

Document Title	Specification of Floating Point Math Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	397
Document Classification	Standard
Document Version	1.5.0
Document Status	Final
Part of Release	4.1
Revision	3

Document Change History			
Date	Version	Changed by	Change Description
31.03.2014	1.5.0	AUTOSAR Release Management	<ul style="list-style-type: none"> Removed: SWS_Efx_00206, SWS_Efx_00207 and SWS_Efx_00281 from Mfl_RampCalc & Mfl_RampCalcJump functions.
31.10.2013	1.4.0	AUTOSAR Release Management	<ul style="list-style-type: none"> Deprecated: Mfl_DeadTime function Removed: SWS_Mfl_00197 from Mfl_Hypot function Added: SWS_Mfl_00835 for Mfl_RampCalc function, a note for Mfl_RampGetSwitchPos function Modified: Description for Mfl_RampSetParam function, Parameter (in) definition for Mfl_RateLimiter_f32 Editorial changes
27.02.2013	1.3.0	AUTOSAR Administration	<ul style="list-style-type: none"> Description and requirements are modified for Mfl_RampCalcJump, Mfl_RampCalc Formatting error in superscripts are corrected Corrected "DT1" to "I" in I-Controller functions Description of the parameter "State" is corrected in Mfl_Debounce and Mfl_DebounceInIt functions Corrected for 'DependencyOnArtifact'
09.12.2011	1.2.0	AUTOSAR Administration	<ul style="list-style-type: none"> Removal of 'Accumulator routine' Revised 'Trigonometric routines' names Added 'Median Sort Routines'

Document Change History

Date	Version	Changed by	Change Description
15.11.2010	1.1.0	AUTOSAR Administration	<ul style="list-style-type: none">• Introduction of additional LIMITED Functions for controllers• Ramp functions optimised for effective usage• Separation of DT1 Type 1 and Type 2 Controller functions• Introduction of additional approximative function for calculatio of TeQ
07.12.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

This specification 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 specification.

The material contained in this specification is protected by copyright and other types of Intellectual Property Rights. The commercial exploitation of the material contained in this specification requires a license to such Intellectual Property Rights.

This specification 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 specification may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The AUTOSAR specifications have been developed for automotive applications only. They have neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1	Introduction and functional overview	6
2	Acronyms and abbreviations	7
3	Related documentation.....	8
3.1	Input documents.....	8
3.2	Related standards and norms	8
4	Constraints and assumptions	9
4.1	Limitations	9
4.2	Applicability to car domains.....	9
5	Dependencies to other modules.....	10
5.1	File structure	10
6	Requirements traceability	11
7	Functional specification	20
7.1	Error classification	20
7.2	Error detection.....	20
7.3	Error notification	20
7.4	Initialization and shutdown	20
7.5	Using Library API	20
7.6	library implementation	21
8	Routine specification	23
8.1	Imported types.....	23
8.2	Type definitions	23
8.3	Comment about rounding.....	23
8.4	Comment about routines optimized for target	24
8.5	Routine definitions.....	25
8.5.1	Floating point to Fixed-Point Conversion	25
8.5.2	Fixed-Point to Floating-Point Conversion	25
8.5.3	Rounding	26
8.5.4	Controller routines	28
8.5.5	Magnitude and Sign.....	59
8.5.6	Limiting	60
8.5.7	Logarithms and Exponentials	62
8.5.8	Trigonometry.....	64
8.5.9	Average	70
8.5.10	Array Average.....	71
8.5.11	Hypotenuse	71
8.5.12	Ramp routines	72
8.5.13	Hysteresis routines	79
8.5.14	Mfl_DeadTime	83
8.5.15	Debounce routines.....	84
8.5.16	Ascending Sort Routine	87
8.5.17	Descending Sort Routine.....	88

8.5.18	Median sort routine	88
8.6	Examples of use of functions	89
8.7	Version API	89
8.7.1	Mfl_GetVersionInfo	89
8.8	Call-back notifications	90
8.9	Scheduled functions	90
8.10	Expected Interfaces.....	90
8.10.1	Mandatory Interfaces	90
8.10.2	Optional Interfaces.....	90
8.10.3	Configurable interfaces.....	91
9	Sequence diagrams	92
10	Configuration specification.....	93
10.1	Published Information.....	93
10.2	Configuration option	93
11	Not applicable requirements	94

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.

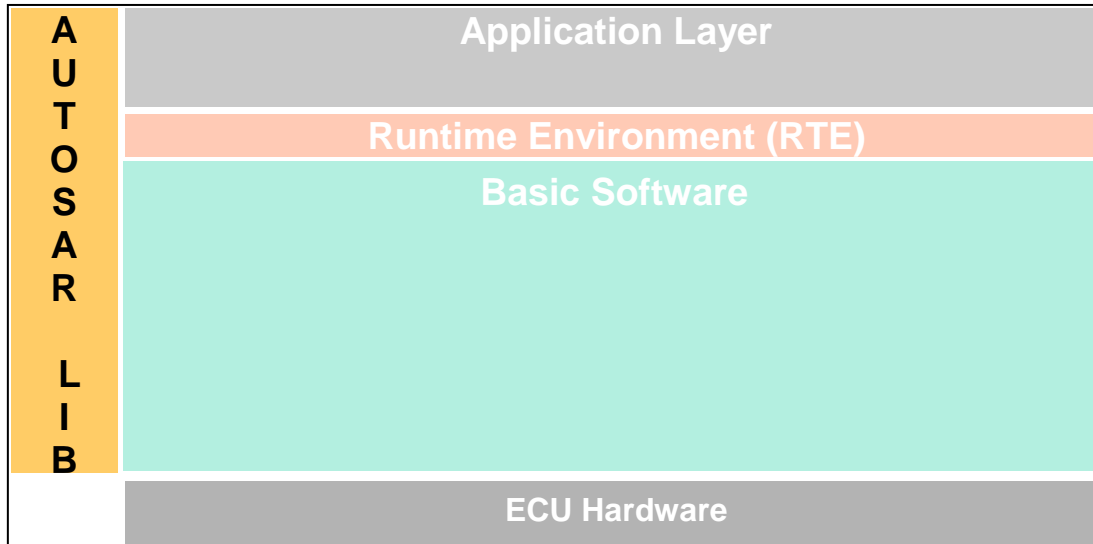


Figure : Layered architecture

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	Development Error Tracer
f32	Mnemonic for the float32, specified in AUTOSAR_SWS_PlatformTypes
f64	Mnemonic for the float64, 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

- [1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf
- [2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf
- [3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf
- [4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf
- [5] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf
- [6] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf
- [7] Requirement on Libraries,
AUTOSAR_SRS_Libraries.pdf
- [8] Memory mapping mechanism,
AUTOSAR_SRS_MemoryMapping.pdf

3.2 Related standards and norms

- [10] ISO/IEC 9899:1990 Programming Language – C
- [11] MISRA-C 2004: Guidelines for the use of the C language in critical systems, October 2004

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_'.
- Header file Mfl.h provides all public function prototypes and types defined by the Mfl library specification] (SRS_LIBS_00005)

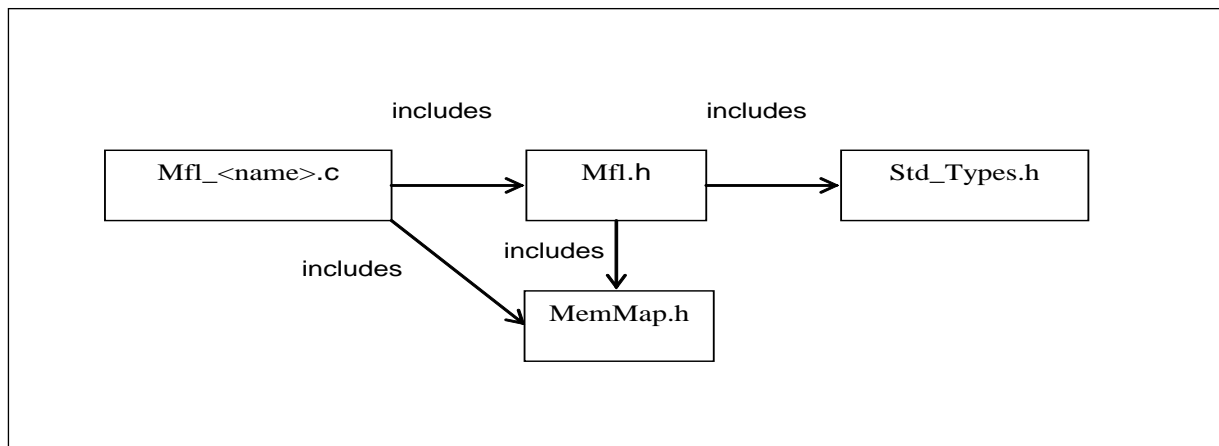


Figure : File structure

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 traceability

Requirement	Description	Satisfied by
-	-	SWS_Mfl_00005
-	-	SWS_Mfl_00006
-	-	SWS_Mfl_00007
-	-	SWS_Mfl_00008
-	-	SWS_Mfl_00009
-	-	SWS_Mfl_00010
-	-	SWS_Mfl_00011
-	-	SWS_Mfl_00012
-	-	SWS_Mfl_00013
-	-	SWS_Mfl_00015
-	-	SWS_Mfl_00017
-	-	SWS_Mfl_00018
-	-	SWS_Mfl_00020
-	-	SWS_Mfl_00025
-	-	SWS_Mfl_00026
-	-	SWS_Mfl_00027
-	-	SWS_Mfl_00030
-	-	SWS_Mfl_00031
-	-	SWS_Mfl_00032
-	-	SWS_Mfl_00033
-	-	SWS_Mfl_00034
-	-	SWS_Mfl_00035
-	-	SWS_Mfl_00036
-	-	SWS_Mfl_00037
-	-	SWS_Mfl_00038
-	-	SWS_Mfl_00039
-	-	SWS_Mfl_00040
-	-	SWS_Mfl_00041
-	-	SWS_Mfl_00042
-	-	SWS_Mfl_00043
-	-	SWS_Mfl_00044
-	-	SWS_Mfl_00045
-	-	SWS_Mfl_00046
-	-	SWS_Mfl_00047
-	-	SWS_Mfl_00048
-	-	SWS_Mfl_00049
-	-	SWS_Mfl_00050

