VECTOR

MICROSAR Classic IP Base

Technical Reference

IP Base Module

Version 1.05.01

Authors	Alex Lunkenheimer
Status	Released

Document Information

History

Author	Date	Version	Remarks
visalr	2011-05-20	1.0	Creation of the document
visalr	2011-12-05	1.1	Sock subcomponent added
visalr	2011-12-05	1.01.01	Released
visalr	2013-03-12	1.01.02	Update
visalr	2014-01-03	1.01.03	- New API IpBase_CalcTcpIpChecksum32 - Review integration
visalr	2014-01-03	1.01.04	- New API IpBase_CalcTcpIpChecksumAdd - Review integration
visalr	2014-02-07	1.01.05	- New API IpBase_CalcTcpIpChecksumAdd replaces IpBase_CalcTcpIpChecksum32
visalr	2014-02-07	1.01.06	- AUTOSAR version dependent architecture in 2.1 Architecture Overview
visalr	2015-02-27	1.01.07	- Adapted struct IpBase_SockAddrIn6Type and define IPBASE_AF_INET6
visalr	2015-03-02	1.02.00	- IpBase_CopySmallData introduced
visalr	2015-05-06	1.02.01	- IpBase_Copy as macro
visalr	2015-05-06	1.02.02	- Review integration
visalr	2017-11-07	1.03.00	- Remove IpBase_Copy.c - Remove all configuration options but DevErrorDetect - Add IpBase_Ber.c
visalr	2018-04-18	1.03.01	- Review integration
visalr	2018-10-19	1.03.02	- Enhance calling context to ISR for string and pbuf handling - minor adaptions for 4.00.02 implementation
visalr	2019-07-24	1.03.03	- Enhance put and get inline functions by 24, 48 and 64-bit access
visalr	2022-02-07	1.03.04	- chapter Template File Integration added
visalr	2022-05-02	1.04.00	- DaVinci Configurator Pro based configuration of IpBase_Cfg.h
visese	2023-04-18	1.05.00	- Add APIs IpBase_TcpIpChecksumAdd, IpBase_TcpIpChecksumCopyAdd
visese	2023-07-04	1.05.01	- Adapt description of TcpIpChecksumCopyAdd

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DET.pdf	2.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	2.2.0
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[4]	AUTOSAR	AUTOSAR_TR_SecureHardwareExtensions.pdf	R19-11

Scope of the Document

This technical reference describes the general use of the IpBase base software.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00.xx	Initial component version
1.01.xx	Extension by string length
1.02.xx	Data types adapted, ASN.1 / BER decoder added, Bug fixing
2.00.xx	Adapted struct IpBase_SockAddrIn6Type and define IPBASE_AF_INET6
2.01.xx	IpBase_Copy as macro from VStdLib (performance improvement)
3.00.xx	Process improvements
4.00.xx	ISO26262 compliance (ASIL-B)

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW module IpBase as specified in [1].

Supported AUTOSAR Release:	not relevant	
Supported Configuration Variants:	not relevant	
Vendor ID:	IpBase_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	IpBase_MODULE_ID	255 decimal (according to ref. [4])

The IpBase component provides general functions used within MICROSAR Classic IP. Its functionality covers copy, buffer and string handling as well as type definitions.

2.1 Architecture Overview

The following figure shows where the IpBase is located in the AUTOSAR architecture.

