InTypeMn>_f32
[SWS_Mfl_00844]	Definition of API function Mfl_DivLim_f32
[SWS_Mfl_91000]	Definition of API function Mfl_Bitpat_f32_u32
[SWS_Mfl_91001]	Definition of API function Mfl_RSqrt_f32
[SWS_Mfl_91003]	Definition of API function Mfl_Cos_s32_f32
[SWS_Mfl_91004]	Definition of API function Mfl_Sin_s32_f32

**Table B.2: Changed Specification Items in R23-11**

### B.1.3 Deleted Specification Items in R23-11

Number	Heading
[SWS_Mfl_00849]	

**Table B.3: Deleted Specification Items in R23-11**