-	-	SWS_Mfl_00051
-	-	SWS_Mfl_00053
-	-	SWS_Mfl_00054
-	-	SWS_Mfl_00055
-	-	SWS_Mfl_00056
-	-	SWS_Mfl_00057
-	-	SWS_Mfl_00058
-	-	SWS_Mfl_00059
-	-	SWS_Mfl_00060
-	-	SWS_Mfl_00061
-	-	SWS_Mfl_00062
-	-	SWS_Mfl_00063
-	-	SWS_Mfl_00064
-	-	SWS_Mfl_00066
-	-	SWS_Mfl_00067
-	-	SWS_Mfl_00068
-	-	SWS_Mfl_00069
-	-	SWS_Mfl_00070
-	-	SWS_Mfl_00071
-	-	SWS_Mfl_00072
-	-	SWS_Mfl_00073
-	-	SWS_Mfl_00074
-	-	SWS_Mfl_00075
-	-	SWS_Mfl_00076
-	-	SWS_Mfl_00077
-	-	SWS_Mfl_00078
-	-	SWS_Mfl_00079
-	-	SWS_Mfl_00080
-	-	SWS_Mfl_00081
-	-	SWS_Mfl_00082
-	-	SWS_Mfl_00083
-	-	SWS_Mfl_00084
-	-	SWS_Mfl_00085
-	-	SWS_Mfl_00086
-	-	SWS_Mfl_00087
-	-	SWS_Mfl_00088
-	-	SWS_Mfl_00089
-	-	SWS_Mfl_00090
-	-	SWS_Mfl_00091

-	-	SWS_Mfl_00092
-	-	SWS_Mfl_00093
-	-	SWS_Mfl_00094
-	-	SWS_Mfl_00095
-	-	SWS_Mfl_00096
-	-	SWS_Mfl_00097
-	-	SWS_Mfl_00098
-	-	SWS_Mfl_00099
-	-	SWS_Mfl_00100
-	-	SWS_Mfl_00101
-	-	SWS_Mfl_00102
-	-	SWS_Mfl_00103
-	-	SWS_Mfl_00104
-	-	SWS_Mfl_00105
-	-	SWS_Mfl_00106
-	-	SWS_Mfl_00107
-	-	SWS_Mfl_00108
-	-	SWS_Mfl_00110
-	-	SWS_Mfl_00111
-	-	SWS_Mfl_00112
-	-	SWS_Mfl_00113
-	-	SWS_Mfl_00114
-	-	SWS_Mfl_00115
-	-	SWS_Mfl_00116
-	-	SWS_Mfl_00117
-	-	SWS_Mfl_00118
-	-	SWS_Mfl_00119
-	-	SWS_Mfl_00120
-	-	SWS_Mfl_00121
-	-	SWS_Mfl_00122
-	-	SWS_Mfl_00123
-	-	SWS_Mfl_00130
-	-	SWS_Mfl_00131
-	-	SWS_Mfl_00132
-	-	SWS_Mfl_00133
-	-	SWS_Mfl_00135
-	-	SWS_Mfl_00136
-	-	SWS_Mfl_00137
-	-	SWS_Mfl_00140

-	-	SWS_Mfl_00141
-	-	SWS_Mfl_00142
-	-	SWS_Mfl_00145
-	-	SWS_Mfl_00146
-	-	SWS_Mfl_00147
-	-	SWS_Mfl_00150
-	-	SWS_Mfl_00151
-	-	SWS_Mfl_00152
-	-	SWS_Mfl_00155
-	-	SWS_Mfl_00156
-	-	SWS_Mfl_00157
-	-	SWS_Mfl_00160
-	-	SWS_Mfl_00161
-	-	SWS_Mfl_00163
-	-	SWS_Mfl_00165
-	-	SWS_Mfl_00167
-	-	SWS_Mfl_00168
-	-	SWS_Mfl_00170
-	-	SWS_Mfl_00172
-	-	SWS_Mfl_00175
-	-	SWS_Mfl_00177
-	-	SWS_Mfl_00180
-	-	SWS_Mfl_00182
-	-	SWS_Mfl_00183
-	-	SWS_Mfl_00190
-	-	SWS_Mfl_00191
-	-	SWS_Mfl_00192
-	-	SWS_Mfl_00193
-	-	SWS_Mfl_00195
-	-	SWS_Mfl_00196
-	-	SWS_Mfl_00200
-	-	SWS_Mfl_00201
-	-	SWS_Mfl_00202
-	-	SWS_Mfl_00208
-	-	SWS_Mfl_00209
-	-	SWS_Mfl_00210
-	-	SWS_Mfl_00211
-	-	SWS_Mfl_00212
-	-	SWS_Mfl_00213

-	-	SWS_Mfl_00214
-	-	SWS_Mfl_00215
-	-	SWS_Mfl_00216
-	-	SWS_Mfl_00217
-	-	SWS_Mfl_00218
-	-	SWS_Mfl_00219
-	-	SWS_Mfl_00220
-	-	SWS_Mfl_00221
-	-	SWS_Mfl_00222
-	-	SWS_Mfl_00223
-	-	SWS_Mfl_00224
-	-	SWS_Mfl_00225
-	-	SWS_Mfl_00226
-	-	SWS_Mfl_00236
-	-	SWS_Mfl_00237
-	-	SWS_Mfl_00238
-	-	SWS_Mfl_00239
-	-	SWS_Mfl_00240
-	-	SWS_Mfl_00241
-	-	SWS_Mfl_00242
-	-	SWS_Mfl_00243
-	-	SWS_Mfl_00244
-	-	SWS_Mfl_00245
-	-	SWS_Mfl_00246
-	-	SWS_Mfl_00247
-	-	SWS_Mfl_00248
-	-	SWS_Mfl_00249
-	-	SWS_Mfl_00250
-	-	SWS_Mfl_00251
-	-	SWS_Mfl_00252
-	-	SWS_Mfl_00253
-	-	SWS_Mfl_00254
-	-	SWS_Mfl_00255
-	-	SWS_Mfl_00256
-	-	SWS_Mfl_00257
-	-	SWS_Mfl_00258
-	-	SWS_Mfl_00259
-	-	SWS_Mfl_00260
-	-	SWS_Mfl_00261

-	-	SWS_Mfl_00262
-	-	SWS_Mfl_00263
-	-	SWS_Mfl_00264
-	-	SWS_Mfl_00265
-	-	SWS_Mfl_00266
-	-	SWS_Mfl_00267
-	-	SWS_Mfl_00268
-	-	SWS_Mfl_00269
-	-	SWS_Mfl_00270
-	-	SWS_Mfl_00271
-	-	SWS_Mfl_00273
-	-	SWS_Mfl_00275
-	-	SWS_Mfl_00276
-	-	SWS_Mfl_00285
-	-	SWS_Mfl_00287
-	-	SWS_Mfl_00288
-	-	SWS_Mfl_00289
-	-	SWS_Mfl_00300
-	-	SWS_Mfl_00301
-	-	SWS_Mfl_00302
-	-	SWS_Mfl_00303
-	-	SWS_Mfl_00304
-	-	SWS_Mfl_00305
-	-	SWS_Mfl_00306
-	-	SWS_Mfl_00307
-	-	SWS_Mfl_00308
-	-	SWS_Mfl_00310
-	-	SWS_Mfl_00311
-	-	SWS_Mfl_00315
-	-	SWS_Mfl_00316
-	-	SWS_Mfl_00320
-	-	SWS_Mfl_00321
-	-	SWS_Mfl_00322
-	-	SWS_Mfl_00323
-	-	SWS_Mfl_00324
-	-	SWS_Mfl_00325
-	-	SWS_Mfl_00326
-	-	SWS_Mfl_00327
-	-	SWS_Mfl_00328

-	-	SWS_Mfl_00329
-	-	SWS_Mfl_00330
-	-	SWS_Mfl_00331
-	-	SWS_Mfl_00332
-	-	SWS_Mfl_00333
-	-	SWS_Mfl_00334
-	-	SWS_Mfl_00335
-	-	SWS_Mfl_00340
-	-	SWS_Mfl_00341
-	-	SWS_Mfl_00342
-	-	SWS_Mfl_00343
-	-	SWS_Mfl_00344
-	-	SWS_Mfl_00345
-	-	SWS_Mfl_00346
-	-	SWS_Mfl_00347
-	-	SWS_Mfl_00348
-	-	SWS_Mfl_00349
-	-	SWS_Mfl_00350
-	-	SWS_Mfl_00352
-	-	SWS_Mfl_00353
-	-	SWS_Mfl_00354
-	-	SWS_Mfl_00356
-	-	SWS_Mfl_00357
-	-	SWS_Mfl_00359
-	-	SWS_Mfl_00360
-	-	SWS_Mfl_00362
-	-	SWS_Mfl_00363
-	-	SWS_Mfl_00365
-	-	SWS_Mfl_00366
-	-	SWS_Mfl_00367
-	-	SWS_Mfl_00368
-	-	SWS_Mfl_00369
-	-	SWS_Mfl_00370
-	-	SWS_Mfl_00371
-	-	SWS_Mfl_00372
-	-	SWS_Mfl_00821
-	-	SWS_Mfl_00822
-	-	SWS_Mfl_00823
-	-	SWS_Mfl_00824

