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

Document Version	1.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
09.12.2011	1.2.0	AUTOSAR Administration	• Removal of 'Accumulator routine' • Revised 'Trigonometric routines' names • Added 'Median Sort Routines'
15.11.2010	1.1.0	AUTOSAR Administration	• Introduction of additional LIMITED Functions for controllers • Ramp functions optimised for effective usage • Separation of DT1 Type 1 and Type 2 Controller functions • Introduction of additional approximative function for calculatio of TeQ
07.12.2009	1.0.0	AUTOSAR Administration	Initial Release

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 ... 12
7.1 Error classification ... 12
7.2 Error detection ... 12
7.3 Error notification ... 12
7.4 Initialization and shutdown ... 12
7.5 Using Library API ... 12
7.6 library implementation ... 13
8 Routine specification ... 15
8.1 Imported types ... 15
8.2 Type definitions ... 15
8.3 Comment about rounding ... 15
8.4 Comment about routines optimized for target ... 16
8.5 Routine definitions ... 17
8.5.1 Floating point to Fixed-Point Conversion ... 17
8.5.2 Fixed-Point to Floating-Point Conversion ... 17
8.5.3 Rounding ... 18
8.5.4 Controller routines ... 20
8.5.4.1 Structure definitions for controller routines ... 20
8.5.4.2 Proportional Controller ... 22
8.5.4.2.1 ‘P’ Controller ... 22
8.5.4.2.2 Get ‘P’ output ... 23
8.5.4.3 Proportional controller with first order time constant ... 23
8.5.4.3.1 ‘PT1’ Controller ... 23
8.5.4.3.2 ‘PT1’ Set State Value ... 24
8.5.4.3.3 Calculate time equivalent Value ... 24
8.5.4.3.4 Calculate an approximate time equivalent Value ... 25
8.5.4.3.5 Get ‘PT1’ output ... 25
8.5.4.4 Differential component with time delay : DT1 ... 26
8.5.4.4.1 ‘DT1’ Controller - Type1 ... 26
8.5.4.4.2 ‘DT1’ Controller - Type2 ... 27
8.5.4.4.3 Set ‘DT1’ State Value – Type1 ... 28

8.5.4.4.4 Set 'DT1' State Value – Type2 ... 28
8.5.4.4.5 Get 'DT1' output – Type1 ... 29
8.5.4.4.6 Get 'DT1' output – Type2 ... 29
8.5.4.5 Proportional & Differential controller ... 30
8.5.4.5.1 PD Controller ... 30
8.5.4.5.2 PD Set State Value ... 30
8.5.4.5.3 Set 'PD' Parameters ... 31
8.5.4.5.4 Get 'PD' output ... 32
8.5.4.6 Integral component ... 32
8.5.4.6.1 'I' Controller ... 32
8.5.4.6.2 'I' Controller with limitation ... 33
8.5.4.6.3 Set limits for controllers ... 34
8.5.4.6.4 Set 'I' State Value ... 34
8.5.4.6.5 Get 'I' output ... 35
8.5.4.7 Proportional & Integral controller ... 35
8.5.4.7.1 'PI' Controller – Type1 (Implicit type) ... 36
8.5.4.7.2 'PI' Controller – Type1 with limitation (Implicit type) ... 36
8.5.4.7.3 'PI' Controller – Type2 (Explicit type) ... 37
8.5.4.7.4 'PI' Controller – Type2 with limitation (Explicit type) ... 38
8.5.4.7.5 Set 'PI' State Value ... 39
8.5.4.7.6 Set 'PI' Parameters ... 39
8.5.4.7.7 Get 'PI' output ... 40
8.5.4.8 Proportional, Integral & Differential controller ... 40
8.5.4.8.1 'PID' Controller – Type1 (Implicit type) ... 40
8.5.4.8.2 'PID' Controller – Type1 with limitation (Implicit type) ... 41
8.5.4.8.3 'PID' Controller – Type2 (Explicit type) ... 42
8.5.4.8.4 'PID' Controller – Type2 with limitation (Explicit type) ... 43
8.5.4.8.5 Set 'PID' State Value ... 44
8.5.4.8.6 Set 'PID' Parameters ... 44
8.5.4.8.7 Get 'PID' output ... 45
8.5.5 Magnitude and Sign ... 45
8.5.6 Limiting ... 46
8.5.7 Logarithms and Exponentials ... 48
8.5.8 Trigonometry ... 50
8.5.9 Average ... 55
8.5.10 Array Average ... 56
8.5.11 Hypotenuse ... 56
8.5.12 Ramp routines ... 57
8.5.12.1 Ramp routine ... 57
8.5.12.2 Ramp Initialisation ... 58
8.5.12.3 Ramp Set Slope ... 59
8.5.12.4 Ramp Out routine ... 59
8.5.12.5 Ramp Jump routine ... 60
8.5.12.6 Ramp switch routine ... 61
8.5.12.7 Get Ramp Switch position ... 63
8.5.12.8 Check Ramp Activity ... 63
8.5.13 Hysteresis routines ... 64
8.5.13.1 Hysteresis center half delta ... 64
8.5.13.2 Hysteresis left right ... 64
8.5.13.3 Hysteresis delta right ... 65

8.5.13.4 Hysteresis left delta ... 66
8.5.14 Mfl_DeadTime ... 66
8.5.15 Debounce routines ... 67
8.5.15.1 Mfl_Debounce ... 67
8.5.15.2 Mfl_DebounceInit ... 68
8.5.15.3 Mfl_DebounceSetParam ... 69
8.5.16 Ascending Sort Routine ... 70
8.5.17 Descending Sort Routine ... 70
8.5.18 Median sort routine ... 71
8.6 Examples of use of functions ... 72
8.7 Version API ... 72
8.7.1 Mfl_GetVersionInfo ... 72
8.8 Call-back notifications ... 72
8.9 Scheduled functions ... 72
8.10 Expected Interfaces ... 73
8.10.1 Mandatory Interfaces ... 73
8.10.2 Optional Interfaces ... 73
8.10.3 Configurable interfaces ... 73
9 Sequence diagrams ... 74
10 Configuration specification ... 75
10.1 Published Information ... 75
10.2 Configuration option ... 75
11 Not applicable requirements ... 76

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.

