Document Title	Specification of Floating Point Interpolation Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	398

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Added import type
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes.
2022-11-24	R22-11	AUTOSAR Release Management	• Requirements added SWS_Ifl_91000 to SWS_Ifl_91003 • Requirements added SWS_Ifl_00226, SWS_Ifl_00228, SWS_Ifl_00229, SWS_Ifl_00231, SWS_Ifl_00232, SWS_Ifl_00234, SWS_Ifl_00235 • Modified SWS_Ifl_00170 , SWS_Ifl_00011 and SWS_Ifl_00221
2021-11-25	R21-11	AUTOSAR Release Management	• Editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• Chapter 7.1 Error sections updated
2019-11-28	R19-11	AUTOSAR Release Management	• Editorial changes • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes

△

2017-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Section 2 has been revisited to update Default Error Tracer instead of Development Error tracer • Updated IFL document to support MISRA 2012 standard. (Removed redundant statements in SWS_Ifl_00209 which already exist in SWS_BSW document and SWS_SRS document) • Updated the correct reference to SRS_BSW_General (SRS_BSW_00437) & (SRS_BSW_00448) for SWS_Ifl_00210 & SWS_Ifl_00224 requirements.
2015-07-31	4.2.2	AUTOSAR Release Management	• Updated Record layouts definitions for SWS_Ifx_00170 • Updated SWS_Ifl_00001 for naming convention under Section 5.1, File Structure • Updated valid range for float32 in Table 1 of Section 8.1
2014-10-31	4.2.1	AUTOSAR Release Management	• Added IFL RecordLayout Blueprint reference in section 3.1 • The usage of const is updated in function parameters for SWS_Ifl_00010, SWS_Ifl_00021 & SWS_Ifl_00025 • IFL Blueprint modified for the schema version • Serial numbers in Section 3.2
2013-10-31	4.1.2	AUTOSAR Release Management	• Corrected array-out-of-bounds for Ifl_IpoMap function • Editorial changes

▽

△

2013-03-15	4.1.1	AUTOSAR Administration	• Corrected the formula for integrated map interpolation and map interpolation • Corrected array out-of-bounds for curve interpolation • Modified the reference to non-existant metamodel elementCalprmElementPrototype to Param-eterDataPrototype • Corrected for 'DependencyOnArtifact'
2011-12-22	4.0.3	AUTOSAR Administration	• Error classification support and defi-nition removed as DET call not sup-ported by library • Configuration parameter description / support removed for XXX_ GetVersionInfo routine. • XXX_GetVersionInfo routine name corrected.
2010-09-30	3.1.5	AUTOSAR Administration	• DPSearch function optimised using structure pointer • Removal of normalised functions
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

Contents

1 Introduction and functional overview

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

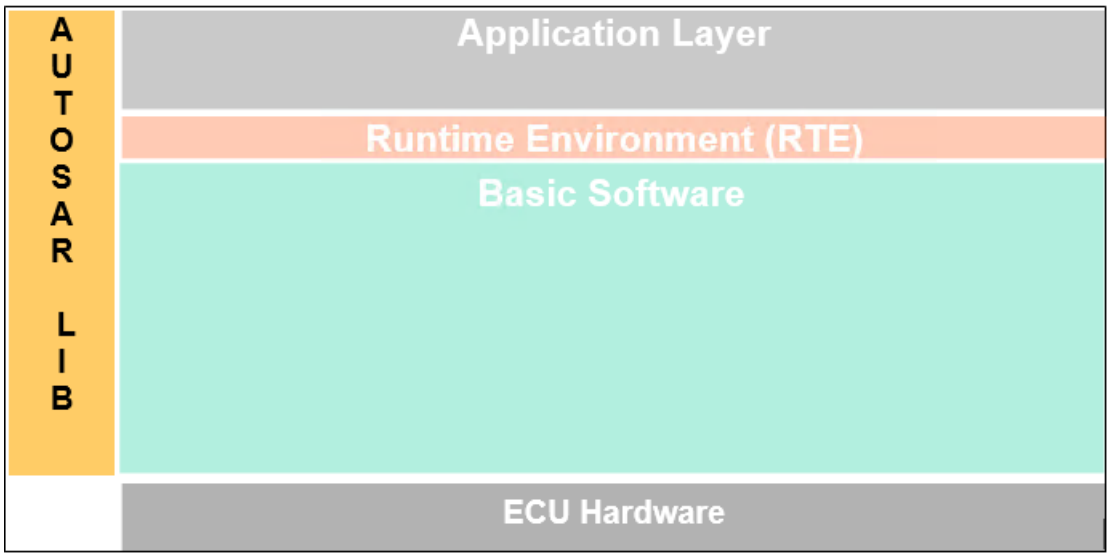

Figure 1.1: Layered Architecture

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

The interpolation library contains the following routines:

- Distributed data point search and interpolation
- Integrated data point search and interpolation

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
DET	Default Error Tracer
ROM	Read only memory
hex	Hexadecimal
Rev	Revision
f32	Mnemonic for the float32, specified in AUTOSAR_SWS_PlatformTypes
IFL	Interpolation Floating point Library
Mn	Mnemonic
Lib	Library
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

Table 2.1: Acronyms and Abbreviations

3 Related documentation

3.1 Input documents & related standards and norms

[1] IFL_RecordLayout_Blueprint
AUTOSAR_MOD_IFL_RecordLayout_Blueprint.arxml

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

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [3, SWS BSW General], which is also valid for IFL Library.