-	-	SWS_Mfl_00825
-	-	SWS_Mfl_00826
-	-	SWS_Mfl_00827
-	-	SWS_Mfl_00828
-	-	SWS_Mfl_00829
-	-	SWS_Mfl_00830
-	-	SWS_Mfl_00831
-	-	SWS_Mfl_00832
-	-	SWS_Mfl_00833
-	-	SWS_Mfl_00834
-	-	SWS_Mfl_00835
SRS_BSW_00003	All software modules shall provide version and identification information	SWS_Mfl_00815
SRS_BSW_00007	All Basic SW Modules written in C language shall conform to the MISRA C 2004 Standard.	SWS_Mfl_00809
SRS_BSW_00304	-	SWS_Mfl_00812
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Mfl_00813
SRS_BSW_00318	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	SWS_Mfl_00815
SRS_BSW_00321	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	SWS_Mfl_00815
SRS_BSW_00348	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	SWS_Mfl_00811
SRS_BSW_00374	All Basic Software Modules shall provide a readable module vendor identification	SWS_Mfl_00814
SRS_BSW_00378	AUTOSAR shall provide a boolean type	SWS_Mfl_00812
SRS_BSW_00379	All software modules shall provide a module identifier in the header file and in the module XML description file.	SWS_Mfl_00814
SRS_BSW_00402	Each module shall provide version information	SWS_Mfl_00814
SRS_BSW_00407	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	SWS_Mfl_00815, SWS_Mfl_00816
SRS_BSW_00411	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	SWS_Mfl_00816
SRS_BSW_00436	-	SWS_Mfl_00810
SRS_LIBS_00001	The functional behavior of each library functions shall not be configurable	SWS_Mfl_00818
SRS_LIBS_00002	A library shall be operational before all BSW modules and application SW-Cs	SWS_Mfl_00800
SRS_LIBS_00003	A library shall be operational until the shutdown	SWS_Mfl_00801
SRS_LIBS_00005	Each library shall provide one header file with its public interface	SWS_Mfl_00001
SRS_LIBS_00013	The error cases, resulting in the check at runtime of the	SWS_Mfl_00817,

	value of input parameters, shall be listed in SWS	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

7 Functional specification

7.1 Error classification

[SWS_Mfl_00821]

No error classification definition as DET call not supported by library

()

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 HIS subset of the MISRA C Standard.