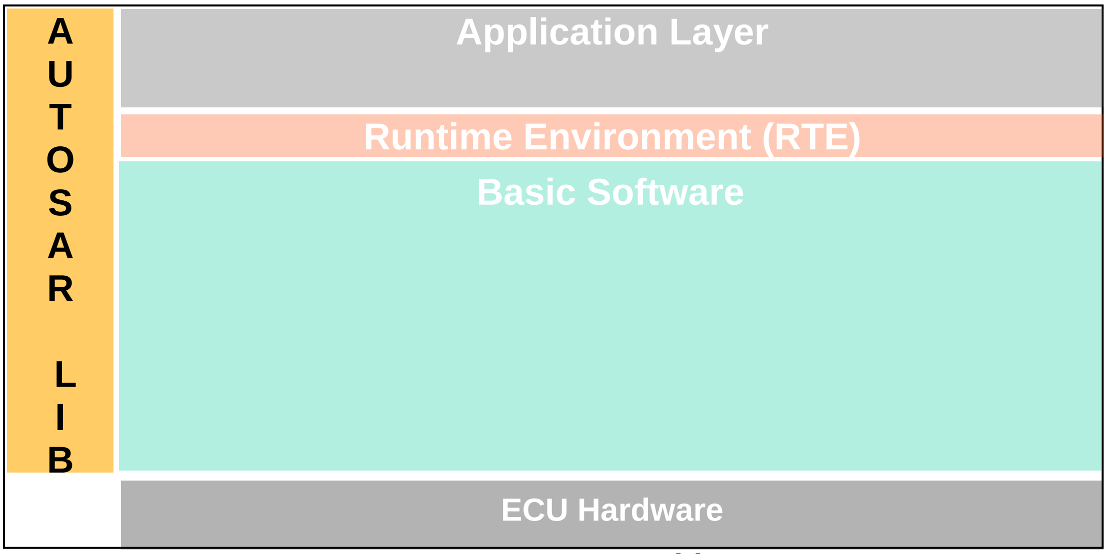

Figure : Layered architecture

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

[MFL001] ⌈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 ⌋ (BSW31400005)

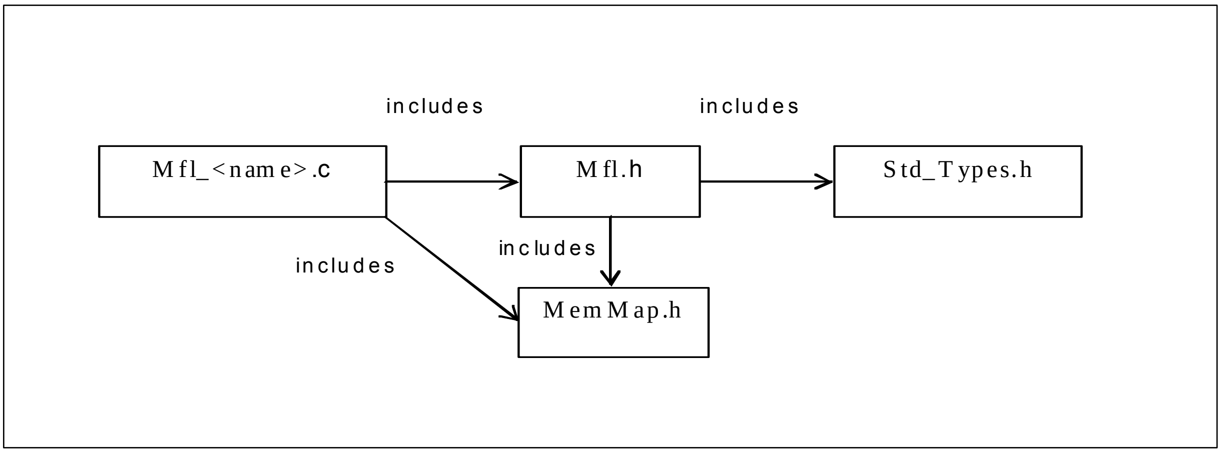

Figure : File structure

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
BSW003		MFL815
BSW00304	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	MFL812
BSW00306	All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, un...	MFL813
BSW00318		MFL815
BSW00321		MFL815
BSW00348	Each AUTOSAR library Module implementation *.	MFL811
BSW00378	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	MFL812
BSW00407		MFL815, MFL816
BSW00411		MFL816
BSW00436	Each AUTOSAR library Module implementation *.	MFL810
BSW007	The library, written in C programming language, should conform to the HIS subset of the MISRA C S...	MFL809
BSW31400002	Mfl library shall not require initialization phase.	MFL800
BSW31400003	Mfl library shall not require a shutdown operation phase.	MFL801
BSW31400005	The Mfl module shall provide the following files:	MFL001
BSW31400013	Error detection: The validity of the parameters passed to library functions must be checked at th...	MFL819, MFL817
BSW31400015	The Mfl library shall be implemented in a way that the code can be shared among callers in differ...	MFL806
BSW31400017	Usage of macros should be avoided.	MFL807
BSW31400018	A library function shall not call any BSW modules functions, e.	MFL808

7 Functional specification

7.1 Error classification

[MFL821] ⌈No error classification definition as DET call not supported by library⌋ ( )

7.2 Error detection

[MFL819] ⌈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. ⌋ (BSW31400013)
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

[MFL817] ⌈The functions shall not call the DET for error notification. ⌋ (BSW31400013)

7.4 Initialization and shutdown

[MFL800] ⌈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. ⌋ (BSW31400002)

[MFL801] ⌈Mfl library shall not require a shutdown operation phase. ⌋ (BSW31400003)

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-DependencyOnLibrary 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

[MFL806] ⌈The Mfl library shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋ (BSW31400015)

[MFL807] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used. ⌋ (BSW31400017)

[MFL808] ⌈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. ⌋ (BSW31400018)

[MFL809] ⌈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*/⌋ (BSW007)

[MFL810] ⌈Each AUTOSAR library Module implementation <library>*.c and <library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism. ⌋ (BSW00436)

[MFL811] ⌈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. ⌋ (BSW00348)

[MFL812] ⌈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. ⌋ (BSW00304, BSW00378)

[MFL813] ⌈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. ⌋ (BSW00306)

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

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

Results are rounded towards zero.
- 0 <= X < 1 rounded to 0
- -1 < X <= 0 rounded to 0

8.4 Comment about routines optimized for target

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

For example, depending on the target, two types of optimization can be done:
- Some routines can be replaced by another routine using integer promotion.

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

8.5 Routine definitions

8.5.1 Floating point to Fixed-Point Conversion

[MFL005] ⌈

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:	**MFL006:** Returns the integer value of the fixed point result of the conversion, determined according to the following equation. Result = ValFloat * 2ValFixedExponent **MFL007:** The return value shall be saturated to the return type boundary values in the event of overflow or underflow. **MFL008:** If it is necessary to round the result of this equation, it is rounded toward zero.	