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

4 Constraints and assumptions

4.1 Limitations

No limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

[SWS_Ifl_00001] ⌈The Ifl module shall provide the following files:

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

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.: Ifl_IntIpoMap_f32f32_f32.c etc.

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

- 2.1 Group by object family: eg.:Ifl_IpoCur.c, Ifl_DPSearch.c
- 2.2 Group by routine family: eg.: Ifl_IpoMap.c
- 2.3 Group by method family: eg.: Ifl_Ipo.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 Ifl functions, eg.: Ifl.c.

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

6 Requirements Tracing

Requirement	Description	Satisfied by
[SRS_BSW_00003]	All software modules shall provide version and identification information	[SWS_Ifl_00215]
[SRS_BSW_00007]	All Basic SW Modules written in C language shall conform to the MISRA C 2012 Standard.	[SWS_Ifl_00209]
[SRS_BSW_00304]	All AUTOSAR Basic Software Modules shall use only AUTOSAR data types instead of native C data types	[SWS_Ifl_00212]
[SRS_BSW_00306]	AUTOSAR Basic Software Modules shall be compiler and platform independent	[SWS_Ifl_00213]
[SRS_BSW_00318]	Each AUTOSAR Basic Software Module file shall provide version numbers in the header file	[SWS_Ifl_00215]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_Ifl_00215]
[SRS_BSW_00348]	All AUTOSAR standard types and constants shall be placed and organized in a standard type header file	[SWS_Ifl_00211]
[SRS_BSW_00374]	All Basic Software Modules shall provide a readable module vendor identification	[SWS_Ifl_00214]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_Ifl_00212]
[SRS_BSW_00379]	All software modules shall provide a module identifier in the header file and in the module XML description file.	[SWS_Ifl_00214]
[SRS_BSW_00402]	Each module shall provide version information	[SWS_Ifl_00214]
[SRS_BSW_00407]	Each BSW module shall provide a function to read out the version information of a dedicated module implementation	[SWS_Ifl_00215] [SWS_Ifl_00216]
[SRS_BSW_00411]	All AUTOSAR Basic Software Modules shall apply a naming rule for enabling/disabling the existence of the API	[SWS_Ifl_00216]
[SRS_BSW_00437]	Memory mapping shall provide the possibility to define RAM segments which are not to be initialized during startup	[SWS_Ifl_00210]
[SRS_BSW_00448]	Module SWS shall not contain requirements from other modules	[SWS_Ifl_00224]
[SRS_LIBS_00001]	The functional behavior of each library functions shall not be configurable	[SWS_Ifl_00218]
[SRS_LIBS_00002]	A library shall be operational before all BSW modules and application SW-Cs	[SWS_Ifl_00200]
[SRS_LIBS_00003]	A library shall be operational until the shutdown	[SWS_Ifl_00201]

△

Requirement	Description	Satisfied by
[SRS_LIBS_00005]	Each library shall provide one header file with its public interface	[SWS_Ifl_91000] [SWS_Ifl_91001] [SWS_Ifl_91002] [SWS_Ifl_91003]
[SRS_LIBS_00009]	All library functions shall be re-entrant	[SWS_Ifl_91000] [SWS_Ifl_91001] [SWS_Ifl_91002] [SWS_Ifl_91003]
[SRS_LIBS_00011]	All function names and type names shall start with "Library short name_"	[SWS_Ifl_91000] [SWS_Ifl_91001] [SWS_Ifl_91002] [SWS_Ifl_91003]
[SRS_LIBS_00013]	The error cases, resulting in the check at runtime of the value of input parameters, shall be listed in SWS	[SWS_Ifl_00217] [SWS_Ifl_00219]
[SRS_LIBS_00015]	It shall be possible to configure the microcontroller so that the library code is shared between all callers	[SWS_Ifl_00206]
[SRS_LIBS_00017]	Usage of macros should be avoided	[SWS_Ifl_00207]
[SRS_LIBS_00018]	A library function may only call library functions	[SWS_Ifl_00208]

Table 6.1: Requirements Tracing

7 Functional specification

7.1 Error Classification

[SWS_Ifl_00223] ⌈Section 7.1 "Error Handling" of the document "General Specification of Basic Software Modules" describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of five error types which may occur in BSW modules.⌋

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

7.1.1 Development Errors

There are no development errors.

7.1.2 Runtime Errors

There are no runtime errors.

7.1.3 Production Errors

There are no production errors.

7.1.4 Extended Production Errors

There are no extended production errors.

7.2 Error detection

[SWS_Ifl_00219]

Upstream requirements: SRS_LIBS_00013

⌈Error detection: Function should check at runtime (both in production and development code) the value of input parameters, especially cases where erroneous value can bring to fatal error or unpredictable result, if they have the values allowed by the function specification. All the error cases shall be listed in SWS and the function should

return a specified value (in SWS) 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.⌋

E.g. If passed value > 32 for a bit-position

or a negative number of samples of an axis distribution is passed to a routine.