Only in technically reasonable, exceptional cases MISRA violations are permissible. Such violations against MISRA rules shall be clearly identified and documented within comments in the C source code (including rationale why MISRA rule is violated). The comment shall be placed right above the line of code which causes the violation and have the following syntax:

```
/* MISRA RULE XX VIOLATION: This the reason why the MISRA rule could not be followed in this special case*/] (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_00436)

[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)

8 Routine specification

8.1 Imported types

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

Header file	Imported Type
Std_Types.h	boolean, sint8, uint8, sint16, uint16, sint32, uint32, float32, float64

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 PlatformTypes.h [AUTOSAR_SWS_PlatformTypes]. The following mnemonic are used in the library routine names.

Size	Platform Type	Mnemonic	Range
unsigned 8-Bit	boolean	NA	[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	NA
64-Bit	float64	f64	NA

Table 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>).

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] [

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

] ()

[SWS_Mfl_00006][

Result = ValFloat * 2^{ValFixedExponent}

]()

[SWS_Mfl_00007][

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

]()

[SWS_Mfl_00008][

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

]()

Function ID and prototypes

[SWS_Mfl_00009][

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

]()

8.5.2 Fixed-Point to Floating-Point Conversion

[SWS_Mfl_00010] [

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

] ()

[SWS_Mfl_00011]

Result = ValFixedInteger * 2^{-ValFixedExponent}

] ()

Function ID and prototypes

[SWS_Mfl_00012] [

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

] ()

8.5.3 Rounding

[SWS_Mfl_00013] [

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

] ()

For example:

36.56 will be truncated to 36.00

[SWS_Mfl_00015] [

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

] ()

For example:

36.56 will be rounded to 37.00

[SWS_Mfl_00017][

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

]()

For example:

36.5 will be rounded to 37.00

[SWS_Mfl_00018] [

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

] ()

[SWS_Mfl_00020] [

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

] ()

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
Y_n	Actual output to calculate
Y_{n-1}	Output value, one time step before
X_n	Actual input, given from the input
X_{n-1}	Input, one time step before
X_{n-2}	Input, two time steps before
X_1	Input, n-1 time steps before
X_0	Input, n time steps before
$a_1, b_0, b_1, b_2, b_{n-1}, b_n$	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 X_n 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, $TeQ = \exp (dT/ T1)$.

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

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

[SWS_Mfl_00025] [

Name:	Mfl_StatePT1_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for PT1 controller routine		

] ()

[SWS_Mfl_00823] [

Name:	Mfl_StateDT1Typ1_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	X2	Input value, two time steps before
	float32	Y1	Output value, one time step before
Description:	System State Structure for DT1-Type1 controller routine		

] ()

[SWS_Mfl_00824] [

Name:	Mfl_StateDT1Typ2_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for DT1-Type2 controller routine		

] ()

[SWS_Mfl_00825] [

Name:	Mfl_StatePD_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for PD controller routine		

] ()

[SWS_Mfl_00826] [

Name:	Mfl_ParamPD_Type		
Type:	Structure		
Element:	float32	K_C	Amplification factor
	float32	Tv_C	Lead time
Description:	System and Time equivalent parameter Structure for PD controller routine		

⌋()

[SWS_Mfl_00827]⌈

Name:	Mfl_StateI_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for I controller routine		

⌋()

[SWS_Mfl_00828]⌈

Name:	Mfl_StatePI_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for PI additive (<i>Type1 and Type 2</i>) controller routine		

⌋()

[SWS_Mfl_00829]⌈

Name:	Mfl_ParamPI_Type		
Type:	Structure		
Element:	float32	K_C	Amplification factor
	float32	Tnrec_C	Reciprocal follow up time (1/Tn)
Description:	System and Time equivalent parameter Structure for PI additive (<i>Type1 and Type 2</i>) controller routine		

⌋()

[SWS_Mfl_00830]⌈

Name:	Mfl_StatePID_Type		
Type:	Structure		
Element:	float32	X1	Input value, one time step before
	float32	X2	Input value, two time step before
	float32	Y1	Output value, one time step before
Description:	System State Structure for PID additive (<i>Type1 and Type 2</i>) controller routine		

⌋()

[SWS_Mfl_00831]⌈

Name:	Mfl_ParamPID_Type		
Type:	Structure		
Element:	float32	K_C	Amplification factor
	float32	Tv_C	Reciprocal follow up time (1/Tn)
	float32	Tnrec_C	Lead time
Description:	System and Time equivalent parameter Structure for PID additive (<i>Type1 and Type 2</i>) controller routine		

⌋()

[SWS_Mfl_00832]⌈

Name:	Mfl_Limits_Type		
Type:	Structure		
Element:	float32	Min_C	Minimum limit value
	float32	Max_C	Maximum limit value
Description:	Controller limit value structure		

⌋()

8.5.4.2 Proportional Controller

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

8.5.4.2.1 'P' Controller

[SWS_Mfl_00026] ⌈

Service name:	Mfl_PCcalc		
Syntax:	<pre>void Mfl_PCcalc(float32 X_f32, float32* P_pf32, float32 K_f32)</pre>		
Service ID[hex]:	0x10		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (in):	X_f32	input value	
	K_f32	Amplification factor	
Parameters (in-out):	P_pf32	Pointer to the calculated state	
Parameters (out):	None		
Return value:	None		
Description:	Differential equation: $Y = K * X$		

⌋()

[SWS_Mfl_00027]⌈

Implemented difference equation:

$$*P_pf32 = K_f32 * X_f32$$

⌋()

8.5.4.2.2 Get 'P' output

This routine can be realised using inline function.

[SWS_Mfl_00030] ⌈

Service name:	Mfl_POut_f32		
Syntax:	<pre>float32 Mfl_POut_f32(const float32* const P_pf32)</pre>		
Service ID[hex]:	0x12		
Sync/Async:	Synchronous		

Reentrancy:	Reentrant	
Parameters (in):	P_pf32	Pointer to the calculated state
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return 'P' controller output value
Description:	This routine returns 'P' controllers output value limited by the return data type	

] ()

[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

8.5.4.3.1 'PT1' Controller

[SWS_Mfl_00032] [

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

] ()

[SWS_Mfl_00033][

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

This derives implementation:

$$\text{Output_value} = (\text{TeQ_f32} * \text{State_cpst} \rightarrow Y1) + K_f32 * (1 - \text{TeQ_f32}) * \text{State_cpst} \rightarrow X1$$

where $\text{TeQ_f32} = \exp(-dT/T1)$

]()

[SWS_Mfl_00034]

Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter TeQ_f32.

l()

[SWS_Mfl_00035]

If (T1 = 0) then PT1 controller follows Input value,
State_cpst->Y1 = k_f32 * X_f32

l()

[SWS_Mfl_00036]

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

State_cpst->Y1 = Output_value

State_cpst->X1 = X_f32

l()

8.5.4.3.2 'PT1' Set State Value

This routine can be realised using inline function.

[SWS_Mfl_00037] [

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

] ()

[SWS_Mfl_00038]

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

l()

[SWS_Mfl_00039]

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32.

l()

8.5.4.3.3 Calculate time equivalent Value

This routine can be realised using inline function.

[SWS_Mfl_00040] [

Service name:	Mfl_CalcTeQ_f32	
Syntax:	float32 Mfl_CalcTeQ_f32 (float32 T1rec_f32, float32 dT_f32)	
Service ID[hex]:	0x1C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	T1rec_f32	Reciprocal delay time
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Time Equivalent TeQ
Description:	This routine calculates time equivalent factor	

] ()

[SWS_Mfl_00041]

$TeQ = \exp(-T1rec_f32 * dT_f32)$

l()

8.5.4.3.4 Calculate an approximate time equivalent Value

This routine calculates approximate time equivalent and can be realised using inline function

[SWS_Mfl_00315] [

Service name:	Mfl_CalcTeQApp_f32	
Syntax:	float32 Mfl_CalcTeQApp_f32 (float32 T1rec_f32, float32 dT_f32)	
Service ID[hex]:	0x1E	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	T1rec_f32	Reciprocal delay time
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Time Equivalent TeQ
Description:	This routine calculates time equivalent factor	

] ()

[SWS_Mfl_00316]

TeQApp = 1 - (T1rec_f32 * dT_f32)

] ()

8.5.4.3.5 Get 'PT1' output

This routine can be realised using inline function.

[SWS_Mfl_00042] [

Service name:	Mfl_PT1Out_f32
Syntax:	float32 Mfl_PT1Out_f32(const Mfl_StatePT1_Type* const State_cpst)
Service ID[hex]:	0x1D
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	State_cpst Pointer to state structure
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 Return 'PT1' controller output value
Description:	This routine returns 'PT1' controllers output value

] ()

[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 8.5.4.3.3, shall be used for Mfl_DT1_f32 function to calculate the time equivalent TeQ.

8.5.4.4.1 'DT1' Controller - Type1

[SWS_Mfl_00044] [

Service name:	Mfl_DT1Typ1Calc
Syntax:	void Mfl_DT1Typ1Calc(float32 X_f32, Mfl_StateDT1Typ1_Type* const State_cpst, float32 K_f32, float32 TeQ_f32, float32 dT_f32)
Service ID[hex]:	0x20
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the DT1 controller
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
	dT_f32	Sample Time
Parameters (in-out):	State_cpst	Pointer to state structure
Parameters (out):	None	
Return value:	None	
Description:	This routine computes DT1 controller output value using differential equation	

] ()

[SWS_Mfl_00045]

$$Y_n = \exp(-dT/T_1) * Y_{n-1} + K * (1 - \exp(-dT/T_1)) * ((X_{n-1} - X_{n-2}) / dT)$$

This derives implementation:

$$\text{Output_value} = (\text{TeQ} * \text{State_cpst} \rightarrow Y_1) + K_f32 * (1 - \text{TeQ}) * ((\text{State_cpst} \rightarrow X_1 - \text{State_cpst} \rightarrow X_2) / dT)$$

where $\text{TeQ} = \exp(-dT/T_1)$

]()

[SWS_Mfl_00046]

Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter

TeQ_f32.

]()

[SWS_Mfl_00047]

If $(T_1 = 0)$ then DT1 controller follows Input value,

$$\text{Output_value} = k_f32 * (X_f32 - \text{State_cpst} \rightarrow X_1) / dT$$

]()

[SWS_Mfl_00048]

Calculated Output_value shall be stored to State_cpst->Y1.

$$\text{State_cpst} \rightarrow Y_1 = \text{Output_value}$$

]()

[SWS_Mfl_00049]

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

$$\text{State_cpst} \rightarrow X_2 = \text{State_cpst} \rightarrow X_1$$

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

$$\text{State_cpst} \rightarrow X_1 = X_f32$$

]()

8.5.4.4.2 'DT1' Controller - Type2

[SWS_Mfl_00300] [

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

] ()

[SWS_Mfl_00301]

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

This derives implementation:

$$\text{Output_value} = (\text{TeQ} * \text{State_cpst} \rightarrow Y_1) + K_f32 * (1 - \text{TeQ}) * ((X_f32 - \text{State_cpst} \rightarrow X_1) / dT)$$

where $\text{TeQ} = \exp(-dT/T_1)$

] ()

[SWS_Mfl_00302]

Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter TeQ_f32.

] ()