⌋ ( )

Function ID and prototypes

[MFL009] ⌈

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

[MFL010] ⌈

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:	**MFL011:** Returns the floating-point result of the conversion, determined according to the following equation. Result = ValFixedInteger * 2-ValFixedExponent	

⌋ ( )

Function ID and prototypes

[MFL012] ⌈

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

[MFL013] ⌈

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:	MFL014 Returns the integer value determined by rounding the argument toward zero. Eg. 36.56 will be truncated to 36.00	

⌋ ( )

[MFL015] ⌈

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:	**MFL016** Returns the integer value determined by rounding the argument toward the near-est whole number. Eg. 36.56 will be rounded to 37.00 **MFL017** If the argument is halfway between two integers, it is rounded away from zero. Eg. 36.5 will be rounded to 37.00	

⌋ ( )

[MFL018] ⌈

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:	**MFL019** Returns the integer value determined by rounding the argument toward positive infinity.	

⌋ ( )

[MFL020] ⌈

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:	**MFL021** 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 yn and can be represented in the following equation.

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

In the equation, the following symbols are used

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

8.5.4.1 Structure definitions for controller routines

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

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

K : Amplification factor, the description of the semantic is given in
T1 : Decay time constant
Tv : Lead time
Tn : Follow-up time

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

dT : Time step = sampling interval

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

K_<size>, K_C : Amplification factor
T1rec_<size> : Reciprocal delay time constant = 1/ T1
Tv _<size>, Tv _C : Lead time
Tnrec _<size>, Tnrec _C : Reciprocal follow-up time = 1/ Tn.
dT_<size> : Time step = sampling interval
TeQ_<size> : Time equivalent, 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.

[MFL025] ⌈

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		

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		

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		

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		

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		

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		

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 (*Type1 and Type 2*) controller routine		

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 (*Type1 and Type 2*) controller routine		

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 (*Type1 and Type 2*) controller routine		

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 (*Type1 and Type 2*) controller routine		

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) = Kp * X.

8.5.4.2.1 ‘P’ Controller

[MFL026] ⌈

Service name:	Mfl_PCalc	

Syntax:	`void Mfl_PCalc(`
	`    float32 X_f32,`
	`    float32* P_pf32,`
	`    float32 K_f32`
	`)`	

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

	MFL027: 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.

[MFL030] ⌈

Service name:	Mfl_POut_f32	

Syntax:	`float32 Mfl_POut_f32(`
	`    const float32* const P_pf32`
	`)`	

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:	**MFL031:** This routine returns 'P' controllers output value limited by the return data type 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

[MFL032] ⌈

Service name:	Mfl_PT1Calc	

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

Service ID[hex]:	0x1A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the PT1 element
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
Parameters (in-out):	State_cpst	Pointer to PT1 state structure
Parameters (out):	None	
Return value:	None	
Description:	**MFL033:**	

	This routine computes PT1 controller output value using below difference equation Yn= exp(-dT/T1) * Yn-1+ K(1- exp(-dT/T1)) * Xn-1 This derives implementation: Output_value = (TeQ_f32 * State_cpst->Y1) + K_f32 * (1 - TeQ_f32) * State_cpst->X1 where TeQ_f32 = exp (-dT/T1) **MFL034:** Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter TeQ_f32. **MFL035:** If (T1 = 0) then PT1 controller follows Input value, State_cpst->Y1 = k_f32 * X_f32 **MFL036:** 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

⌋ ( )

8.5.4.3.2 'PT1' Set State Value

This routine can be realised using inline function.

[MFL037] ⌈

Service name:	Mfl_PT1SetState	

Syntax:	`void Mfl_PT1SetState(`
	`    Mfl_StatePT1_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 Y1_f32`
	`)`	

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. **MFL038:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL039:** Initialisation of input state variable X1. State_cpst->X1 = X1_f32.	

⌋ ( )

8.5.4.3.3 Calculate time equivalent Value

This routine can be realised using inline function.

[MFL040] ⌈

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:	**MFL041:** This routine calculates time equivalent factor TeQ = exp(-T1rec_f32 * dT_f32)	

⌋ ( )

8.5.4.3.4 Calculate an approximate time equivalent Value

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

[MFL315] ⌈

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:	**MFL316:** This routine calculates time equivalent factor TeQApp = 1 - (T1rec_f32 * dT_f32)	

⌋ ( )

8.5.4.3.5 Get 'PT1' output

This routine can be realised using inline function.

[MFL042] ⌈

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:	**MFL043:** This routine returns 'PT1' controllers output value 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

[MFL044] ⌈

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:	**MFL045:** This routine computes DT1 controller output value using differential equation, Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn-1 - Xn-2) / dT) This derives implementation: Output_value = (TeQ * State_cpst->Y1) + K_f32 * (1 - TeQ) * ((State_cpst->X1 - State_cpst->X2) / dT) where TeQ = exp(-dT/T1) **MFL046:**	

	Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter TeQ_f32. **MFL047:** If (T1 = 0) then DT1 controller follows Input value, Output_value = k_f32 * (X_f32 - State_cpst->X1) / dT **MFL048:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL049:** 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

⌋ ( )

8.5.4.4.2 'DT1' Controller - Type2

[MFL300] ⌈

Service name:	Mfl_DT1Typ2Calc	

Syntax:	`void Mfl_DT1Typ2Calc(`
	`    float32 X_f32,`
	`    Mfl_StateDT1Typ2_Type* const State_cpst,`
	`    float32 K_f32,`
	`    float32 TeQ_f32,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0xC0	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the DT1 controller
	K_f32	Amplification factor
	TeQ_f32	Time equivalent
	dT_f32	Sample Time
Parameters (in-out):	State_cpst	Pointer to state structure
Parameters (out):	None	
Return value:	None	
Description:	**MFL301:** This routine computes DT1 controller output value using differential equation, Yn= exp(-dT/T1) * Yn-1+ K * (1- exp(-dT/T1)) * ((Xn - Xn-1) / dT) This derives implementation: Output_value = (TeQ * State_cpst->Y1) + K_f32 * (1 - TeQ) * ((X_f32 - State_cpst->X1) / dT) where TeQ = exp(-dT/T1) **MFL302:** Mfl_CalcTeQ_f32 shall be used for calculation of time equivalent parameter TeQ_f32. **MFL303:** If (T1 = 0) then DT1 controller follows Input value, Output_value = k_f32 * (X_f32 - State_cpst->X1) / dT	

	MFL304: Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL305:** Current input value X_f32 shall be stored to State_cpst->X1. State_cpst->X1 = X_f32

⌋ ( )

8.5.4.4.3 Set 'DT1' State Value – Type1

This routine can be realised using inline function.

[MFL050] ⌈

Service name:	Mfl_DT1Typ1SetState	

Syntax:	`void Mfl_DT1Typ1SetState(`
	`    Mfl_StateDT1Typ1_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 X2_f32,`
	`    float32 Y1_f32`
	`)`	

Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for the input state X1
	X2_f32	Initial value for the input state X2
	Y1_f32	Initial value for the output state
Parameters (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. **MFL051:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL052:** 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.