7.3 Error notification

[SWS_Ifl_00217]

Upstream requirements: SRS_LIBS_00013

⌈The functions shall not call the DET for error notification.⌋

7.4 Initialization and shutdown

[SWS_Ifl_00200]

Upstream requirements: SRS_LIBS_00002

⌈Ifl 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.⌋

[SWS_Ifl_00201]

Upstream requirements: SRS_LIBS_00003

⌈Ifl library shall not require a shutdown operation phase.⌋

7.5 Using Library API

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

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

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

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

7.6 Library implementation

[SWS_Ifl_00206]

Upstream requirements: SRS_LIBS_00015

⌈The Ifl library shall be implemented in a way that the code can be shared among callers in different memory partitions.⌋

[SWS_Ifl_00207]

Upstream requirements: SRS_LIBS_00017

⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used.⌋

[SWS_Ifl_00208]

Upstream requirements: SRS_LIBS_00018

⌈A library function can call other library functions because all library functions shall be re-entrant. A library function shall not call any BSW modules functions, e.g. the DET.⌋

[SWS_Ifl_00209]

Upstream requirements: SRS_BSW_00007

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

Please refer to SWS_BSW_00115 for more details.⌋

[SWS_Ifl_00210]

Upstream requirements: SRS_BSW_00437

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

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

[SWS_Ifl_00211]

Upstream requirements: SRS_BSW_00348

⌈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.⌋

[SWS_Ifl_00212]

Upstream requirements: SRS_BSW_00304, SRS_BSW_00378

⌈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.⌋

[SWS_Ifl_00213]

Upstream requirements: SRS_BSW_00306

⌈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.⌋

[SWS_Ifl_00220] ⌈If input value is less than first distribution entry then first value of the distribution array shall be returned or used in the interpolation routines. If input value is greater than last distribution entry then last value of the distribution array shall be returned or used in the interpolation routines.⌋

[SWS_Ifl_00221] ⌈Axis distribution passed to Ifx routines shall have normal monotony sequence.⌋

8 API specification

8.1 Imported types

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

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

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

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

Size	Platform Type	Mnemonic	Range
unsigned 8-Bit	boolean	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	[-3.4028235E38, 3.4028235E38]

Table 8.1: Mnemonic for Base Types

As a convention in the rest of the document:

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

[SWS_Ifl_91004] Definition of imported datatypes of module Ifl ⌈

Module	*Header File*	*Imported Type*
Std	Std_Types.h	Std_VersionInfoType

⌋

8.2 Type definitions

Structure definition:

[SWS_Ifl_00005] Definition of datatype Ifl_DPResultF32_Type ⌈

Name	Ifl_DPResultF32_Type	
Kind	Structure	
Elements	Index	
	Type	uint32
	Comment	Data point index
	Ratio	
	Type	float32
	Comment	Data point ratio
Description	Structure used for data point search for index and ratio	
Available via	Ifl.h	

⌋

[SWS_Ifl_00006] ⌈Ifl_DPResultF32_Type structure shall not be read/write/modified by the user directly. Only Ifl routines shall have access to this structure.⌋

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 Interpolation routines definitions

Interpolation between two given points is calculated as shown below.

$$result = y_0 + (y_1 - y_0) \cdot \frac{x - x_0}{x_1 - x_0}$$

where: x is the input value

$x_0 =$ data point before x

$x_1 =$ data point after x

$y_0 =$ value at x_0

$y_1 =$ value at x_1

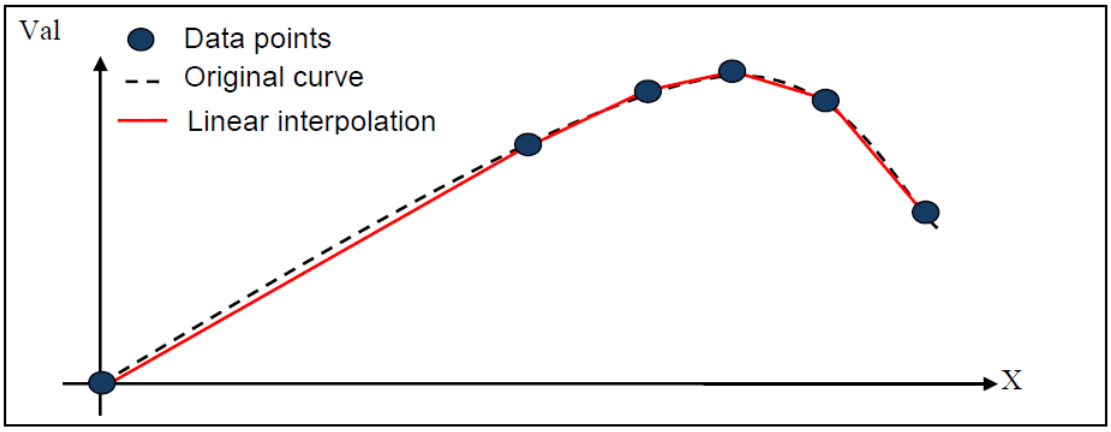

Figure 8.1: Linear interpolation

Data point arrays can be grouped as one array or one structure for all elements as shown below.

one array for all elements :

float32 Curve_f32 []={5,0.0,10.0,26.0,36.0,64.0,1.0,12.0,17.0,11.0,6.0};

one structure for all elements :

struct

{ uint32 N = 5;

float32 X[] ={0.0,10.0,26.0,36.0,64.0};

float32 Y[] ={1.0,12.0,17.0,11.0,6.0};

} Curve_f32;

where, number of samples = 5

X axis distribution = 0.0 to 64.0

Y axis distribution = 1.0 to 6.0

Interpolation routines accepts arguments separately to support above scenarios. Routine call example is given below for array and structure grouping respectively.

