Document Title	Specification of Floating Point Interpolation Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	398
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
12.12.2011	1.2.0	AUTOSAR Administration	• Error classification support and definition removed as DET call not supported by library • Configuration parameter description / support removed for XXX_GetVersionInfo routine. • XXX_GetVersionInfo routine name corrected.
15.11.2010	1.1.0	AUTOSAR Administration	• DPSearch function optimised using structure pointer • Removal of normalised functions
07.12.2009	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

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

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

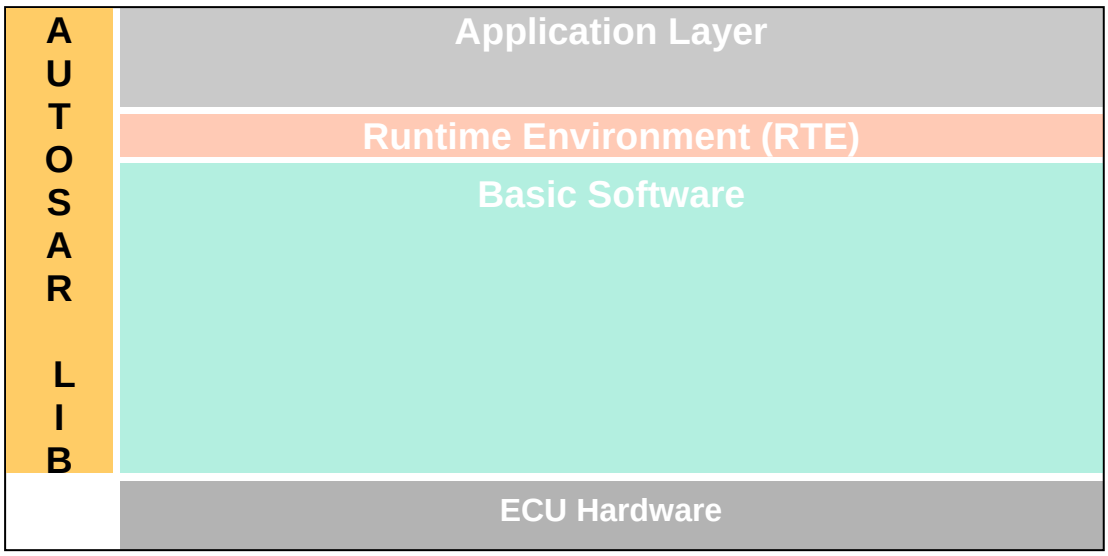

Figure : Layered architecture

This specification specifies the functionality, API and the configuration of the AUTOSAR library dedicated to 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	Development 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

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] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[8] Requirement on Libraries,
AUTOSAR_SRS_Libraries.pdf

[9] Specification of Memory Mapping,
AUTOSAR_SWS_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

[IFL001] ⌈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_'.
- Header file Ifl.h provides all public function prototypes and types defined by the Ifl library specification ⌋(BSW31400005)

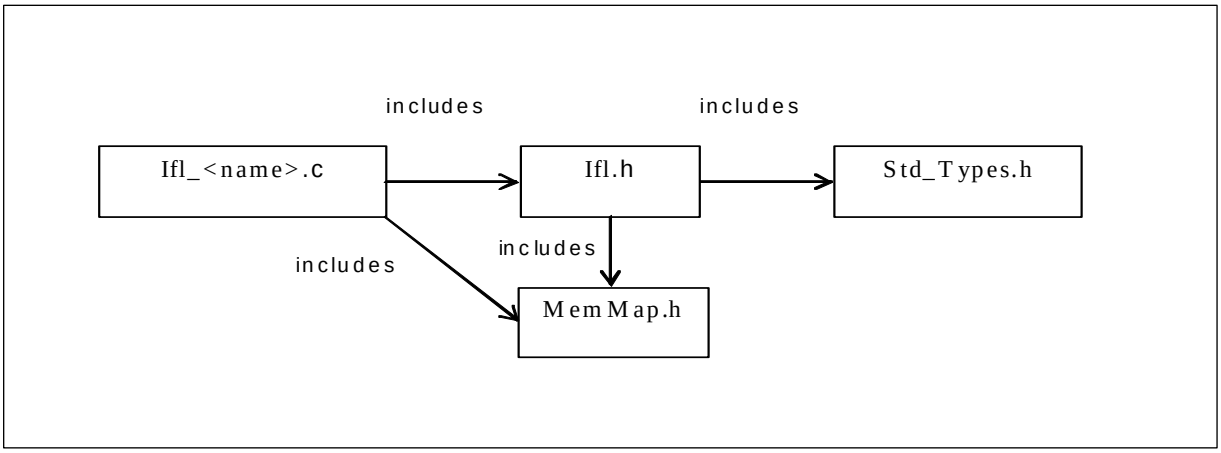

Figure : File structure

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.: 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 traceability