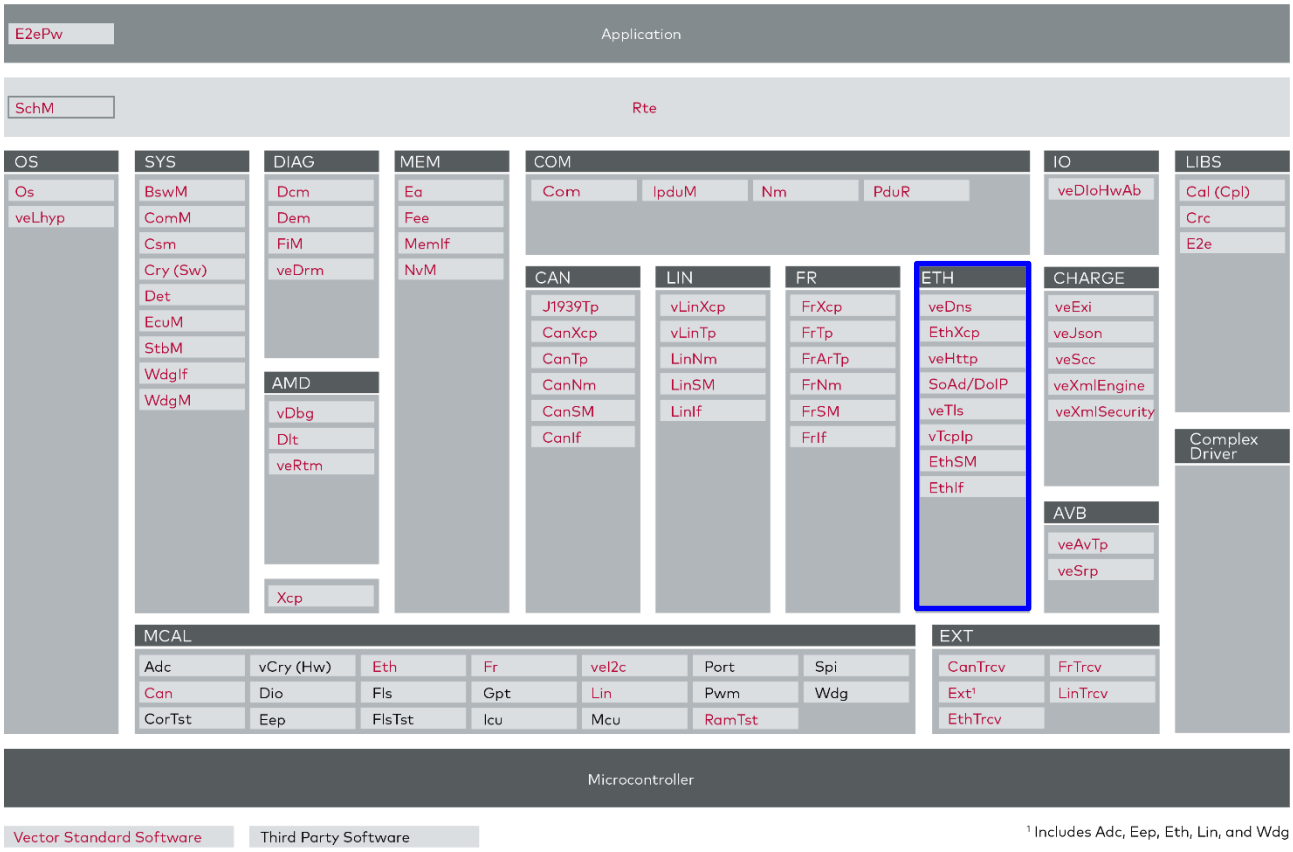

Figure 2-1 AUTOSAR 4.x Architecture Overview

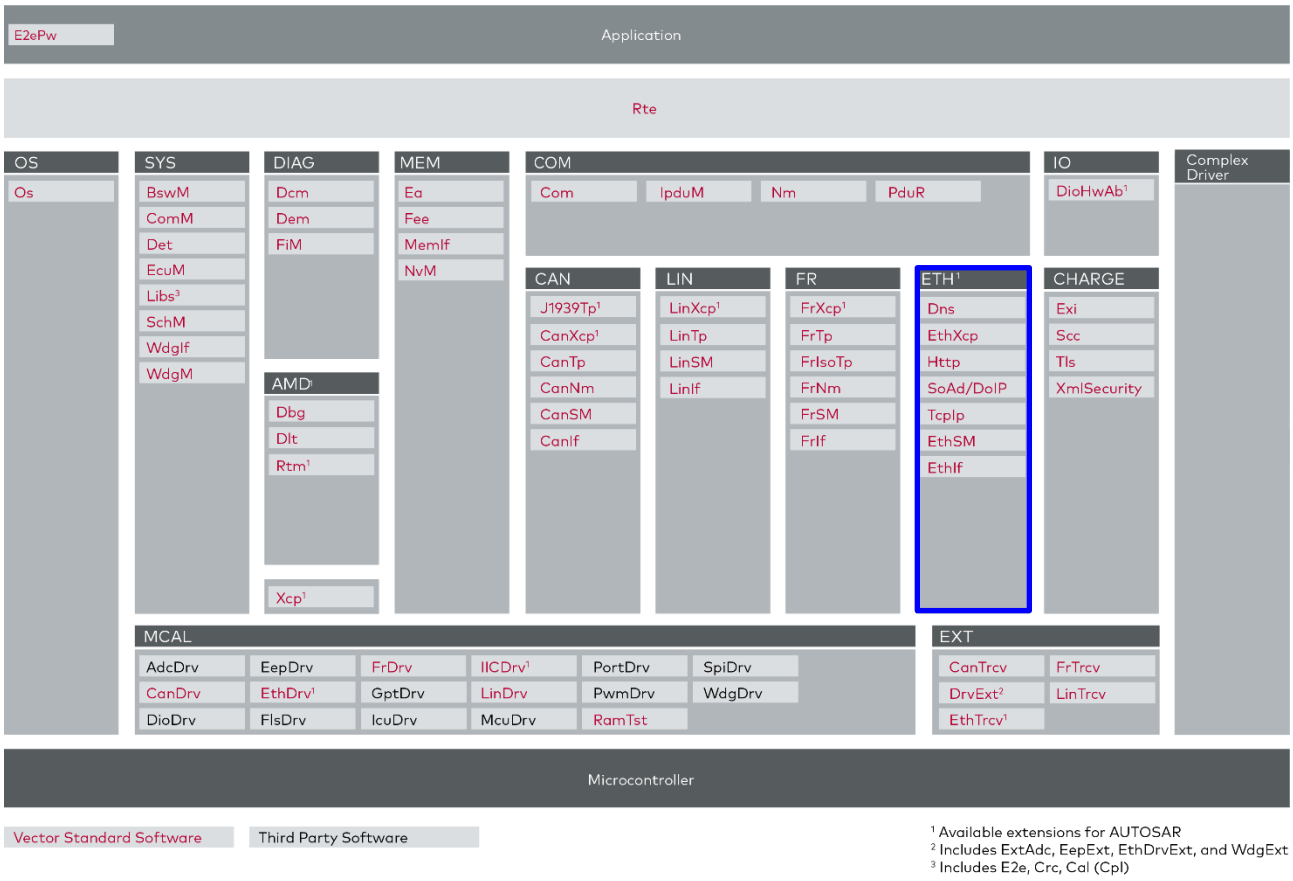

Figure 2-2 AUTOSAR 3.x Architecture Overview

The next figure shows the interfaces of IpBase provided to its users. These interfaces are described in chapter 5.