Example :

float32 Ifl_IntIpoCur_f32_f32 (15, Curve_f32[0], &Curve_f32[1], &Curve_f32[6]);

float32 Ifl_IntIpoCur_f32_f32 (15, Curve_f32.N, &Curve_f32.X, &Curve_f32.Y);

Interpolation can be calculated in two ways as shown below:

1. Distributed data point search and interpolation
2. Integrated data point search and interpolation

8.5.1 Distributed data point search and interpolation

In this interpolation method data point search (e.g. index and ratio) is calculated using routine Ifl_DPSearch_f32 which returns result structure Ifl_DPResultF32_Type. It contains index and ratio information. This result can be used by curve interpolation and map interpolation.

8.5.1.1 Data Point Search

[SWS_Ifl_00010] Definition of API function Ifl_DPSearch_f32 ⌈

Service Name	Ifl_DPSearch_f32	

Syntax	`void Ifl_DPSearch_f32 (`
	`  Ifl_DPResultF32_Type* dpResult,`
	`  float32 Xin,`
	`  uint32 N,`
	`  const float32* X_array`
	`)`	

Service ID [hex]	0x001	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Xin	Input value

▽

△

	N	Number of samples
	X_array	Pointer to distribution array
Parameters (inout)	None	
Parameters (out)	dpResult	Pointer to the result structure
Return value	None	
Description	This routine searches the position of input Xin within the given distribution array X_array, and returns index and ratio necessary for interpolation.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00011] ⌈Returned Index shall be the lowest index for which (X_array[index] < Xin < X_array[index + 1]).

If (X_array[0] <=Xin <= X_array[N-1]), then returned Index shall be the lowest index.

dpResult->Index=indexdpResult->Ratio=(Xin-X_array[index]) / (X_array[index + 1] - X_array[index]) dpResult ->Index = index

dpResult ->Ratio = (Xin - X_array[index]) / (X_array [index+1] - X_array [index])⌋

For a given array float32 X[] ={0.0,10.0,26.0,36.0,64.0};

If Xin = 20.0 then

dpResult ->Index = 1

dpResult ->Ratio = (20.0 - 10.0) / (26.0 - 10.0) = 0.625

[SWS_Ifl_00012] ⌈If the input value matches with one of the distribution array values, then

return respective index and ratio as 0.0.

If Input Xin == X_array[index], then

dpResult ->Index = index (Index of the set point)

dpResult ->Ratio = 0.0⌋

[SWS_Ifl_00013] ⌈If (Xin < X_array[0]), then return first index of an array and ratio = 0.0

dpResult ->Index = 0

dpResult ->Ratio = 0.0⌋

[SWS_Ifl_00014] ⌈If (Xin > X_array[N-1]), then return last index of an array and ratio = 0.0

dpResult ->Index = N - 1

dpResult ->Ratio = 0.0⌋

[SWS_Ifl_00015] ⌈The minimum value of N shall be 1⌋

[SWS_Ifl_00016] ⌈If X_array[Index+1] == X_array[Index], then the Ratio shall be zero.

dpResult->Ratio = 0.0⌋

[SWS_Ifl_00017] ⌈This routine returns index and ratio through the structure of type Ifl_DPResultF32_Type⌋

8.5.1.2 Curve interpolation

[SWS_Ifl_00021] Definition of API function Ifl_IpoCur_f32 ⌈

Service Name	Ifl_IpoCur_f32	
Syntax	`float32 Ifl_IpoCur_f32 ( const Ifl_DPResultF32_Type* dpResult, const float32* Val_array )`	
Service ID [hex]	0x004	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResult	Data point search result
	Val_array	Pointer to the result distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the Interpolation
Description	Based on searched index and ratio information, this routine calculates and returns interpolation for curve.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00022] ⌈index = dPResult->Index

if dPResult->Ratio == 0.0

Result = Val_array[index]

else

Result = Val_array[index] + (Val_array[index+1] - Val_array[index]) * dpResult->Ratio⌋

[SWS_Ifl_00180] ⌈Do not call this routine until you have searched the axis using the Ifl_DPSearch routine. Only then it is ensured that the search result (Ifl_DPResultF32_Type) contains valid data and is not used uninitialized.⌋

8.5.1.3 Map interpolation

[SWS_Ifl_00025] Definition of API function Ifl_IpoMap_f32 ⌈

Service Name	Ifl_IpoMap_f32	
Syntax	`float32 Ifl_IpoMap_f32 ( const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const float32* Val_array )`	
Service ID [hex]	0x005	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResultX	Data point search result for x axis
	dpResultY	Data point search result for y axis
	num_value	Number of y axis points
	Val_array	Pointer to result distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the Interpolation
Description	Based on searched indices and ratios information using the Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for map.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00026] ⌈Based on searched indices and ratios information using the Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for map.