Requirement	Description	Satisfied by
BSW003		IFL215
BSW00304	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	IFL212
BSW00306	All AUTOSAR library Modules should avoid direct use of compiler and platform specific keyword, un...	IFL213
BSW00318		IFL215
BSW00321		IFL215
BSW00348	Each AUTOSAR library Module implementation *.	IFL211
BSW00378	All AUTOSAR library Modules should use the AUTOSAR data types (integers, boolean) instead of nati...	IFL212
BSW00407		IFL215, IFL216
BSW00411		IFL216
BSW00436	Each AUTOSAR library Module implementation *.	IFL210
BSW007	The library, written in C programming language, should conform to the HIS subset of the MISRA C S...	IFL209
BSW31400002	Ifl library shall not require initialization phase.	IFL200
BSW31400003	Ifl library shall not require a shutdown operation phase.	IFL201
BSW31400005	The Ifl module shall provide the following files:	IFL001
BSW31400013	Error detection: Function should check at runtime (both in production and development code) the v...	IFL219, IFL217
BSW31400015	The Ifl library shall be implemented in a way that the code can be shared among callers in differ...	IFL206
BSW31400017	Usage of macros should be avoided.	IFL207
BSW31400018	A library function can call other library functions because all library functions shall be re-ent...	IFL208

7 Functional specification

7.1 Error classification

[IFL223] ⌈No error classification definition as DET call not supported by library⌋ ( )

7.2 Error detection

[IFL219] ⌈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. ⌋(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

[IFL217] ⌈The functions shall not call the DET for error notification. ⌋ (BSW31400013)

7.4 Initialization and shutdown

[IFL200] ⌈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. ⌋(BSW31400002)

[IFL201] ⌈Ifl library shall not require a shutdown operation phase. ⌋(BSW31400003)

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-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 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

[IFL206] ⌈The Ifl library shall be implemented in a way that the code can be shared among callers in different memory partitions. ⌋(BSW31400015)

[IFL207] ⌈Usage of macros should be avoided. The function should be declared as function or inline function. Macro #define should not be used. ⌋(BSW31400017)

[IFL208] ⌈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. ⌋(BSW31400018)

[IFL209] ⌈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)

[IFL210] ⌈Each AUTOSAR library Module implementation <library>*.c and <library>*.h shall map their code to memory sections using the AUTOSAR memory mapping mechanism. ⌋(BSW00436)

[IFL211] ⌈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)

[IFL212] ⌈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)

[IFL213] ⌈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)

[IFL220] ⌈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. ⌋( )

[IFL221] ⌈Axis distribution passed to Ifl routines shall have strong monotony sequence. ⌋( )

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	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 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

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 <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>).

8.2 Type definitions

[IFL005] ⌈
Structure definition:

Name:	`Ifl_DPResultF32_Type`		
Type:	`Structure`		
Element:	`uint32`	`Index`	Data point index
	`float32`	`Ratio`	Data point ratio
Description:	Structure used for data point search for index and ratio **IFL006:** Ifl_DPResultF32_Type structure shall not be read/write/modified by the user di-		

	rectly. 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.

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

where: X is the input value
x0 = data point before X
x1 = data point after X
y0 = value at x0
y1 = value at x1

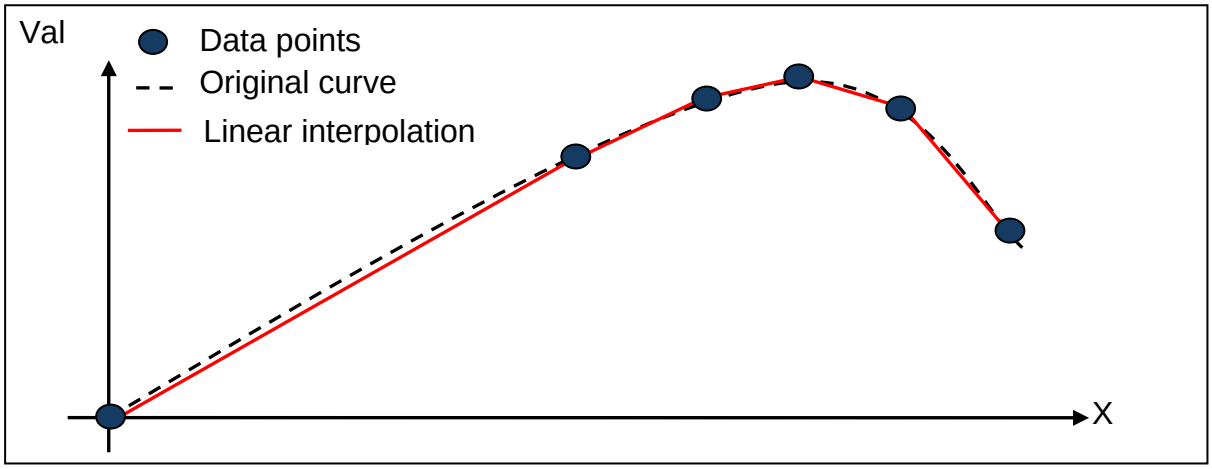

Figure : Linear interpolation

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

[IFL010] ⌈

Service name:	Ifl_DPSearch_f32	

Syntax:	`void Ifl_DPSearch_f32(`
	`  Ifl_DPResultF32_Type* const 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 (in-out):	None	
Parameters (out):	dpResult	Pointer to the result structure
Return value:	None	

Description:	**IFL011:**
	This routine searches the position of input Xin within the given distribution array X_array, and returns index and ratio necessary for interpolation.
	Returned Index shall be the lowest index for which (X_array[index] < Xin < X_array[index + 1]).
	