[SWS_Mfl_00303]

If ($T_1 = 0$) then DT1 controller follows Input value,

$$\text{Output_value} = k_f32 * (X_f32 - \text{State_cpst} \rightarrow X_1) / dT$$

] ()

[SWS_Mfl_00304]

Calculated Output_value shall be stored to State_cpst->Y1.

$$\text{State_cpst} \rightarrow Y_1 = \text{Output_value}$$

] ()

[SWS_Mfl_00305]

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

$$\text{State_cpst} \rightarrow X_1 = X_f32$$

] ()

8.5.4.4.3 Set 'DT1' State Value – Type1

This routine can be realised using inline function.

[SWS_Mfl_00050] [

Service name:	Mfl_DT1Typ1SetState	
Syntax:	<pre>void Mfl_DT1Typ1SetState(Mfl_StateDT1Typ1_Type* const State_cpst, float32 X1_f32, float32 X2_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for the input state X1
	X2_f32	Initial value for the input state X2
	Y1_f32	Initial value for the output state
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of a DT1 element.	

] ()

[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

]()

8.5.4.4.4 Set 'DT1' State Value – Type2

This routine can be realised using inline function.

[SWS_Mfl_00306] [

Service name:	Mfl_DT1Typ2SetState	
Syntax:	<pre>void Mfl_DT1Typ2SetState(Mfl_StateDT1Typ2_Type* const State_cpst, float32 X1_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0xC1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for the input state
	Y1_f32	Initial value for the output state
Parameters (in-out):	None	

Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of a DT1 element.	

] ()

[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

]()

8.5.4.4.5 Get 'DT1' output – Type1

This routine can be realised using inline function.

[SWS_Mfl_00053] [

Service name:	Mfl_DT1Typ1Out_f32	
Syntax:	float32 Mfl_DT1Typ1Out_f32 (const Mfl_StateDT1Typ1_Type* const State_cpst)	
Service ID[hex]:	0x23	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Pointer to state structure
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return 'DT1' controller output value
Description:	This routine returns 'DT1' controller's output value	

] ()

[SWS_Mfl_00054]

Output value = State_cpst->Y1

]()

8.5.4.4.6 Get 'DT1' output – Type2

This routine can be realised using inline function.

[SWS_Mfl_00310] [

Service name:	Mfl_DT1Typ2Out_f32	
Syntax:	float32 Mfl_DT1Typ2Out_f32 (const Mfl_StateDT1Typ2_Type* const State_cpst)	
Service ID[hex]:	0xC2	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Pointer to state structure
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return 'DT1' controller output value
Description:	This routine returns 'DT1' controller's output value	

] ()

[SWS_Mfl_00311]

Output value = State_cpst->Y1

]()

8.5.4.5 Proportional & Differential controller

This routine is a combination of proportional & differential controller.

8.5.4.5.1 PD Controller

[SWS_Mfl_00055] [

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

] ()

[SWS_Mfl_00056]

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

This derives implementation:

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

]()

[SWS_Mfl_00057]

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

]()

[SWS_Mfl_00058]

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

State_cpst->X1 = X_f32

]()

8.5.4.5.2 PD Set State Value

This routine can be realised using inline function.

[SWS_Mfl_00059] [

Service name:	Mfl_PDSetState	
Syntax:	<pre>void Mfl_PDSetState(Mfl_StatePD_Type* const State_cpst, float32 X1_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0x2B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of a PD element.	

]()

[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

]()

8.5.4.5.3 Set 'PD' Parameters

This routine can be realised using inline function.

[SWS_Mfl_00062] [

Service name:	Mfl_PDSetParam	
Syntax:	<pre>void Mfl_PDSetParam(Mfl_ParamPD_Type* const Param_cpst, float32 K_f32, float32 Tv_f32)</pre>	
Service ID[hex]:	0x2C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	K_f32	Amplification factor
	Tv_f32	Lead time
Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to internal parameter structure
Return value:	None	
Description:	The routine sets the parameter structure of a PD element.	

] ()

[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

] ()

8.5.4.5.4 Get 'PD' output

This routine can be realised using inline function.

[SWS_Mfl_00066] [

Service name:	Mfl_PDOut_f32	
Syntax:	<pre>float32 Mfl_PDOut_f32(const Mfl_StatePD_Type* const State_cpst)</pre>	
Service ID[hex]:	0x2D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Pointer to state structure
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return 'PD' controller output value
Description:	This routine returns 'PD' controllers output value.	

] ()

[SWS_Mfl_00067]

Output value = State_cpst->Y1

I()

8.5.4.6 Integral component

This routine calculates Integration element.

8.5.4.6.1 'I' Controller

[SWS_Mfl_00068] [

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

] ()

[SWS_Mfl_00069]

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

This derives implementation:

$$\text{Output_value} = \text{State_cpst} \rightarrow Y1 + K_f32 * dT_f32 * \text{State_cpst} \rightarrow X1$$

I()

[SWS_Mfl_00070]

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

$$\text{State_cpst} \rightarrow Y1 = \text{Output_value}$$

$$\text{State_cpst} \rightarrow X1 = X_f32$$

I()

8.5.4.6.2 'I' Controller with limitation

[SWS_Mfl_00320] [

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

] ()

[SWS_Mfl_00321]

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

This derives implementation:

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

]()

[SWS_Mfl_00322]

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

]()

[SWS_Mfl_00323]

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

State_cpst->Y1 = Output_value

State_cpst->X1 = X_f32

]()

8.5.4.6.3 Set limits for controllers

[SWS_Mfl_00324] [

Service name:	Mfl_CtrlSetLimit
Syntax:	<pre>void Mfl_CtrlSetLimit(float32 Min f32,</pre>

	<pre>float32 Max_f32, Mfl_Limits_Type* const Limit_cpst)</pre>	
Service ID[hex]:	0x34	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Min_f32	Minimum limit
	Max_f32	Maximum limit
Parameters (in-out):	Limit_cpst	Pointer to limit structure
Parameters (out):	None	
Return value:	None	
Description:	Update limit structure	

] ()

[SWS_Mfl_00325]

Update limit structure

Limit_cpst->Min_C = Min_f32

Limit_cpst->Max_C = Max_f32

] ()

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

Replacement routine : Mfl_CtrlSetLimits "

[SWS_Mfl_00367] [

Service name:	Mfl_CtrlSetLimits	
Syntax:	<pre>void Mfl_CtrlSetLimits(Mfl_Limits_Type* const Limit_cpst, float32 Min_f32, float32 Max_f32)</pre>	
Service ID[hex]:	0xC9	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Min_f32	Minimum limit
	Max_f32	Maximum limit
Parameters (in-out):	Limit_cpst	Pointer to limit structure
Parameters (out):	None	
Return value:	None	
Description:	Update limit structure	

] ()

[SWS_Mfl_00368]

Update limit structure

Limit_cpst->Min_C = Min_f32

Limit_cpst->Max_C = Max_f32

] ()

8.5.4.6.4 Set 'I' State Value

This routine can be realised using inline function.

[SWS_Mfl_00071] [

Service name:	Mfl_ISetState	
Syntax:	<pre>void Mfl_ISetState(Mfl_StateI_Type* const State_cpst, float32 X1_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0x31	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of an I element.	

] ()

[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

] ()

8.5.4.6.5 Get 'I' output

This routine can be realised using inline function.

[SWS_Mfl_00074] [

Service name:	Mfl_IOut_f32	
Syntax:	<pre>float32 Mfl_IOut_f32(const Mfl_StateI_Type* const State_cpst)</pre>	
Service ID[hex]:	0x33	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State_cpst	Pointer to state structure
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return 'I' controller output value
Description:	This routine returns 'I' controllers output value.	

}]()

[SWS_Mfl_00075]

Output value = State_cpst->Y1

]()

8.5.4.7 Proportional & Integral controller

This routine is a combination of Proportional & Integral controller.

8.5.4.7.1 'PI' Controller – Type1 (Implicit type)

[SWS_Mfl_00076] [

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

}]()

[SWS_Mfl_00077]

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

This derives implementation:

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

]()

[SWS_Mfl_00078]

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

]()

[SWS_Mfl_00079]

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

State_cpst->X1 = X_f32

})();

8.5.4.7.2 'PI' Controller – Type1 with limitation (Implicit type)
[SWS_Mfl_00326] [

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

})();

[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]

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

]()

[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

l()

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

[SWS_Mfl_00331] [

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

l()

[SWS_Mfl_00332][

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

This derives implementation:

$$\text{Output_value} = \text{State_cpst} \rightarrow Y1 + (\text{Param_cpst} \rightarrow K_C * (1 + \text{Param_cpst} \rightarrow Tnrec_C * dT_f32) * X_f32) - (\text{Param_cpst} \rightarrow K_C * \text{State_cpst} \rightarrow X1)$$

l()

[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

l()

[SWS_Mfl_00334][

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

l()

[SWS_Mfl_00335]

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

State_cpst->X1 = X_f32

l()

8.5.4.7.5 Set 'PI' State Value

This routine can be realised using inline function.

[SWS_Mfl_00084] [

Service name:	Mfl_PISetState	
Syntax:	<pre>void Mfl_PISetState(Mfl_StatePI_Type* const State_cpst, float32 X1_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0x37	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of a PI element.	

l()

[SWS_Mfl_00085]

Initialisation of output state variable Y1.

State_cpst->Y1 = Y1_f32

l()

[SWS_Mfl_00086]

Initialisation of input state variable X1.

State_cpst->X1 = X1_f32

l()

8.5.4.7.6 Set 'PI' Parameters

This routine can be realised using inline function.

[SWS_Mfl_00087] [

Service name:	Mfl_PISetParam	
Syntax:	<pre>void Mfl_PISetParam(Mfl_ParamPI_Type* const Param_cpst, float32 K_f32, float32 Tnrec)</pre>	
Service ID[hex]:	0x38	
Sync/Async:	Synchronous	

Reentrancy:	Reentrant	
Parameters (in):	K_f32	Amplification factor
	Tnrec	Reciprocal follow-up time
Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to internal parameter structure
Return value:	None	
Description:	The routine sets the parameter structure of a PI element.	

] ()

[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

]()

8.5.4.7.7 Get 'PI' output

This routine can be realised using inline function.

[SWS_Mfl_00090] [

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

] ()

[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

8.5.4.8.1 'PID' Controller – Type1 (Implicit type)

[SWS_Mfl_00092] [

Service name:	Mfl_PIDTyp1Calc	
Syntax:	<pre>void Mfl_PIDTyp1Calc(float32 X_f32, Mfl_StatePID_Type* const State_cpst, const Mfl_ParamPID_Type* const Param_cpst, float32 dT_f32)</pre>	
Service ID[hex]:	0x3A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to the internal state structure.
Return value:	None	