[MFL306] ⌈

Service name:	Mfl_DT1Typ2SetState

Syntax:	`void Mfl_DT1Typ2SetState(`
	`    Mfl_StateDT1Typ2_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 Y1_f32`
	`)`

Service ID[hex]:	0xC1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for the input state
	Y1_f32	Initial value for the output state
Parameters (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. **MFL307:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL308:** 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.

[MFL053] ⌈

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:	**MFL054:** This routine returns 'DT1' controller's output value Output value = State_cpst->Y1	

⌋ ( )

8.5.4.4.6 Get 'DT1' output – Type2

This routine can be realised using inline function.

[MFL310] ⌈

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:	**MFL311:** This routine returns 'DT1' controller's output value 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

[MFL055] ⌈

Service name:	Mfl_PDCalc	

Syntax:	`void Mfl_PDCalc(`
	`    float32 X_f32,`
	`    Mfl_StatePD_Type* const State_cpst,`
	`    const Mfl_ParamPD_Type* const Param_cpst,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0x2A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the PD controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (in-out):	State_cpst	Pointer to state structure
Parameters (out):	None	
Return value:	None	
Description:	**MFL056:** This routine computes proportional plus derivative controller output value using differential equation: Yn= K(1+Tv/dT) * Xn- K(Tv/dT) * Xn-1 This derives implementation: Output_value = (Param_cpst->K_C * (1+ Param_cpst->Tv_C/dT_f32) * X_f32) - (Param_cpst->K_C * (Param_cpst->Tv_C/dT_f32) * State_cpst->X1) **MFL057:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL058:** 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.

[MFL059] ⌈

Service name:	Mfl_PDSetState	

Syntax:	`void Mfl_PDSetState(`
	`    Mfl_StatePD_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 Y1_f32`
	`)`	

Service ID[hex]:	0x2B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (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. **MFL060:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL061:** 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.

[MFL062] ⌈

Service name:	Mfl_PDSetParam	

Syntax:	`void Mfl_PDSetParam(`
	`    Mfl_ParamPD_Type* const Param_cpst,`
	`    float32 K_f32,`
	`    float32 Tv_f32`
	`)`	

Service ID[hex]:	0x2C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	K_f32	Amplification factor
	Tv_f32	Lead time
Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to internal parameter structure
Return value:	None	
Description:	**MFL063:** The routine sets the parameter structure of a PD element. Initialisation of amplification factor. Param_cpst->K_C = K_f32 **MFL064:**	

	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.

[MFL066] ⌈

Service name:	Mfl_PDOut_f32	

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

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:	**MFL067:** This routine returns 'PD' controllers output value. Output value = State_cpst->Y1	

⌋ ( )

8.5.4.6 Integral component

This routine calculates Integration element.

8.5.4.6.1 'I' Controller

[MFL068] ⌈

Service name:	Mfl_ICalc	

Syntax:	`void Mfl_ICalc(`
	`    float32 X_f32,`
	`    Mfl_StateI_Type* const State_cpst,`
	`    float32 K_f32,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0x30	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'I' controller
	K_f32	Amplification factor
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to state variable.
Return value:	None	
Description:	**MFL069:** This routine computes DT1 controller output value using differential equation,	

	Yn= Yn-1 + K * dT * Xn-1 This derives implementation: Output_value = State_cpst->Y1 + K_f32 * dT_f32 * State_cpst->X1 **MFL070:** 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

⌋ ( )

8.5.4.6.2 ‘I’ Controller with limitation

[MFL320] ⌈

Service name:	Mfl_ILimCalc	

Syntax:	`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`
	`)`	

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:	**MFL321:**
	This routine computes DT1 controller output value using differential equation,
	Yn= Yn-1 + K * dT * Xn-1
	
	This derives implementation:
	Output_value = State_cpst->Y1 + K_f32 * dT_f32 * State_cpst->X1
	
	MFL322:
	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
	
	MFL323:
	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

[MFL324] ⌈

Service name:	Mfl_CtrlSetLimit	

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

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:	**MFL325:** 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 "

[MFL367] ⌈

Service name:	Mfl_CtrlSetLimits	

Syntax:	`void Mfl_CtrlSetLimits(`
	`    Mfl_Limits_Type* const Limit_cpst,`
	`    float32 Min_f32,`
	`    float32 Max_f32`
	`)`	

Service ID[hex]:	0xC9	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Min_f32	Minimum limit
	Max_f32	Maximum limit
Parameters (in-out):	Limit_cpst	Pointer to limit structure
Parameters (out):	None	
Return value:	None	
Description:	**MFL368:** 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.

[MFL071] ⌈

Service name:	Mfl_ISetState	

Syntax:	`void Mfl_ISetState(`
	`    Mfl_StateI_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 Y1_f32`
	`)`	

Service ID[hex]:	0x31	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (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. **MFL072:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL073:** 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.

[MFL074] ⌈

Service name:	Mfl_IOut_f32	

Syntax:	`float32 Mfl_IOut_f32(`
	`    const Mfl_StateI_Type* const State_cpst`
	`)`	

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:	**MFL075:** This routine returns 'I' controllers output value. 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)

[MFL076] ⌈

Service name:	Mfl_PITyp1Calc	

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

Service ID[hex]:	0x35	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to the internal state structure.
Return value:	None	
Description:	**MFL077:** This routine computes Proportional plus integral controller (implicit type) output value using differential equation: Yn= Yn-1+ K * Xn- K * (1 - dT/Tn) * Xn-1 This derives implementation: Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_f32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_f32) * State_cpst->X1) **MFL078:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL079:** 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)

[MFL326] ⌈

Service name:	Mfl_PITyp1LimCalc	

Syntax:	`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`
	`)`	

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:	**MFL327:**
	This routine computes Proportional plus integral controller (implicit type) output value using differential equation:
	Yn= Yn-1+ K * Xn- K * (1 - dT/Tn) * Xn-1
	
	This derives implementation:
	Output_value = State_cpst->Y1 + (Param_cpst->K_C * X_f32) - (Param_cpst->K_C * (1 - Param_cpst->Tnrec_C * dT_f32) * State_cpst->X1)
	
	MFL328:
	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
	
	MFL329:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	MFL330:
	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)

[MFL080] ⌈

Service name:	Mfl_PITyp2Calc	

Syntax:	`void Mfl_PITyp2Calc(`
	`    float32 X_f32,`
	`    Mfl_StatePI_Type* const State_cpst,`
	`    const Mfl_ParamPI_Type* const Param_cpst,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0x36	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'PI' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to the internal state structure.
Return value:	None	
Description:	**MFL081:** This routine computes Proportional plus integral controller (explicit type) output value using differential equation: Yn= Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1	

	This derives implementation: Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_f32) * X_f32) - (Param_cpst->K_C * State_cpst->X1) **MFL082:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL083:** Current input value X_f32 shall be stored to State_cpst->X1. State_cpst->X1 = X_f32

⌋ ( )

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

[MFL331] ⌈

Service name:	Mfl_PITyp2LimCalc	

Syntax:	`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`
	`)`	

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:	**MFL332:**
	This routine computes Proportional plus integral controller (explicit type) output value using differential equation:
	Yn= Yn-1 + K * (1 + dT/Tn) * Xn - K * Xn-1
	
	This derives implementation:
	Output_value = State_cpst->Y1 + (Param_cpst->K_C * (1 + Param_cpst->Tnrec_C * dT_f32) * X_f32) - (Param_cpst->K_C * State_cpst->X1)
	
	MFL333:
	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
	
	MFL334:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	MFL335:	

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

⌋ ( )

8.5.4.7.5 Set 'PI' State Value

This routine can be realised using inline function.