BaseIndex = dpResultX->Index * num_value + dpResultY->Index

if (dpResultX->Ratio == 0)

if (dpResultY->Ratio == 0)

Result = Val_array [BaseIndex]

else

LowerY = Val_array [BaseIndex]

UpperY = Val_array [BaseIndex + 1]

Result = LowerY + (UpperY - LowerY) * dpResultY->Ratio

else

if (dpResultY->Ratio == 0)

LowerX = Val_array [BaseIndex]

UpperX = Val_array [BaseIndex + num_value]

Result = LowerX + (UpperX - LowerX) * dpResultX->Ratio

else

LowerY = Val_array [BaseIndex]

UpperY = Val_array [BaseIndex + 1]

LowerX = LowerY + (UpperY - LowerY) * dpResultY->Ratio

LowerY = Val_array [BaseIndex + num_value]

UpperY = Val_array [BaseIndex + num_value + 1]

UpperX = LowerY + (UpperY - LowerY) * dpResultY->Ratio

Result = LowerX + (UpperX - LowerX) * dpResultX->Ratio⌋

[SWS_Ifl_00181] ⌈Do not call this routine until you have searched the axis using the Ifl_DPSearch routine. Only then it is ensured that the search result (Ifl_DPResultF32_Type) contains valid data and is not used uninitialized.⌋

8.5.1.4 Single point interpolation

[SWS_Ifl_00030] Definition of API function Ifl_Interpolate_f32 ⌈

Service Name	Ifl_Interpolate_f32	
Syntax	`float32 Ifl_Interpolate_f32 ( float32 Value1, float32 Value2, float32 Coef )`	
Service ID [hex]	0x006	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Value1	First value to be used in the interpolation.
	Value2	Second value to be used in the interpolation.
	Coef	Interpolation coefficient.
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Iterpolated value
Description	Returns the result of the linear interpolation (Result), determined according to the following equation.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00031] ⌈Result = Value1 + (Coef * (Value2 - Value1))⌋

8.5.2 Integrated data point search and interpolation

In this method of interpolation, single routine does data point search (e.g. Index and ratio) and interpolation for curve, map.

8.5.2.1 Integrated curve interpolation

[SWS_Ifl_00035] Definition of API function Ifl_IntIpoCur_f32_f32 ⌈

Service Name	Ifl_IntIpoCur_f32_f32	
Syntax	`float32 Ifl_IntIpoCur_f32_f32 ( float32 X_in, uint32 N, const float32* X_array, const float32* Val_array )`	
Service ID [hex]	0x010	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	X_in	Input value
	N	Number of samples
	X_array	Pointer to X distribution
	Val_array	Pointer to Y values
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the Interpolation
Description	This routine calculates interpolation of a curve at position Xin using below equa-tion.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00036] ⌈index = minimum value of integer index if (X_array[index] < Xin < X_array[index+1])

RatioX = (Xin - X_array[index]) / (X_array [index+1] - X_array [index])

Result = Val_array[index] + (Val_array[index+1] - Val_array[index])*RatioX⌋

[SWS_Ifl_00037] ⌈If the input value matches with one of the distribution array values, then result will be the respective Y array element indicated by the index.

If (Xin == X_array[index]),

Result = Val_array[index]⌋

[SWS_Ifl_00038] ⌈If Xin < X_array[0], then

Result = Val_array[0]⌋

[SWS_Ifl_00039] ⌈If Xin > X_array[N-1], then

Result = Val_array[N-1]⌋

[SWS_Ifl_00040] ⌈The minimum value of N shall be 1⌋

8.5.2.2 Integrated map interpolation

[SWS_Ifl_00041] Definition of API function Ifl_IntIpoMap_f32f32_f32 ⌈

Service Name	Ifl_IntIpoMap_f32f32_f32	
Syntax	`float32 Ifl_IntIpoMap_f32f32_f32 ( float32 Xin, float32 Yin, uint32 Nx, uint32 Ny, const float32* X_array, const float32* Y_array, const float32* Val_array )`	
Service ID [hex]	0x011	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Xin	Input value for X axis
	Yin	Input value for Y axis
	Nx	Number of X axis intervals
	Ny	Number of Y axis intervals
	X_array	Pointer to the X axis distribution array
	Y_array	Pointer to the Y axis distribution array
	Val_array	Pointer to the result axis distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the Map Interpolation
Description	This routine calculates Interpolation of a map at position X and Y using below equations.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00042] ⌈indexX = minimum value of index if (X_array[indexX] < Xin < X_array[indexX+1])

indexY = minimum value of index if (Y_array[indexY] < Yin < Y_array[indexY+1])

RatioX = (Xin - X_array[indexX]) / (X_array [indexX+1] - X_array [indexX])