Description:	This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation	

] ()

[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:

$$calc1 = Param_cpst \rightarrow K_C * (1 + t_val) * X_f32$$

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

$$State_cpst \rightarrow X1$$

$$calc3 = Param_cpst \rightarrow K_C * t_val * State_cpst \rightarrow X2$$

$$Output_value = State_cpst \rightarrow Y1 + calc1 - calc2 + calc3$$

 Where $t_val = Param_cpst \rightarrow T_v_C / dT_f32$

] ()

[SWS_Mfl_00094]

Calculated Output_value shall be stored to State_cpst->Y1.

$$State_cpst \rightarrow Y1 = Output_value$$

] ()

[SWS_Mfl_00095]

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

$$State_cpst \rightarrow X2 = State_cpst \rightarrow X1$$

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

$$State_cpst \rightarrow X1 = X_f32$$

] ()

8.5.4.8.2 'PID' Controller – Type1 with limitation (Implicit type)
[SWS_Mfl_00340] [

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

] ()

[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:

$$\text{calc1} = \text{Param_cpst} \rightarrow K_C * (1 + t_val) * X_f32$$

$$\text{calc2} = \text{Param_cpst} \rightarrow K_C * (1 - dT_f32 * \text{Param_cpst} \rightarrow T_{nrec_C} + 2 * t_val) * \text{State_cpst} \rightarrow X1$$

$$\text{calc3} = \text{Param_cpst} \rightarrow K_C * t_val * \text{State_cpst} \rightarrow X2$$

$$\text{Output_value} = \text{State_cpst} \rightarrow Y1 + \text{calc1} - \text{calc2} + \text{calc3}$$

$$\text{Where } t_val = \text{Param_cpst} \rightarrow T_v_C / dT_f32$$

$$\text{Where } t_val = \text{Param_cpst} \rightarrow T_v_C / dT_f32$$

] ()

[SWS_Mfl_00342]

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

$$\text{Output_value} = \text{Limit_cpst} \rightarrow \text{Min_C}$$

If (Output_value > Limit_cpst->Max_C) Then,

$$\text{Output_value} = \text{Limit_cpst} \rightarrow \text{Max_C}$$

] ()

[SWS_Mfl_00343]

Calculated Output_value shall be stored to State_cpst->Y1.

$$\text{State_cpst} \rightarrow Y1 = \text{Output_value}$$

] ()

[SWS_Mfl_00344]

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.

State_cpst->X1 = X_f32

})();

8.5.4.8.3 'PID' Controller – Type2 (Explicit type)

[SWS_Mfl_00096] [

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

})();

[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->T_v_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

]()

8.5.4.8.4 'PID' Controller – Type2 with limitation (Explicit type)

[SWS_Mfl_00345] [

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

]()

[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:

$$\text{calc1} = \text{Param_cpst} \rightarrow K_C * (1 + dT_f32 * \text{Param_cpst} \rightarrow Tnrec_C + t_val) * X_f32$$

$$\text{calc2} = \text{Param_cpst} \rightarrow K_C * (1 + 2 * t_val) * \text{State_cpst} \rightarrow X1$$

$$\text{calc3} = \text{Param_cpst} \rightarrow K_C * t_val * \text{State_cpst} \rightarrow X2$$

$$\text{Output_value} = \text{State_cpst} \rightarrow Y1 + \text{calc1} - \text{calc2} + \text{calc3}$$

Where t_val = Param_cpst->Tv_C / dT_f32

]()

[SWS_Mfl_00347]

Limit output value with maximum and minimum controller limits.

If (Output_value < Limit_cpst->Min_C) Then,

Output_value = Limit_cpst->Min_C

If (Output_value > Limit_cpst->Max_C) Then,

Output_value = Limit_cpst->Max_C

]()

[SWS_Mfl_00348]

Calculated Output_value shall be stored to State_cpst->Y1.

State_cpst->Y1 = Output_value

]()

[SWS_Mfl_00349]

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

State_cpst->X2 = State_cpst->X1

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

State_cpst->X1 = X_f32

]()

8.5.4.8.5 Set 'PID' State Value

This routine can be realised using inline function.

[SWS_Mfl_00100] [

Service name:	Mfl_PIDSetState	
Syntax:	<pre>void Mfl_PIDSetState(Mfl_StatePID_Type* const State_cpst, float32 X1_f32, float32 X2_f32, float32 Y1_f32)</pre>	
Service ID[hex]:	0x3C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	X2_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to internal state structure
Return value:	None	
Description:	The routine initialises internal state variables of a PID element.	

]()

[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

]()

8.5.4.8.6 Set 'PID' Parameters

This routine can be realised using inline function.

[SWS_Mfl_00103] [

Service name:	Mfl_PIDSetParam	
Syntax:	<pre>void Mfl_PIDSetParam(Mfl_ParamPID_Type* const Param_cpst, float32 K_f32, float32 Tv_f32, float32 Tnrec_f32)</pre>	
Service ID[hex]:	0x3D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	K_f32	Amplification factor
	Tv_f32	Lead Time
	Tnrec_f32	Reciprocal follow-up timer
Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to internal parameter structure
Return value:	None	
Description:	The routine sets the parameter structure of a PID element.	

] ()

[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

]()

8.5.4.8.7 Get 'PID' output

This routine can be realised using inline function.

[SWS_Mfl_00107] [

Service name:	Mfl_PIDOut_f32	
Syntax:	<pre>float32 Mfl_PIDOut_f32(const Mfl_StatePID_Type* const State_cpst</pre>	

)
Service ID[hex]:	0x3E
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	State_cpst Pointer to state structure
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 Return 'PID' controller output value
Description:	This routine returns 'PID' controllers output value.

] ()

[SWS_Mfl_00108]

Output value = State_cpst->Y1

]()

8.5.5 Magnitude and Sign

[SWS_Mfl_00110]

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

] ()

[SWS_Mfl_00111]

ValAbs = | ValValue |

]()

[SWS_Mfl_00112]

Service name:	Mfl_Sign_f32
Syntax:	sint8 Mfl_Sign_f32(float32 ValValue)
Service ID[hex]:	0x41
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	ValValue Floating-point operand.
Parameters (in-out):	None
Parameters (out):	None

Return value:	sint8	Integer representing the sign of the operand.
Description:	Returns the sign of the argument (ValSign), determined according to the following equation.	

] ()

[SWS_Mfl_00113]

ValSign = 1, ValValue > 0.0

]()

[SWS_Mfl_00114]

ValSign = 0, ValValue == 0.0

]()

[SWS_Mfl_00115]

ValSign = -1, ValValue < 0.0

]()

8.5.6 Limiting

[SWS_Mfl_00116] [

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

] ()

[SWS_Mfl_00117]

ValMax = ValValue1, ValValue1 ≥ ValValue2

ValMax = ValValue2, ValValue1 < ValValue2

]()

[SWS_Mfl_00118] [

Service name:	Mfl_Min_f32	
Syntax:	float32 Mfl_Min_f32(float32 Value1, float32 Value2)	

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

] ()

[SWS_Mfl_00119]

Min = Value1, Value1 ≤ Value2

Min = Value2, Value1 > Value2

]()

[SWS_Mfl_00120] [

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

] ()

[SWS_Mfl_00121]

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

Result = oldval + maxdif

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

Result = oldval - maxdif

else

Result = newval

]()

[SWS_Mfl_00122] [

Service name:	Mfl_Limit_f32	
Syntax:	<pre>float32 Mfl_Limit_f32(float32 val, float32 lowLim, float32 upLim)</pre>	
Service ID[hex]:	0x48	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	val	Quantity to be bounded.
	lowLim	Lower bound.
	upLim	Upper bound
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Limited value.
Description:	Returns the bounded value (newVal), determined according to the following equation.	

] ()

[SWS_Mfl_00123]

newVal = lowLim, $val \leq lowLim$

newVal = upLim, $val \geq upLim$

newVal = val, $lowLim < val < upLim$

]()

8.5.7 Logarithms and Exponentials

[SWS_Mfl_00130] [

Service name:	Mfl_Pow_f32	
Syntax:	<pre>float32 Mfl_Pow_f32(float32 ValBase, float32 ValExp)</pre>	
Service ID[hex]:	0x50	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ValBase	Base to be raised to an exponent. Valid range: ValBase > 0.0
	ValExp	Exponent by which to raise the base.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	ValBase raised to ValExp power.
Description:		

] ()

[SWS_Mfl_00131]

ValResult = ValBase^{ValExp}

]()

[SWS_Mfl_00132]

If ValExp = 0, and ValBase = 0, ValResult = 1, ($0^0 = 1$)
 If ValBase = 0 and ValExp <> 0, ValResult = 0, ($0^{\text{ValExp}} = 0$)
]()

[SWS_Mfl_00133]

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

[SWS_Mfl_00135]

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

]()

[SWS_Mfl_00136]

ValSqrt = ValValue^{1/2}
]()

[SWS_Mfl_00137]

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

[SWS_Mfl_00140]

Service name:	Mfl_Exp_f32
Syntax:	float32 Mfl_Exp_f32(float32 ValValue)
Service ID[hex]:	0x53
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	ValValue Floating-point operand.
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 e raised to ValValue power
Description:	Returns the exponential of the operand (ValExp), determined according to the

	following equation.
--	---------------------

l()

[SWS_Mfl_00141]

ValExp = eValValue

l()

[SWS_Mfl_00142]

ValValue Range shall be [-24PI, +24PI]

l()

[SWS_Mfl_00145]

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

l()

[SWS_Mfl_00146]

ValLog = loge(ValValue)

l()

[SWS_Mfl_00147]

ValValue shall be passed as > 0 value.

l()

8.5.8 Trigonometry

[SWS_Mfl_00150]

Service name:	Mfl_Sin_f32	
Syntax:	float32 Mfl_Sin_f32 (float32 value)	
Service ID[hex]:	0x55	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	

Parameters (in):	value	angle in radians
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	result = sine (value)
Description:	Calculates the sine of the argument.	

l()

[SWS_Mfl_00151]

Result: result = sine (value)

l()

[SWS_Mfl_00152]

Range of value shall be [-24PI, +24PI]

l()

[SWS_Mfl_00155]

Service name:	Mfl_Cos_f32	
Syntax:	float32 Mfl_Cos_f32 (float32 value)	
Service ID[hex]:	0x56	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	value	angle in radians
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	result = cosine (value)
Description:	Calculates the cosine of the argument.	

l()

[SWS_Mfl_00156]

Result: result = cosine (value)

l()

[SWS_Mfl_00157]

Range of value shall be [-24PI, +24PI]

l()

[SWS_Mfl_00160]

Service name:	Mfl_Tan_f32	
Syntax:	float32 Mfl_Tan_f32 (float32 value)	
Service ID[hex]:	0x57	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	

Parameters (in):	value	angle in radians
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	result = tangent(value)
Description:	Calculates the tangent of the argument.	

l()

[SWS_Mfl_00161]

Result: result = tangent(value)

l()

[SWS_Mfl_00163]

Range of the value shall be [-24PI, +24PI]

l()

[SWS_Mfl_00165]

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

l()

[SWS_Mfl_00167]

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

l()

[SWS_Mfl_00168]

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

l()

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

Replacement routine : Mfl_ArcSin_f32"

[SWS_Mfl_00350]

Service name:	Mfl_ArcSin_f32	
Syntax:	float32 Mfl_ArcSin_f32 (float32 value)	

)
Service ID[hex]:	0xBC
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	value The value whose arc sine is to be returned
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 The arc sine of the argument, in radians
Description:	Returns the arc sine of an angle, in the range of $-\pi/2$ through $\pi/2$.

})();

[SWS_Mfl_00352]

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

})();

[SWS_Mfl_00353]

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