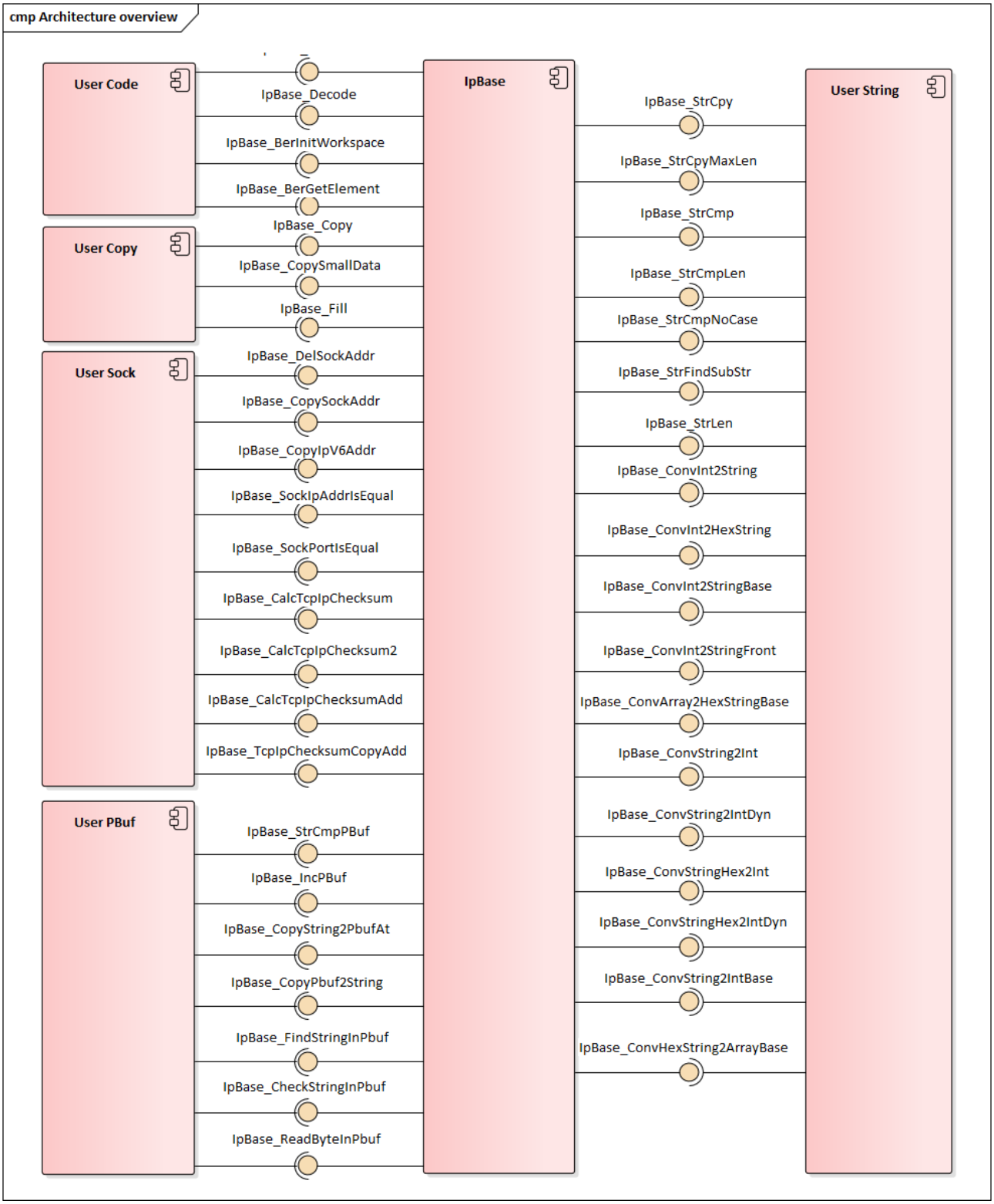

Figure 2-3 Interfaces of IpBase

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality provided by the module.

The "supported" features are presented in the following table.

The following features are supported:

Supported Feature
Base64 and ASN.1 (BER) encoding and decoding
Generic base copy
Linked buffer handling
String handling (copy, compare, conversion)
Socket handling (compare, copy, reset and checksum calculation)

Table 3-1 Supported IpBase features

3.2 Initialization

The IpBase component does not require initialization.

3.3 States

The IpBase component is always operational.

3.4 Main Functions

The IpBase does not provide a main function.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `IPBASE_DEV_ERROR_REPORT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator but must have the same signature as the service `Det_ReportError()`.

The reported IpBase ID is 255, the instance ID is 110.