RatioY = (Yin - Y_array[indexY]) / (Y_array [indexY+1] - Y_array [indexY])

BaseIndex = IndexX * Ny + indexY

LowerY = Val_array [BaseIndex]

UpperY = Val_array [BaseIndex + 1]

LowerX = LowerY + (UpperY - LowerY) * RatioY

LowerY = Val_array [BaseIndex + Ny]

UpperY = Val_array [BaseIndex + Ny + 1]

UpperX = LowerY + (UpperY - LowerY) * RatioY

Result = LowerX + (UpperX - LowerX) * RatioX⌋

[SWS_Ifl_00043] ⌈If (Xin == X_array[indexX]) and (Y_array[indexY] < Yin < Y_array[indexY+1])

Result = Val_array [BaseIndex] + (Val_array [BaseIndex+1] - Val_array[BaseIndex]) * RatioY⌋

[SWS_Ifl_00044] ⌈If (Yin == Y_array[indexY]) and (X_array[indexX] < Xin < X_array[indexX+1])

Result = Val_array [BaseIndex] + (Val_array [BaseIndex+Ny] - Val_array[BaseIndex]) * RatioX⌋

[SWS_Ifl_00045] ⌈If (Xin == X_array[indexX]) and (Yin == Y_array[indexY])

Result = Val_array [BaseIndex]⌋

[SWS_Ifl_00046] ⌈If Xin < X_array[0], then

indexX = 0,

RatioX = 0.0⌋

[SWS_Ifl_00047] ⌈If Xin > X_array[Nx-1], then

indexX = Nx - 1,

RatioX = 0.0⌋

[SWS_Ifl_00048] ⌈If Yin < Y_array[0], then

indexY = 0,

RatioY = 0.0⌋

[SWS_Ifl_00049] ⌈If Yin > Y_array[Ny-1], then

indexY = Ny - 1,

RatioY = 0.0⌋

[SWS_Ifl_00050] ⌈The minimum value of N shall be 1⌋

8.5.2.3 Cuboid 3D interpolation

[SWS_Ifl_91000] Definition of API function Ifl_IpoCub_f32

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service Name	Ifl_IpoCub_f32	
Syntax	`float32 Ifl_IpoCub_f32 ( const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const float32* Val_array )`	
Service ID [hex]	0x12	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResultX	Data point search result for X axis
	dpResultY	Data point search result for Y axis
	dpResultZ	Data point search result for Z axis
	num_x	Number of X axis points
	num_y	Number of Y axis points
	Val_array	Pointer to the result axis distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the interpolation
Description	Based on searched indices and ratios information using the relevant Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for a 3D cuboid.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00226] ⌈Based on searched indices and ratios information using the Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for 3D cuboids.

The axis order memory representation is [z][x][y]. This is the column-major orientation COLUMN_DIR from ASAM standard. The first axis z specifies the selected slice.

Implementation:

Linear interpolation along x-axis between the result of two 2D interpolations between neighbouring X/Y Maps.

num_slice = num_x * num_y

if(dpResultZ->Ratio==0.0)

Result=Ifl_IpoMap_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice * dpResultZ->Index])

else

LowerXY=Ifl_IpoMap_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice * dpResultZ ->Index])

UpperXY=Ifl_IpoMap_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice * dpResultZ ->Index + 1])

Result=Ifl_Interpolate_f32 (LowerXY, UpperXY, dpResultZ->Ratio)⌋

8.5.2.4 Mixed type interpolation of integer curve

[SWS_Ifl_91001] Definition of API function Ifl_IpoCur_<InTypeMn>_f32

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service Name	Ifl_IpoCur_<InTypeMn>_f32	
Syntax	`float32 Ifl_IpoCur_<InTypeMn>_f32 ( const Ifl_DPResultF32_Type* dpResult, const <InType>* Val_array )`	
Service ID [hex]	0x13 to 0x16	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResult	Data point search result
	Val_array	Pointer to the result axis distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the interpolation
Description	Based on searched indices and ratios information using the relevant Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for a curve.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00228] ⌈index=dpResult->Index

if dpResult->Ratio==0.0

Result=Val_array[index]

else

Result=Val_array[index] + (Val_array[index + 1]- Val_array[index]) * dpResult->Ratio⌋

Here is the list of implemented routines:

[SWS_Ifl_00229] ⌈

Routine ID[hex]	Routine prototype
0x013	float32 Ifl_IpoCur_u8_f32 (const Ifl_DPResultF32_Type* dpResult, const uint8* Val_array)
0x014	float32 Ifl_IpoCur_u16_f32 (const Ifl_DPResultF32_Type* dpResult, const uint16* Val_array)
0x015	float32 Ifl_IpoCur_s8_f32 (const Ifl_DPResultF32_Type* dpResult, const sint8* Val_array)
0x016	float32 Ifl_IpoCur_s16_f32 (const Ifl_DPResultF32_Type* dpResult, const sint16* Val_array)

⌋

8.5.2.5 Mixed type interpolation of integer map

[SWS_Ifl_91002] Definition of API function Ifl_IpoMap_<InTypeMn>_f32

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service Name	Ifl_IpoMap_<InTypeMn>_f32	
Syntax	`float32 Ifl_IpoMap_<InTypeMn>_f32 ( const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const <InType>* Val_array )`	
Service ID [hex]	0x18 to 0x1b	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResultX	Data point search result for X axis
	dpResultY	Data point search result for Y axis
	num_value	Number of Y axis points
	Val_array	Pointer to the result axis distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the interpolation

▽

△

Description	Based on searched indices and ratios information using the relevant Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for a map.
Available via	Ifl.h

⌋

[SWS_Ifl_00231] ⌈Based on searched indices and ratios information using the Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for map.