[MFL084] ⌈

Service name:	Mfl_PISetState	

Syntax:	`void Mfl_PISetState(`
	`    Mfl_StatePI_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 Y1_f32`
	`)`	

Service ID[hex]:	0x37	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (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. **MFL085:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL086:** Initialisation of input state variable X1. State_cpst->X1 = X1_f32	

⌋ ( )

8.5.4.7.6 Set 'PI' Parameters

This routine can be realised using inline function.

[MFL087] ⌈

Service name:	Mfl_PISetParam	

Syntax:	`void Mfl_PISetParam(`
	`    Mfl_ParamPI_Type* const Param_cpst,`
	`    float32 K_f32,`
	`    float32 Tnrec`
	`)`	

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:	**MFL088:** The routine sets the parameter structure of a PI element. Initialisation of amplification factor. Param_cpst->K_C = K_f32 **MFL089:** 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.

[MFL090] ⌈

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:	**MFL091:** This routine returns 'PI' controllers output value. 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)

[MFL092] ⌈

Service name:	Mfl_PIDTyp1Calc	

Syntax:	`void Mfl_PIDTyp1Calc(`
	`    float32 X_f32,`
	`    Mfl_StatePID_Type* const State_cpst,`
	`    const Mfl_ParamPID_Type* const Param_cpst,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0x3A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time

Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to the internal state structure.
Return value:	None	
Description:	**MFL093:** This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation: Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2 This derives implementation: calc1 = Param_cpst->K_C * (1 + t_val) * X_f32 calc2 = Param_cpst->K_C * (1 - dT_f32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1 calc3 = Param_cpst->K_C * t_val * State_cpst->X2 Output_value = State_cpst->Y1 + calc1 - calc2 + calc3 Where t_val = Param_cpst->Tv_C / dT_f32 **MFL094:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL095:** 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	

⌋ ( )

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

[MFL340] ⌈

Service name:	Mfl_PIDTyp1LimCalc	

Syntax:	`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`
	`)`	

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:	**MFL341:** This routine computes Proportional plus integral plus derivative controller (implicit type) output value using differential equation: Yn=Yn-1+ K * (1 + Tv/dT) * Xn- K *(1 - dT/Tn + 2Tv/dT) * Xn-1 + K * (Tv/dT) * Xn-2 This derives implementation:	

	calc1 = Param_cpst->K_C * (1 + t_val) * X_f32
	calc2 = Param_cpst->K_C * (1 - dT_f32 * Param_cpst->Tnrec_C + 2 * t_val) * State_cpst->X1
	calc3 = Param_cpst->K_C * t_val * State_cpst->X2
	Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
	Where t_val = Param_cpst->Tv_C / dT_f32
	
	MFL342:
	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
	
	MFL343:
	Calculated Output_value shall be stored to State_cpst->Y1.
	State_cpst->Y1 = Output_value
	
	MFL344:
	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

⌋ ( )

8.5.4.8.3 'PID' Controller – Type2 (Explicit type)

[MFL096] ⌈

Service name:	Mfl_PIDTyp2Calc	

Syntax:	`void Mfl_PIDTyp2Calc(`
	`    float32 X_f32,`
	`    Mfl_StatePID_Type* const State_cpst,`
	`    const Mfl_ParamPID_Type* const Param_cpst,`
	`    float32 dT_f32`
	`)`	

Service ID[hex]:	0x3B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X_f32	Input value for the 'PID' controller
	Param_cpst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (in-out):	None	
Parameters (out):	State_cpst	Pointer to the internal state structure
Return value:	None	
Description:	**MFL097:** This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation: Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2 This derives implementation: calc1 = Param_cpst->K_C * (1 + dT_f32 * Param_cpst->Tnrec_C + t_val) * X_f32 calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1 calc3 = Param_cpst->K_C * t_val * State_cpst->X2 Output_value = State_cpst->Y1 + calc1 - calc2 + calc3	

	Where t_val = Param_cpst->Tv_C / dT_f32 **MFL098:** Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL099:** 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)

[MFL345] ⌈

Service name:	Mfl_PIDTyp2LimCalc	

Syntax:	`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`
	`)`	

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:	**MFL346:**
	This routine computes Proportional plus integral plus derivative controller (explicit type) output value using differential equation:
	Yn = Yn-1 + K * (1 + dT/Tn+ Tv/dT) * Xn- K *(1 + 2Tv/dT) * Xn-1+ K * (Tv/dT) * Xn-2
	
	This derives implementation:
	calc1 = Param_cpst->K_C * (1 + dT_f32 * Param_cpst->Tnrec_C + t_val) * X_f32
	calc2 = Param_cpst->K_C * (1 + 2 * t_val) * State_cpst->X1
	calc3 = Param_cpst->K_C * t_val * State_cpst->X2
	Output_value = State_cpst->Y1 + calc1 - calc2 + calc3
	Where t_val = Param_cpst->Tv_C / dT_f32
	
	MFL347:
	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	

	MFL348: Calculated Output_value shall be stored to State_cpst->Y1. State_cpst->Y1 = Output_value **MFL349:** 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.

[MFL100] ⌈

Service name:	Mfl_PIDSetState	

Syntax:	`void Mfl_PIDSetState(`
	`    Mfl_StatePID_Type* const State_cpst,`
	`    float32 X1_f32,`
	`    float32 X2_f32,`
	`    float32 Y1_f32`
	`)`	

Service ID[hex]:	0x3C	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	X1_f32	Initial value for input state
	X2_f32	Initial value for input state
	Y1_f32	Initial value for output state
Parameters (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. **MFL101:** Initialisation of output state variable Y1. State_cpst->Y1 = Y1_f32 **MFL102:** 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.