	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
	
	IFL012:
	Input value matches with one of the distribution array value 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
	
	IFL013:	

	If (Xin < X_array[0]), then return first index of an array and ratio = 0.0
	dpResult ->Index = 0
	dpResult ->Ratio = 0.0
	
	IFL014:
	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
	
	IFL015:
	The minimum value of N shall be 1
	
	IFL016:
	If X_array[Index+1] == X_array[Index], then the Ratio shall be zero.
	dpResult->Ratio = 0.0
	
	IFL01:
	This routine returns index and ratio through the structure of type Ifl_DPResultF32_Type

⌋ ( )

8.5.1.2 Curve interpolation

[IFL021] ⌈

Service name:	Ifl_IpoCur_f32	

Syntax:	`float32 Ifl_IpoCur_f32(`
	`  const Ifl_DPResultF32_Type* const 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 (in-out):	None	
Parameters (out):	None	
Return value:	float32	Result of the Interpolation
Description:	**IFL022:** Based on searched index and ratio information, this routine calculates and returns interpolation for curve. index = dpResult->Index Return value = Val_array[index] + (Val_array[index+1] - Val_array[index]) * dpResult->Ratio **IFL180:** Do not call this routine until you have searched the axis to ensure the search result contains valid data and is not used uninitialized.	

⌋ ( )

8.5.1.3 Map interpolation

[IFL025] ⌈

Service name:	Ifl_IpoMap_f32

Syntax:	`float32 Ifl_IpoMap_f32(`
	`  const Ifl_DPResultF32_Type* const dpResultX,`
	`  const Ifl_DPResultF32_Type* const 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 (in-out):	None	
Parameters (out):	None	
Return value:	float32	Result of the Interpolation
Description:	**IFL026:** Based on searched index and ratio information, this routine calculates and returns interpolation for map. BaseIndex = dpResultX->Index * num_value + dpResultY->Index 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 Z = LowerX + (UpperX - LowerX) * dpResultX->Ratio **IFL181:** Do not call this routine until you have searched the axis to ensure the search result contains valid data and is not used uninitialized.	

⌋ ( )

8.5.1.4 Single point interpolation

[IFL030] ⌈

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 (in-out):	None	
Parameters (out):	None	
Return value:	float32	Iterpolated value

Description:	**IFL031:** Returns the result of the linear interpolation (Result), determined according to the following equation. 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

[IFL035] ⌈

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 (in-out):	None	
Parameters (out):	None	
Return value:	float32	Result of the Interpolation

Description:	**IFL036:**
	This routine calculates interpolation of a curve at position Xin using below equation.
	
	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
	
	IFL037:
	Input value matches with one of the distribution array value then result shall be respective Y array element indicated by index.
	If (Xin == X_array[index])
	Result = Val_array[index]
	
	IFL038:
	If Xin < X_array[0], then
	Result = Val_array[0]
	
	IFL039:	

	If Xin > X_array[N-1], then Result = Val_array[N-1] **IFL040:** The minimum value of N shall be 1

⌋ ( )

8.5.2.2 Integrated map interpolation

[IFL041] ⌈

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 (in-out):	None	
Parameters (out):	None	
Return value:	float32	Result of the Map Interpolation
Description:	**IFL042:** This routine calculates Interpolation of a map at position X and Y using below equations. 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 Z = LowerX + (UpperX - LowerX) * RatioX **IFL043:** 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
	
	IFL044:
	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]) * RatioY
	
	IFL045:
	If (Xin == X_array[indexX]) and (Yin == Y_array[indexY])
	Result = Val_array [BaseIndex]
	
	IFL046:
	If Xin < X_array[0], then
	indexX = 0,
	RatioX = 0.0
	
	IFL047:
	If Xin > X_array[Nx-1], then
	indexX = Nx - 1,
	RatioX = 0.0
	
	IFL048:
	If Yin < Y_array[0], then
	indexY = 0,
	RatioY = 0.0
	
	IFL049:
	If Yin > Y_array[Ny-1], then
	indexY = Ny - 1,
	RatioY = 0.0
	
	IFL050:
	The minimum value of N shall be 1

⌋ ( )

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 CalprmElementPrototype. 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 definitions

Below table specifies record layouts supported for interpolation routines.

[IFL170] ⌈

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

⌋ ( )

8.6 Examples of use of functions

None

8.7 Version API

8.7.1 Ifl_GetVersionInfo

[IFL215] ⌈

Service name:	Ifl_GetVersionInfo	

Syntax:	`void Ifl_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)

[IFL216] ⌈
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. ⌋ (BSW00407, BSW00411)

8.8 Call-back 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

[IFL214] ⌈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

[IFL218] ⌈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.⌋(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

[IFL224] ⌈These requirements are not applicable to this specification.⌋ ( )