BaseIndex = dpResultX->Index * num_value + dpResultY->Index

if (dpResultX->Ratio == 0.0)

if (dpResultY->Ratio == 0.0)

Result = Val_array [BaseIndex]

else

LowerY = Val_array [BaseIndex]

UpperY = Val_array [BaseIndex + 1]

Result = LowerY + (UpperY - LowerY) * dpResultY->Ratio

else

if (dpResultY->Ratio == 0.0)

LowerX = Val_array [BaseIndex]

UpperX = Val_array [BaseIndex + num_value]

Result = LowerX + (UpperX - LowerX) * dpResultX->Ratio

else

LowerY = Val_array [BaseIndex]

UpperY = Val_array [BaseIndex + 1]

LowerX = LowerY + (UpperY - LowerY) * dpResultY->Ratio

LowerY = Val_array [BaseIndex + num_value]

UpperY = Val_array [BaseIndex + num_value + 1]

UpperX = LowerY + (UpperY - LowerY) * dpResultY->Ratio

Result = LowerX + (UpperX - LowerX) * dpResultX->Ratio⌋

Here is the list of implemented routines.

[SWS_Ifl_00232] ⌈

Routine ID[hex]	Routine prototype
0x018	float32 Ifl_IpoMap_u8_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const uint8* Val_array)
0x019	float32 Ifl_IpoMap_u16_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const uint16* Val_array)
0x01A	float32 Ifl_IpoMap_s8_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const sint8* Val_array)
0x01B	float32 Ifl_IpoMap_s16_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, uint32 num_value, const sint16* Val_array)

⌋

8.5.2.6 Mixed type interpolation of integer 3D Cuboid

[SWS_Ifl_91003] Definition of API function Ifl_IpoCub_<InTypeMn>_f32

Upstream requirements: SRS_LIBS_00005, SRS_LIBS_00009, SRS_LIBS_00011

⌈

Service Name	Ifl_IpoCub_<InTypeMn>_f32	
Syntax	`float32 Ifl_IpoCub_<InTypeMn>_f32 ( const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const <InType>* Val_array )`	
Service ID [hex]	0x1C to 0x1F	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	dpResultX	Data point search result for X axis
	dpResultY	Data point search result for Y axis
	dpResultZ	Data point search result for Z axis
	num_x	Number of X axis points
	num_y	Number of Y axis points
	Val_array	Pointer to the result axis distribution array
Parameters (inout)	None	
Parameters (out)	None	
Return value	float32	Result of the interpolation
Description	Based on searched indices and ratios information using the relevant Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for a 3D cuboid.	
Available via	Ifl.h	

⌋

[SWS_Ifl_00234] ⌈Based on searched indices and ratios information using the Ifl_DPSearch_f32 routine, this routine calculates and returns the interpolation result for 3D cuboids.

The axis order memory representation is [z][x][y]. This is the column-major orientation COLUMN_DIR from the ASAM standard. The first axis z specifies the selected slice.

Implementation:

Linear interpolation along x-axis between the result of two 2D interpolations between neighboring X/Y Maps.

num_slice = num_x*num_y

if (dpResultZ->Ratio == 0.0)

Result = Ifl_IpoMap_<InTypeMn>_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice*dpResultZ->Index])

else

LowerXY = Ifl_IpoMap_<InTypeMn>_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice*dpResultZ->Index])

UpperXY = Ifl_IpoMap_<InTypeMn>_f32 (dpResultX, dpResultY, num_y, Val_array[num_slice*dpResultZ->Index + 1])

Result = Ifl_Interpolate_f32 (LowerXY, UpperXY, dpResultZ->Ratio)⌋

Here is the list of implemented routines.

[SWS_Ifl_00235] ⌈

Routine ID[hex]	Routine prototype
0x01C	float32 Ifl_IpoCub_u8_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const uint8* Val_array)
0x01D	float32 Ifl_IpoCub_u16_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const uint16* Val_array)
0x01E	float32 Ifl_IpoCub_s8_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const sint8* Val_array)
0x01F	float32 Ifl_IpoCub_s16_f32 (const Ifl_DPResultF32_Type* dpResultX, const Ifl_DPResultF32_Type* dpResultY, const Ifl_DPResultF32_Type* dpResultZ, uint16 num_x, uint16 num_y, const sint16* Val_array)

⌋

8.5.3 Record layouts for interpolation routines

Record layout specifies calibration data serialization in the ECU memory which describes the shape of the characteristics. Single record layout can be referred by multiple instances of interpolation ParameterDataPrototype. Record layouts can be nested particular values refer to the particular property of the object. With different properties of record layouts it is possible to specify complex objects.