[MFL103] ⌈

Service name:	Mfl_PIDSetParam
Syntax:	`void Mfl_PIDSetParam(` `    Mfl_ParamPID_Type* const Param_cpst,`

	`float32 K_f32,`
	`float32 Tv_f32,`
	`float32 Tnrec_f32`
	`)`	

Service ID[hex]:	0x3D	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	K_f32	Amplification factor
	Tv_f32	Lead Time
	Tnrec_f32	Reciprocal follow-up timer
Parameters (in-out):	None	
Parameters (out):	Param_cpst	Pointer to internal parameter structure
Return value:	None	
Description:	**MFL104:** The routine sets the parameter structure of a PID element. Initialisation of amplification factor. Param_cpst->K_C = K_f32 **MFL105:** Initialisation of lead time state variable Param_cpst->Tv_C = Tv_f32 **MFL106:** 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.

[MFL107] ⌈

Service name:	Mfl_PIDOut_f32	

Syntax:	`float32 Mfl_PIDOut_f32(`
	`const Mfl_StatePID_Type* const State_cpst`
	`)`	

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:	**MFL108:** This routine returns 'PID' controllers output value. Output value = State_cpst->Y1	

⌋ ( )

8.5.5 Magnitude and Sign

[MFL110] ⌈

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:	**MFL111** Returns the absolute value of the argument (ValAbs), determined according to the following equation. ValAbs = | ValValue |	

⌋ ( )

[MFL112] ⌈

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. **MFL113:** ValSign = 1, ValValue > 0.0 **MFL114:** ValSign = 0, ValValue == 0.0 **MFL115:** ValSign = -1, ValValue < 0.0	

⌋ ( )

8.5.6 Limiting

[MFL116] ⌈

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:	**MFL117:** Returns the value of the larger of the two arguments (ValMax), determined according to the following equation. ValMax = ValValue1, ValValue1 ≥ ValValue2 ValMax = ValValue2, ValValue1 < ValValue2	

⌋ ( )

[MFL118] ⌈

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:	**MFL119:** Returns the value of the smaller of the two arguments (Min), determined according to the following equation. Min = Value1, Value1 ≤ Value2 Min = Value2, Value1 > Value2	

⌋ ( )

[MFL120] ⌈

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	Prevuous value of newval.
	maxdif	Maximum difference allowed between old value and the new value.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Limited value.

Description:	**MFL121:**
	An increasing value and decreasing value is rate limited by maxdif
	
	if (newval > oldval) and ((newval - oldval) > maxdif)
	Result = oldval + maxdif
	else if (newval < oldval) and ((oldval - newval) > maxdif)
	Result = oldval - maxdif
	else
	Result = newval

⌋ ( )

[MFL122] ⌈

Service name:	Mfl_Limit_f32	

Syntax:	`float32 Mfl_Limit_f32(`
	`    float32 val,`
	`    float32 lowLim,`
	`    float32 upLim`
	`)`	

Service ID[hex]:	0x48	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	val	Quantity to be bounded.
	lowLim	Lower bound.
	upLim	Upper bound
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	float32	Limited value.
Description:	**MFL123:** Returns the bounded value (newVal), determined according to the following equation. newVal = lowLim, val ≤ lowLim newVal = upLim, val ≥ upLim newVal = val, lowLim < val < upLim	

⌋ ( )

8.5.7 Logarithms and Exponentials

[MFL130] ⌈

Service name:	Mfl_Pow_f32	

Syntax:	`float32 Mfl_Pow_f32(`
	`    float32 ValBase,`
	`    float32 ValExp`
	`)`	

Service ID[hex]:	0x50	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	ValBase	Base to be raised to an exponent. Valid range:ValBase > 0.0
	ValExp	Exponent by which to raise the base.
Parameters (in-out):	None	
Parameters (out):	None	

Return value:	float32	ValBase raised to ValExp power.
Description:	**MFL131:** ValResult = ValBaseValExp **MFL132:** If ValExp = 0, and ValBase = 0, ValResult = 1, (00 = 1) If ValBase = 0 and ValExp<> 0, ValResult = 0 (0ValExp **=** 0) MFL133: If ValBase and ValExp are having maximum value of type float32, the return value will be toward positive infinity.	

⌋ ( )

[MFL135] ⌈

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:	**MFL136:** Returns the square root of the operand (ValSqrt), determined according to the following equation ValSqrt = ValValue1/2 **MFL137:** ValValue shall be passed as positive value. (ValValue ≥ 0)	

⌋ ( )

[MFL140] ⌈

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:	**MFL141:** Returns the exponential of the operand (ValExp), determined according to the following equation. ValExp = eValValue	

	MFL142: ValValue Range shall be [-24PI, +24PI]

⌋ ( )

[MFL145] ⌈

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:	**MFL146:** Returns the natural (base-e) logarithm of the operand (ValLog), determined according to the following equation. ValLog = loge(ValValue) **MFL147:** ValValue shall be passed as > 0 value.	

⌋ ( )

8.5.8 Trigonometry

[MFL150] ⌈

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:	MFL151: Calculates the sine of the argument. Result: result = sine (value) MFL152: Range of value shall be [-24PI, +24PI]	

⌋ ( )

[MFL155] ⌈

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:	MFL156: Calculates the cosine of the argument. Result: result = cosine (value) MFL157: Range of value shall be [-24PI, +24PI]

⌋ ( )

[MFL160] ⌈

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:	**MFL161:** Calculates the tangent of the argument. Result: result = tangent(value) **MFL163:** Range of the value shall be [-24PI, +24PI]

⌋ ( )

[MFL165] ⌈

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:	**MFL166:** Returns the arc sine of an angle, in the range of -pi/2 through pi/2. **MFL167:** If the argument is zero, then the result is a zero. **MFL168:** Range of the value shall be [-1, +1]	

⌋ ( )

Note : "This routine (Mfl_arcSin_f32) is depreciated and will not be supported in future release
Replacement routine : Mfl_ArcSin_f32"

[MFL350] ⌈

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:	**MFL351:** Returns the arc sine of an angle, in the range of -pi/2 through pi/2. **MFL352:** If the argument is zero, then the result is a zero. **MFL353:** Range of the value shall be [-1, +1]	

⌋ ( )

[MFL170] ⌈

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:	**MFL171:** Returns the arc cosine of an angle, in the range of 0.0 through pi.	

	MFL172: 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"

[MFL354] ⌈

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:	**MFL355:** Returns the arc cosine of an angle, in the range of 0.0 through pi. **MFL356:** Range of the value shall be [-1, +1]

⌋ ( )

[MFL175] ⌈

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:	MFL176: Returns the arc tangent of an angle, in the range of -pi/2 through pi/2. MFL177: If the argument is zero, then the result is a zero with the same sign as the argument.

⌋ ( )

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

Replacement routine : Mfl_ArcTan_f32"

[MFL357] ⌈

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:	**MFL358:** Returns the arc tangent of an angle, in the range of -pi/2 through pi/2. **MFL359:** If the argument is zero, then the result is a zero with the same sign as the argu-ment.	