})();

[SWS_Mfl_00170]

Service name:	Mfl_arcCos_f32
Syntax:	float32 Mfl_arcCos_f32(float32 value)
Service ID[hex]:	0x59
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	value The value whose arc cosine is to be returned
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 The arc cosine of the argument, in radians
Description:	Returns the arc cosine of an angle, in the range of 0.0 through π .

})();

[SWS_Mfl_00172]

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

})();

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

Replacement routine : Mfl_ArcCos_f32"

[SWS_Mfl_00354]

Service name:	Mfl_ArcCos_f32
Syntax:	float32 Mfl_ArcCos_f32(float32 value)

)
Service ID[hex]:	0xBD
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	value The value whose arc cosine is to be returned
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 The arc cosine of the argument, in radians
Description:	Returns the arc cosine of an angle, in the range of 0.0 through pi.

l()

[SWS_Mfl_00356]

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

l()

[SWS_Mfl_00175]

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

l()

[SWS_Mfl_00177]

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

l()

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

Replacement routine : Mfl_ArcTan_f32"

[SWS_Mfl_00357]

Service name:	Mfl_ArcTan_f32
Syntax:	float32 Mfl_ArcTan_f32 (float32 value)
Service ID[hex]:	0xBE
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	value The value whose arc tan is to be returned.

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

})();

[SWS_Mfl_00359]

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

})();

[SWS_Mfl_00180]

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

})();

[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]

Service name:	Mfl_ArcTan2_f32
Syntax:	float32 Mfl_ArcTan2_f32(float32 X1_f32, float32 X2_f32)

	float32 X1_f32, float32 X2_f32)
Service ID[hex]:	0xBF
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	X1_f32 Input value 1 X2_f32 Input value 2
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 Returns arctan for inputs X1_f32 & X2_f32
Description:	Returns the arc tangent of an angle, in the range of [-pi to pi]

|()

[SWS_Mfl_00362]

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

|()

[SWS_Mfl_00363]

$Z = X2_f32 / X1_f32$

if ($Z > 1$) Then

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

if ($Z < 1$) Then

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

|()

8.5.9 Average

[SWS_Mfl_00190] [

Service name:	Mfl_Average_f32_f32
Syntax:	float32 Mfl_Average_f32_f32 (float32 value1, float32 value2)
Service ID[hex]:	0x61
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	value1 Input value1 value2 Input value2
Parameters (in-out):	None
Parameters (out):	None
Return value:	float32 Return value of the function
Description:	The routine returns average value.

|()

[SWS_Mfl_00191]

Output = $(Value1 + Value2) / 2$

|()

8.5.10 Array Average

[SWS_Mfl_00192] [

Service name:	Mfl_ArrayAverage_f32_f32	
Syntax:	float32 Mfl_ArrayAverage_f32_f32 (float32* Array, uint32 Count)	
Service ID[hex]:	0x65	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Array	Pointer to an array
	Count	Number of array elements
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return value of the function
Description:	The routine returns average value of an array.	

] ()

[SWS_Mfl_00193]

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

]()

8.5.11 Hypotenuse

[SWS_Mfl_00195] [

Service name:	Mfl_Hypot_f32f32_f32	
Syntax:	float32 Mfl_Hypot_f32f32_f32 (float32 x_value, float32 y_value)	
Service ID[hex]:	0x70	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	x_value	First argument Recommended input range: [-24PI, +24PI]
	y_value	Second argument Recommended input range [-24PI, +24PI]
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return value of the function
Description:	This service computes the length of a vector	

] ()

[SWS_Mfl_00196]

This service computes the length of a vector:

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

]()

8.5.12 Ramp routines

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

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

- Mfl_RampCalcSwitch
- Mfl_RampCalcJump
- Mfl_RampCalc
- Mfl_RampOut_f32

Structure definition for function argument

[SWS_Mfl_00200] [

Name:	Mfl_ParamRamp_Type		
Type:	Structure		
Element:	float32	SlopePos_f32	Positive slope for ramp in absolute value
	float32	SlopeNeg_f32	Negative slope for ramp in absolute value
Description:	Structure definition for Ramp routine		

]()

[SWS_Mfl_00833] [

Name:	Mfl_StateRamp_Type		
Type:	Structure		
Element:	float32	State_f32	State of the ramp
	sint8	Dir_s8	Ramp direction
	sint8	Switch_s8	Position of switch
Description:	Structure definition for Ramp routine		

]()

8.5.12.1 Ramp routine

[SWS_Mfl_00201] [

Service name:	Mfl_RampCalc		
Syntax:	<pre>void Mfl_RampCalc(float32 X_f32, Mfl_StateRamp_Type* const State_cpst, const Mfl_ParamRamp_Type* const Param_cpcst, float32 dT_f32)</pre>		
Service ID[hex]:	0x90		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters (in):	X_f32	Target value for the ramp to reach	
	Param_cpcst	Pointer to parameter structure	

	dT_f32	Sample Time
Parameters (in-out):	State_cpst	Pointer to state structure
Parameters (out):	None	
Return value:	None	
Description:	The ramp output value increases or decreases a value with slope * dT_f32 depending if (State_cpst->State_f32 > X_f32) or (State_cpst->State_f32 < X_f32).	

] ()

[SWS_Mfl_00835]

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

]()

[SWS_Mfl_00202]

If ramp direction is rising then ramp increases a value with slope * dT_f32
 if (State_cpst->Dir_s8 == RISING)
 State_cpst->State_f32 = State_cpst->State_f32 + (Param_cpcst->SlopePos_f32 * dT_f32)

]()

[SWS_Mfl_00203]

If ramp direction is falling then ramp decreases a value with slope * dT_f32
 if (State_cpst->Dir_s8 == FALLING)
 State_cpst->State_f32 = State_cpst->State_f32 - (Param_cpcst->SlopeNeg_f32 * dT_f32)

]()

[SWS_Mfl_00204]

Direction of the ramp is stored so that a change of the target can be recognized and the output will follow immediately to the new target value.
 State_cpst->Dir_s8 states are: RISING, FALLING, END.

]()

[SWS_Mfl_00205]

Comparison of State and Target decides ramp direction.
 If(State_cpst->State_f32 > X_f32) then State_cpst->Dir_s8 = FALLING
 If(State_cpst->State_f32 < X_f32) then State_cpst->Dir_s8 = RISING
 If(State_cpst->State_f32 == X_f32) then State_cpst->Dir_s8 = END

]()

8.5.12.2 Ramp Initialisation

[SWS_Mfl_00208]

Service name:	Mfl_RampInitState
Syntax:	void Mfl_RampInitState(Mfl_StateRamp Type* const State_cpst,

	float32 Val_f32	
)	
Service ID[hex]:	0x91	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Val_f32	Initial value for state variable
Parameters (in-out):	State_cpst	Pointer to the state structure
Parameters (out):	None	
Return value:	None	
Description:	Initializes the state, direction and switch parameters for the ramp.	

] ()

[SWS_Mfl_00209]

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

State_cpst->Dir_s8 = END

]()

For example:

ramp direction states: RISING = 1, FALLING = -1, END = 0

[SWS_Mfl_00275]

Initialisation of state variable

State_cpst ->State_f32 = Val_f32

]()

[SWS_Mfl_00276]

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

State_cpst->Switch_s8 = OFF

]()

For example:

switch states: TARGET_A = 1, TARGET_B = -1, OFF = 0

8.5.12.3 Ramp Set Slope

[SWS_Mfl_00210] [

Service name:	Mfl_RampSetParam	
Syntax:	<pre>void Mfl_RampSetParam(Mfl_ParamRamp_Type* const Param_cpst, float32 SlopePosVal_f32, float32 SlopeNegVal_f32)</pre>	
Service ID[hex]:	0x92	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SlopePosVal_f32	Positive slope value
	SlopeNegVal_f32	Negative slope value

Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to parameter structure
Return value:	None	
Description:	Sets the slope parameter for the ramp provided by the structure Mfl_ParamRamp_Type.	

] ()

[SWS_Mfl_00211]

Sets positive and negative ramp slopes.

Param_cpst->SlopePos_f32 = SlopePosVal_f32

Param_cpst->SlopeNeg_f32 = SlopeNegVal_f32

]()

8.5.12.4 Ramp Out routine

[SWS_Mfl_00212] [

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

] ()

[SWS_Mfl_00213]

Return Value = State_cpcst->State_f32

]()

8.5.12.5 Ramp Jump routine

[SWS_Mfl_00214] [

Service name:	Mfl_RampCalcJump	
Syntax:	void Mfl_RampCalcJump (float32 X_f32, Mfl_StateRamp_Type* const State_cpst)	
Service ID[hex]:	0x94	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Target value for ramp to jump

Parameters (in-out):	State_cpst	Pointer to the state value
Parameters (out):	None	
Return value:	None	
Description:	<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>	

] ()

[SWS_Mfl_00215]

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

State_cpst->State_f32 = X_f32

State_cpst->Dir_s8 = END

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

]()

8.5.12.6 Ramp switch routine

[SWS_Mfl_00216] [

Service name:	Mfl_RampCalcSwitch_f32	
Syntax:	<pre>float32 Mfl_RampCalcSwitch_f32 (float32 Xa_f32, float32 Xb_f32, Mfl_StateRamp_Type* const State_cpst, const Mfl_ParamRamp_Type* const Param_cpcst, float32 dT_f32)</pre>	
Service ID[hex]:	0x95	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Xa_f32	Target value for the ramp to reach if switch is in position 'A'
	Xb_f32	Target value for the ramp to reach if switch is in position 'B'
	Param_cpcst	Pointer to the parameter structure which contains the positive and negative slope of the ramp
	dT_f32	Sample Time
Parameters (in-out):	State_cpst	Pointer to actual value of the ramp
Parameters (out):	None	
Return value:	float32	Returns the actual state of the ramp
Description:	This routine switches ramp between two target values based on the Switch value.	

] ()

[SWS_Mfl_00217]

Switch decides target to select.

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

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

] ()

[SWS_Mfl_00218]

State_cpst->Dir_s8 holds direction information

Ramp direction status: RISING, FALLING, END

] ()

[SWS_Mfl_00219]

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

if (State_cpst->Dir_s8 == RISING)

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

else if (State_cpst->Dir_s8 == FALLING)

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

else if (State_cpst->Dir_s8 == END)

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

] ()

[SWS_Mfl_00220]

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

State_cpst->Dir_s8 == END

] ()

[SWS_Mfl_00221]

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

If (State_cpst->State_f32 == target value)

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

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

] ()

[SWS_Mfl_00222]

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

] ()

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

Replacement routine : Mfl_RampCalcSwitch "

[SWS_Mfl_00369]

Service name:	Mfl_RampCalcSwitch
----------------------	--------------------

Syntax:	<pre>float32 Mfl_RampCalcSwitch(float32 Xa_f32, float32 Xb_f32, boolean Switch, Mfl_StateRamp_Type* const State_cpst)</pre>	
Service ID[hex]:	0xCA	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Xa_f32	Target value for the ramp to reach if switch is in position 'A'