8.5.3.1 Record layout for map values

Due to optimization, the orientation of map values in memory is different depending on the usage of the inputs. See section 8.4.2.

1. If the "X" and "Y" inputs are not swapped then, values "Val" of maps have to be in COLUMN_DIR order.

2. If the "X" and "Y" inputs are swapped then, values "Val" of maps have to be in ROW_DIR order.

According to ASAM standard [ASAM MCD-2MC Version 1.5.1 and 1.6], COLUMN_DIR and ROW_DIR are formats of storing map values (Val[]) and more information can be found in ASAM standard.

The "Z" input of cuboids is the third dimension and selects the slice X / Y or Y / X - 2D maps.

Example for cuboids order:

2x2x2 cuboid representation in memory

Example: cub = [1 2 3 4 5 6 7 8]

COLUMN_DIR order [z][x][y]:

Slice 1:

[1 2

3 4]

Slice 2:

[5 6

7 8]

8.5.3.2 Record layout definitions

Below table specifies record layouts supported for interpolation routines.

[SWS_Ifl_00170] ⌈

Record layout Name	Element1	Element2	Element3	Element4	Element5
Distr_f32	uint32 N	float32 X[]			
Curve_f32	float32 Val[]				
Map_f32	float32 Val[]				
Cub_f32	float32 Val[]				
IntCurve_f32_f32	uint32 N	float32 X[]	float32 Val[]		
IntMap_f32f32_f32	uint32 Nx	uint32 Ny	float32 X[]	float32 Y[]	float32 Val[]

⌋

Remark:

All combinations have to be defined in IFL_RecordLayout_Blueprint, AUTOSAR_MOD_IFL_RecordLayout_Blueprint.arxml

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Ifl_GetVersionInfo

[SWS_Ifl_00215] Definition of API function Ifl_GetVersionInfo

Upstream requirements: SRS_BSW_00407, SRS_BSW_00003, SRS_BSW_00318, SRS_BSW_-00321

⌈

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

▽

△

Description	Returns the version information of this library.
Available via	Ifl.h

⌋

The version information of a BSW module generally contains:

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

[SWS_Ifl_00216]

Upstream requirements: SRS_BSW_00407, SRS_BSW_00411

⌈If source code for caller and callee of Ifl_GetVersionInfo is available, the Ifl library should realize Ifl_GetVersionInfo as a macro defined in the module's header file.⌋

8.8 Callback notifications

None.

8.9 Scheduled routines

The Ifl library does not have scheduled routines.

8.10 Expected interfaces

None.

8.10.1 Mandatory interfaces

None.

8.10.2 Optional interfaces

None.

8.10.3 Configurable interfaces

None.

9 Sequence diagrams

Not applicable.

10 Configuration specification

10.1 Published Information

[SWS_Ifl_00214]

Upstream requirements: SRS_BSW_00402, SRS_BSW_00374, SRS_BSW_00379

⌈The standardized common published parameters as required by SRS_BSW_00402 in the General Requirements on Basic Software Modules [4] 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 [3].⌋

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

10.2 Configuration option

[SWS_Ifl_00218]

Upstream requirements: SRS_LIBS_00001

⌈The Ifl 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.⌋

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

A Not applicable requirements

[SWS_Ifl_00224]

Upstream requirements: SRS_BSW_00448

⌈These requirements are not applicable to this specification.⌋

B Change history of AUTOSAR traceable items

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

B.1 Traceable item history of this document according to AUTOSAR Release R23-11

B.1.1 Added Specification Items in R23-11

none

B.1.2 Changed Specification Items in R23-11

Number	Heading
[SWS_Ifl_00005]	Definition of datatype Ifl_DPResultF32_Type
[SWS_Ifl_00021]	Definition of API function Ifl_IpoCur_f32
[SWS_Ifl_00025]	Definition of API function Ifl_IpoMap_f32
[SWS_Ifl_00030]	Definition of API function Ifl_Interpolate_f32
[SWS_Ifl_00035]	Definition of API function Ifl_IntIpoCur_f32_f32
[SWS_Ifl_00041]	Definition of API function Ifl_IntIpoMap_f32f32_f32
[SWS_Ifl_00170]	
[SWS_Ifl_91000]	Definition of API function Ifl_IpoCub_f32
[SWS_Ifl_91001]	Definition of API function Ifl_IpoCur_<InTypeMn>_f32
[SWS_Ifl_91002]	Definition of API function Ifl_IpoMap_<InTypeMn>_f32
[SWS_Ifl_91003]	Definition of API function Ifl_IpoCub_<InTypeMn>_f32

Table B.1: Changed Specification Items in R23-11

B.1.3 Deleted Specification Items in R23-11

none

B.2 Traceable item history of this document according to AUTOSAR Release R24-11

B.2.1 Added Specification Items in R24-11

Number	Heading
[SWS_Ifl_91004]	Definition of imported datatypes of module Ifl

Table B.2: Added Specification Items in R24-11

B.2.2 Changed Specification Items in R24-11

Number	Heading
[SWS_Ifl_00214]	

Table B.3: Changed Specification Items in R24-11

B.2.3 Deleted Specification Items in R24-11

none