[MFL180] ⌈

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:	**MFL181:**
	Returns the arc tangent of an angle, in the range of [-pi to pi]
	
	MFL182:
	If the argument is zero, then the result is a zero with the same sign as the argu-ment.
	
	MFL183:
	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"

[MFL360] ⌈

Service name:	Mfl_ArcTan2_f32	

Syntax:	`float32 Mfl_ArcTan2_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:	**MFL361:**
	Returns the arc tangent of an angle, in the range of [-pi to pi]
	
	MFL362:
	If the argument is zero, then the result is a zero with the same sign as the argument.
	
	MFL363:
	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

[MFL190] ⌈

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. **MFL191:** Output = (Value1 + Value2) / 2	

⌋ ( )

8.5.10 Array Average

[MFL192] ⌈

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. MFL193**:** Output = (Array[0] + Array[1]+_ _ Array[N-1]) / N	

⌋ ( )

8.5.11 Hypotenuse

[MFL195] ⌈

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:	MFL196: This service computes the length of a vector: Result = square_root (x_value * x_value + y_value * y_value)	

	MFL197: The result is rounded off.

⌋ ( )

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

[MFL200] ⌈

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		

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

[MFL201] ⌈

Service name:	Mfl_RampCalc

Syntax:	`void Mfl_RampCalc(`
	`    float32 X_f32,`
	`    Mfl_StateRamp_Type* const State_cpst,`
	`    const Mfl_ParamRamp_Type* const Param_cpcst,`
	`    float32 dT_f32`
	`)`

Service ID[hex]:	0x90
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	X_f32	Target value for the ramp to reach
	Param_cpcst	Pointer to parameter structure
	dT_f32	Sample Time
Parameters (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 > Target) or (State_cpst->State_f32 < Target).
	
	MFL202:
	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)
	
	MFL203:
	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)
	
	MFL204:
	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.
	
	MFL205:
	Comparison of State and Target decides ramp direction.
	If(State_cpst->State_f32 > Target) then State_cpst->Dir_s8 = RISING
	If(State_cpst->State_f32 < Target) then State_cpst->Dir_s8 = FALLING
	If(State_cpst->State_f32 == Target) then State_cpst->Dir_s8 = END
	
	MFL206:
	This routine returns State value
	Return_value = State_cpst->State_f32
	
	MFL207:
	Calculated ramp value shall be stored to State_cpst->State_f32 variable.	

⌋ ( )

8.5.12.2 Ramp Initialisation

[MFL208] ⌈

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. **MFL209:** Ramp direction is initialised with END value. User has no possibility to change or modify ramp direction. State_cpst->Dir_s8 = END E.g. of ramp direction states: RISING = 1, FALLING = -1, END = 0 **MFL275:** Initialisation of state variable State_cpst ->State_f32 = Val_f32 **MFL276:** Initialisation of switch variable. User has no possibility to change or modify switch initialization value. State_cpst->Switch_s8 = OFF E.g. of switch states: TARGET_A = 1, TARGET_B = -1, OFF = 0

⌋ ( )

8.5.12.3 Ramp Set Slope

[MFL210] ⌈

Service name:	Mfl_RampSetParam	

Syntax:	`void Mfl_RampSetParam(`
	`    Mfl_ParamRamp_Type* const Param_cpst,`
	`    float32 SlopePosVal_f32,`
	`    float32 SlopeNegVal_f32`
	`)`	

Service ID[hex]:	0x92	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SlopePosVal_f32	Positive slope value
	SlopeNegVal_f32	Negative slope value
Parameters (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_RampParam_Type. **MFL211:** 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

[MFL212] ⌈

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:	**MFL213:** Returns the internal state of the ramp element. Return Value = State_cpcst->State_f32	

⌋ ( )

8.5.12.5 Ramp Jump routine

[MFL214] ⌈

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:	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. **MFL215:** 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 = Target State_cpst->Dir_s8 = END **MFL277:** If target value is changed to new value and ramp has reached its old target value then normal ramp behavior is maintained. State_cpst->Dir_s8 = END **MFL278:** 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.	

	MFL279:
	Comparison of State and Target decides ramp direction.
	If(State_cpst->State_f32 > Target) then State_cpst->Dir_s8 = RISING
	If(State_cpst->State_f32 < Target) then State_cpst->Dir_s8 = FALLING
	If(State_cpst->State_f32 == Target) then State_cpst->Dir_s8 = END
	
	MFL281:
	This routine returns State value.
	Return_value = State_cpst->State_f32
	
	MFL282:
	This routine decided if jump has to be done or not in case of change in target.
	Mfl_RampCalc function shall be called after this function that a jump or the standard ramp behaviour is executed.

⌋ ( )

8.5.12.6 Ramp switch routine

[MFL216] ⌈

Service name:	Mfl_RampCalcSwitch_f32	

Syntax:	`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`
	`)`	

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.
	
	MFL217:
	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
	
	MFL218:
	State_cpst->Dir_s8 holds direction information
	Ramp direction status: RISING, FALLING, END
	
	MFL219:
	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. **MFL220:** Once ramp value reaches the selected target value, the ramp direction status is switched to END. State_cpst->Dir_s8 == END **MFL221:** 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) **MFL222:** 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 "

[MFL369] ⌈

Service name:	Mfl_RampCalcSwitch	

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

Service ID[hex]:	0xCA	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Xa_f32	Target value for the ramp to reach if switch is in position 'A'
	Xb_f32	Target value for the ramp to reach if switch is in position 'B'
	Switch	Switch to decide target value
Parameters (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. **MFL370:** Parameter Switch decides which target value is selected. If Switch = TRUE, then Xa_f32 is selected. State_cpst->Switch_s8 is set to TARGET_A Return value = Xa_f32 If Swtich = FALSE, then Xb_f32 is selected. State_cpst->Switch_s8 is set to TARGET_B Return value = Xb_f32	

	MFL371: 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. **MFL372:** Mfl_RampCalcSwitch has to be called before Mfl_RampCalc routine

⌋ ( )

8.5.12.7 Get Ramp Switch position

[MFL223] ⌈

Service name:	Mfl_RampGetSwitchPos	

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

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:	**MFL224:** Gets the current switch position of ramp switch function. Return value = TRUE if Switch position State_cpst->Switch_s8 = TARGET_A Return value = FALSE if Switch position State_cpst->Switch_s8 = TARGET_B	

⌋ ( )

8.5.12.8 Check Ramp Activity

[MFL225] ⌈

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:	**MFL226:** This routine checks the status of the ramp and returns a TRUE if the ramp is active, otherwise it returns FALSE. 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