	Xb_f32	Target value for the ramp to reach if switch is in position 'B'
	Switch	Switch to decide target value
Parameters (in-out):	State_cpst	Pointer to StateRamp structure
Parameters (out):	None	
Return value:	float32	Returns the selected target value
Description:	This routine switches between two target values for a ramp service based on a Switch parameter.	

l()

[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

l()

[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.

l()

[SWS_Mfl_00372]

Mfl_RampCalcSwitch has to be called before Mfl_RampCalc routine

l()

8.5.12.7 Get Ramp Switch position

[SWS_Mfl_00223] [

Service name:	Mfl_RampGetSwitchPos
Syntax:	<pre>boolean Mfl_RampGetSwitchPos(Mfl_StateRamp_Type* const State_cpst</pre>

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

] ()

[SWS_Mfl_00224]

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

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

] ()

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

8.5.12.8 Check Ramp Activity

[SWS_Mfl_00225] [

Service name:	Mfl_RampCheckActivity
Syntax:	boolean Mfl_RampCheckActivity(Mfl_StateRamp_Type* const State_cpst)
Service ID[hex]:	0x97
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	State_cpst Pointer to the state structure
Parameters (in-out):	None
Parameters (out):	None
Return value:	boolean return value TRUE or FALSE
Description:	This routine checks the status of the ramp and returns a TRUE if the ramp is active, otherwise it returns FALSE.

] ()

[SWS_Mfl_00226]

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

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

] ()

8.5.13 Hysteresis routines

8.5.13.1 Hysteresis center half delta

[SWS_Mfl_00236] [

Service name:	Mfl_HystCenterHalfDelta_<InTypeMn>_<OutTypeMn>
Syntax:	boolean Mfl_HystCenterHalfDelta_<InTypeMn>_<OutTypeMn> (<InType> X, <InType> center, <InType> halfDelta, uint8* State)
Service ID[hex]:	0xA0
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	X Input value
	center Center of hysteresis range
	halfDelta Half width of hysteresis range
Parameters (in-out):	State Pointer to state value
Parameters (out):	None
Return value:	boolean Returns TRUE or FALSE depending of input value and state value
Description:	Hysteresis with center and left and right side halfDelta switching point.

] ()

[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

] ()

[SWS_Mfl_00240]

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0xA0	boolean Mfl_HystCenterHalfDelta_f32_u8(float32, float32, float32, uint8 *)

] ()

8.5.13.2 Hysteresis left right

[SWS_Mfl_00241] [

Service name:	Mfl_HystLeftRight_<InTypeMn>_<OutTypeMn>
Syntax:	boolean Mfl_HystLeftRight_<InTypeMn>_<OutTypeMn> (<InTypeMn> X, <OutTypeMn> State)

	<InType> X, <InType> Lsp, <InType> Rsp, uint8* State)	
Service ID[hex]:	0xA3	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X	Input value
	Lsp	Left switching point
	Rsp	Right switching point
Parameters (in-out):	State	Pointer to state value
Parameters (out):	None	
Return value:	boolean	Returns TRUE or FALSE depending of input value and state value
Description:	Hysteresis with left and right switching point.	

] ()

[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

] ()

[SWS_Mfl_00245]

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0xA3	boolean Mfl_HystLeftRight_f32_u8 (float32, float32, float32, uint8 *)

] ()

8.5.13.3 Hysteresis delta right

[SWS_Mfl_00246] [

Service name:	Mfl_HystDeltaRight_<InTypeMn>_<OutTypeMn>
Syntax:	boolean Mfl_HystDeltaRight_<InTypeMn>_<OutTypeMn> (<InType> X, <InType> Delta, <InType> Rsp, uint8* State)
Service ID[hex]:	0xA5
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	X	Input value
	Delta	Left switching point = rsp - delta
	Rsp	Right switching point
	State	Pointer to state value
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	boolean	Returns TRUE or FALSE depending of input value and state value
Description:	Hysteresis with right switching point and delta to left switching point	

] ()

[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.

]()

[SWS_Mfl_00250]

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0xA5	boolean Mfl_HystDeltaRight_f32_u8 (float32, float32, float32, uint8 *)

]()

8.5.13.4 Hysteresis left delta

[SWS_Mfl_00251] [

Service name:	Mfl_HystLeftDelta_<InTypeMn>_<OutTypeMn>	
Syntax:	boolean Mfl_HystLeftDelta_<InTypeMn>_<OutTypeMn> (<InType> X, <InType> Lsp, <InType> Delta, uint8* State)	
Service ID[hex]:	0xA7	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X	Input value
	Lsp	Left switching point
	Delta	Right switching point = lsp + delta

Parameters (in-out):	State	Pointer to state value
Parameters (out):	None	
Return value:	boolean	Returns TRUE or FALSE depending of input value and state value
Description:	Hysteresis with left switching point and delta to right switching point.	

] ()

[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.

]()

[SWS_Mfl_00255]

Here is the list of implemented functions.

Function ID[hex]	Function prototype
0xA7	boolean Mfl_HystLeftDelta_f32_u8 (float32, float32, float32, uint8 *)

]()

8.5.14 Mfl_DeadTime

[SWS_Mfl_00256] [

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

] ()

[SWS_Mfl_00257]

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

] ()

[SWS_Mfl_00258]

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

] ()

Structure definition for function argument

[SWS_Mfl_00259]

Name:	Mfl_DeadTimeParam Type		
Type:	Structure		
Element:	float32	dsintStatic	Time since the last pack was written
	float32	*lszStatic	Pointer to actual buffer position
	float32	*dtbufBegStatic	Pointer to begin of buffer
	float32	*dtbufEndStatic	Pointer to end of buffer
Description:	Structure definition for Dead Time routine		

] ()

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

8.5.15 Debounce routines

8.5.15.1 Mfl_Debounce

[SWS_Mfl_00260] [

Service name:	Mfl_Debounce_u8_u8	
Syntax:	<pre>boolean Mfl_Debounce_u8_u8 (boolean X, Mfl_DebounceState_Type* State, Mfl_DebounceParam_Type* Param, float32 dT)</pre>	
Service ID[hex]:	0xB0	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X	Input value
	Param	Pointer to state structure of type Mfl_DebounceState_Type
	dT	Sample Time
Parameters (in-out):	State	Pointer to structure for debouncing state variables
Parameters (out):	None	
Return value:	boolean	Returns the debounced input value
Description:	This routine debounces a digital input signal and returns the state of the signal as	

	a boolean value.
--	------------------

] ()

[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]

Name:	Mfl_DebounceParam Type		
Type:	Structure		
Element:	float32	TimeHighLow	Time for a High to Low transition, given in 10ms steps
	float32	TimeLowHigh	Time for a Low to High transition, given in 10ms steps
Description:	Structure definition for Debouncing parameters		

]()

[SWS_Mfl_00834]

Name:	Mfl_DebounceState Type		
Type:	Structure		
Element:	boolean	XOld	Old input value from last call
	float32	Timer	Timer for internal state
Description:	Structure definition for Debouncing state variables		

]()

8.5.15.2 Mfl_DebounceInit

[SWS_Mfl_00266] [

Service name:	Mfl_DebounceInit	
Syntax:	<pre>void Mfl_DebounceInit(Mfl_DebounceState_Type* State, boolean X)</pre>	
Service ID[hex]:	0xB1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State	Pointer to structure for debouncing state variables
	X	Initial value for the input state
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	This routine call shall stop the debouncing timer.	

] ()

[SWS_Mfl_00267][

State->Timer = 0

]()

[SWS_Mfl_00268][

Sets the input state to the given init value.

State->XOld = X

]()

8.5.15.3 Mfl_DebounceSetParam

[SWS_Mfl_00269] [

Service name:	Mfl_DebounceSetparam	
Syntax:	<pre>void Mfl_DebounceSetparam(Mfl_DebounceParam_Type* Param, float32 THighLow, float32 TLowHigh)</pre>	
Service ID[hex]:	0xB2	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (in-out):	None	
Parameters (out):	Param	Pointer to state structure of type Mfl_DebounceParam_Type
Return value:	None	

Description:	This routine sets timing parameters, time for high to low transition and time for low to high for debouncing.
---------------------	---

] ()

[SWS_Mfl_00270]

Param-> TimeHighLow = THighLow

Param-> TimeLowHigh = TLowHigh

]()

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

Replacement routine : Mfl_DebounceSetParam "

[SWS_Mfl_00365] [

Service name:	Mfl_DebounceSetParam	
Syntax:	<pre>void Mfl_DebounceSetParam(Mfl_DebounceParam_Type* Param, float32 THighLow, float32 TLowHigh)</pre>	
Service ID[hex]:	0xC8	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (in-out):	None	
Parameters (out):	Param	Pointer to state structure of type Mfl_DebounceParam_Type
Return value:	None	
Description:	This routine sets timing parameters, time for high to low transition and time for low to high for debouncing.	

] ()

[SWS_Mfl_00366]

Param-> TimeHighLow = THighLow

Param-> TimeLowHigh = TLowHigh

]()

8.5.16 Ascending Sort Routine

[SWS_Mfl_00271] [

Service name:	Mfl_SortAscend_f32	
Syntax:	<pre>void Mfl_SortAscend_f32(float32* Array, uint16 Num)</pre>	
Service ID[hex]:	0xB5	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Num	Size of an data array
Parameters (in-out):	Array	Pointer to an data array
Parameters (out):	None	
Return value:	None	
Description:	The sorting algorithm modifies the given input array in ascending order & returns sorted array result via pointer	

] ()

Example for signed array:

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

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

8.5.17 Descending Sort Routine

[SWS_Mfl_00273] [

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

] ()

Example for signed array:

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

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

8.5.18 Median sort routine

[SWS_Mfl_00285] [

Service name:	Mfl_MedianSort_f32_f32	
Syntax:	<pre>float32 Mfl_MedianSort_f32_f32 (float32* const Array, uint8 N)</pre>	
Service ID[hex]:	0xBB	
Sync/Async:	Synchronous	

Reentrancy:	Reentrant	
Parameters (in):	Array	Pointer to an array
	N	Size of an array
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Return value of the function
Description:	This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call.	

↓ ()

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.

↓()

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Mfl_GetVersionInfo

[SWS_Mfl_00815] [

Service name:	Mfl_GetVersionInfo
Syntax:	void Mfl_GetVersionInfo(Std_VersionInfoType* versioninfo

)
Service ID[hex]:	0xff
Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	versioninfo Pointer to where to store the version information of this module. Format according [BSW00321]
Return value:	None
Description:	Returns the version information of this library.

The version information of a BSW module generally contains:

Module Id

Vendor Id

Vendor specific version numbers (SRS_BSW_00407).] (SRS_BSW_00407, SRS_BSW_00003, SRS_BSW_00318, SRS_BSW_00321)

[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 Call-back 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 [3] 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 [1].] (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.

11 Not applicable requirements

[SWS_Mfl_00822]

These requirements are not applicable to this specification.

]()