[MFL236] ⌈

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. MFL237: Return value is TRUE if input is greater then center plus halfDelta switching point. MFL238: Return value is FALSE if input is less then center minus halfDelta switching point. MFL239: Return value is former state value if input is in the range of halfDelta around the center switching point	

⌋ ( )

[MFL240] ⌈

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

[MFL241] ⌈

Service name:	Mfl_HystLeftRight_<InTypeMn>_<OutTypeMn>

Syntax:	`boolean Mfl_HystLeftRight_<InTypeMn>_<OutTypeMn>(`
	`    <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. MFL242: Return value is TRUE if input is greater then right switching point. MFL243: Return value is FALSE if input is less then left switching point. MFL244: Return value is former state value if input is between left and right switching points	

⌋ ( )

[MFL245] ⌈
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

[MFL246] ⌈

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 MFL247: Return value is TRUE if input is greater then right switching point. MFL248: Return value is FALSE if input is less then right switching point minus delta. MFL249:	

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

⌋ ( )

[MFL250] ⌈
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

[MFL251] ⌈

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. MFL252: Return value is TRUE if input is greater then left switching point plus delta. MFL253: Return value is FALSE if input is less then left switching point. MFL254: Return value is former state value if input is between left switching points and left plus delta.	

⌋ ( )

[MFL255] ⌈
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

[MFL256] ⌈

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. MFL257: Buffer data stores input samples hence reproduced output signal will reduce samples in case high delay time. MFL258: Buffer size shall be configured as per the delay time range requirement.	

⌋ ( )

Structure definition for function argument

[MFL259] ⌈

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		

⌋ ( )

8.5.15 Debounce routines

8.5.15.1 Mfl_Debounce

[MFL260] ⌈

Service name:	Mfl_Debounce_u8_u8

Syntax:	`boolean Mfl_Debounce_u8_u8(`
	`    boolean X,`
	`    Mfl_DebounceState_Type* State,`
	`    Mfl_DebounceParam_Type* Param,`
	`    float32 dT`
	`)`

Service ID[hex]:	0xB0
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	X	Input value
	Param	Pointer to state structure of type Mfl_DebounceState_Type
	dT	Sample Time
Parameters (in-out):	State	Time to be delayed
Parameters (out):	None	
Return value:	boolean	Returns the debounced input value

Description:	**MFL261:**
	This routine debounces a digital input signal and returns the state of the signal as a boolean value.
	If(X != State->XOld) then check start debouncing.
	MFL262:
	If transition is from Low to High, then use Param->TimeLowHigh as debouncing time otherwise use Param->TimeHighLow
	
	MFL263:
	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
	MFL264:
	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

[MFL265] ⌈

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		

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

[MFL266] ⌈

Service name:	Mfl_DebounceInit

Syntax:	`void Mfl_DebounceInit(`
	`    Mfl_DebounceState_Type* State,`
	`    boolean X`
	`)`

Service ID[hex]:	0xB1	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	State	--
	X	Initial value for the input state
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	**MFL267:** This routine call shall stop the debouncing timer. State->Timer = 0 **MFL268:** Sets the input state to the given init value. State->XOld = X	

⌋ ( )

8.5.15.3 Mfl_DebounceSetParam

[MFL269] ⌈

Service name:	Mfl_DebounceSetparam	

Syntax:	`void Mfl_DebounceSetparam(`
	`    Mfl_DebounceParam_Type* Param,`
	`    float32 THighLow,`
	`    float32 TLowHigh`
	`)`	

Service ID[hex]:	0xB2	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (in-out):	None	
Parameters (out):	Param	Pointer to state structure of type Mfl_DebounceParam_Type
Return value:	None	
Description:	MFL270: This routine sets timing parameters, time for high to low transition and time for low to high for debouncing. 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 "

[MFL365] ⌈

Service name:	Mfl_DebounceSetParam	
Syntax:	`void Mfl_DebounceSetParam( Mfl_DebounceParam_Type* Param, float32 THighLow, float32 TLowHigh )`	
Service ID[hex]:	0xC8	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	THighLow	Value for TimeHighLow of Mfl_DebounceParam_Type
	TLowHigh	Value for TimeLowHigh of Mfl_DebounceParam_Type
Parameters (in-out):	None	
Parameters (out):	Param	Pointer to state structure of type Mfl_DebounceParam_Type
Return value:	None	
Description:	**MFL366:** This routine sets timing parameters, time for high to low transition and time for low to high for debouncing. Param-> TimeHighLow = THighLow Param-> TimeLowHigh = TLowHigh	

8.5.16 Ascending Sort Routine

[MFL271] ⌈

Service name:	Mfl_SortAscend_f32	
Syntax:	`void Mfl_SortAscend_f32( float32* Array, uint16 Num )`	
Service ID[hex]:	0xB5	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Num	Size of an data array
Parameters (in-out):	Array	Pointer to an data array
Parameters (out):	None	
Return value:	None	
Description:	**MFL272:** 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

[MFL273] ⌈

Service name:	Mfl_SortDescend_f32
Syntax:	`void Mfl_SortDescend_f32( float32* Array,`

	`uint16 Num` `)`	
Service ID[hex]:	0xBA	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Num	Size of an data array
Parameters (in-out):	Array	Pointer to an data array
Parameters (out):	None	
Return value:	None	
Description:	**MFL274:** 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

[MFL285] ⌈

Service name:	Mfl_MedianSort_f32_f32	

Syntax:	`float32 Mfl_MedianSort_f32_f32(`
	`float32* const Array,`
	`uint8 N`
	`)`	

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:	**MFL286:** This routine sorts values of an array in ascending order. Input array passed by the pointer shall have sorted values after this routine call. 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] **MFL287:** Returns the median value of sorted array in case of N is even. Result = (Sorted_array[N/2] + Sorted_array[(N/2) - 1]) / 2 Eg. Sorted_array[4] = [8.0, 10.0, 15.0, 42.0] Result = (15.0 + 10.0) / 2.0 = 12.5 **MFL288:** Returns the median value of sorted array in case of N is odd. Return_Value = Sorted_array [N/2] = 15 Eg. Sorted_array[5] = [8.0, 10.0, 15.0, 42.0, 88.0] Result = 15.0 **MFL289:**	

	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

[MFL815] ⌈

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 (BSW00407). ⌋ (BSW00407, BSW003, BSW00318, BSW00321)

[MFL816] ⌈
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. ⌋ (BSW00407, BSW00411)

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

[MFL814] ⌈The standardized common published parameters as required by BSW00402 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]. ⌋ (BSW00402, BSW00374, BSW00379)

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

10.2 Configuration option

[MFL818] ⌈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. ⌋ (BSW31400001)

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

11 Not applicable requirements

[MFL822]⌈These requirements are not applicable to this specification.⌋( )