The reported service IDs identify the services which are described in chapter 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01	`IpBase_GetVersionInfo`
0x11	`IpBase_Encode`
0x12	`IpBase_Decode`

Service ID	Service
0x13	IpBase_BerInitWorkspace
0x14	IpBase_BerGetElement
0x31	IpBase_StrCopy
0x32	IpBase_StrCopyMaxLen
0x33	IpBase_StrCmp
0x34	IpBase_StrCmpLen
0x35	IpBase_StrCmpNoCase
0x36	IpBase_StrCmpLenNoCase
0x37	IpBase_StrFindSubStr
0x38	IpBase_StrLen
0x39	IpBase_ConvInt2String
0x3A	IpBase_ConvInt2HexString
0x3B	IpBase_ConvInt2StringBase
0x3C	IpBase_ConvArray2HexStringBase
0x3D	IpBase_ConvInt2StringFront
0x3E	IpBase_ConvString2Int
0x3F	IpBase_ConvString2IntDyn
0x40	IpBase_ConvHexString2Int
0x41	IpBase_ConvHexString2IntDyn
0x42	IpBase_ConvString2IntBase
0x43	IpBase_ConvString2SignedIntBase
0x44	IpBase_ConvHexString2ArrayBase
0x51	IpBase_StrCmpPbuf
0x52	IpBase_IncPbuf
0x53	IpBase_CopyString2PBufAt
0x54	IpBase_CopyPbuf2String
0x55	IpBase_FindStringInPbuf
0x56	IpBase_ChkStringInPbuf
0x57	IpBase_ReadByteInPbuf
0x60	IpBase_DelSockAddr
0x61	IpBase_CopySockAddr
0x62	IpBase_CopyIpV6Addr
0x63	IpBase_SockIpAddrIsEqual
0x64	IpBase_SockPortIsEqual
0x65	IpBase_CalcTcpIpChecksum
0x66	IpBase_CalcTcpIpChecksum2
0x67	IpBase_CalcTcpIpChecksumAdd
0x68	IpBase_TcpIpChecksumAdd

Service ID	Service
0x69	IpBase_TcpIpChecksumCopyAdd

Table 3-2 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x01	IPBASE_E_INV_POINTER	Invalid pointer
0x02	IPBASE_E_INV_PARAM	Invalid parameter

Table 3-3 Errors reported to DET

3.5.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled separately. The configuration of en-/disabling the checks is described in chapter 6.2. En-/disabling of single checks is an addition to the AUTOSAR standard which requires to en-/disable the complete parameter checking via the parameter IPBASE_DEV_ERROR_DETECT.

The following table shows which parameter checks are performed on which services:

Service \ Check	IPBASE_E_INV_POINTER	IPBASE_E_INV_PARAM
IpBase_GetVersionInfo	■	
IpBase_Encode	■	
IpBase_Decode	■	
IpBase_BerInitWorkspace	■	
IpBase_BerGetElement	■	
IpBase_Copy		
IpBase_Fill		
IpBase_StrCpy	■	
IpBase_StrCpyMaxLen	■	
IpBase_StrCmp	■	
IpBase_StrCmpLen	■	
IpBase_StrCmpNoCase	■	
IpBase_StrCmpLenNoCase	■	
IpBase_StrFindSubStr	■	■
IpBase_StrLen	■	
IpBase_ConvInt2String	■	

Service \ Check	IPBASE_E_INV_POINTER	IPBASE_E_INV_PARAM
IpBase_ConvInt2HexString	■	
IpBase_ConvInt2StringBase	■	
IpBase_ConvArray2HexStringBase	■	
IpBase_ConvInt2StringFront	■	
IpBase_ConvString2Int	■	■
IpBase_ConvString2IntDyn	■	
IpBase_ConvStringHex2Int	■	■
IpBase_ConvStringHex2IntDyn	■	
IpBase_ConvString2IntBase	■	
IpBase_ConvString2SignedIntBase	■	
IpBase_ConvHexString2ArrayBase	■	
IpBase_StrCmpPBuf	■	
IpBase_IncPBuf	■	
IpBase_CopyString2PbufAt	■	■
IpBase_CopyPbuf2String	■	■
IpBase_FindStrInPbuf	■	■
IpBase_CheckStringInPbuf	■	■
IpBase_ReadByteInPbuf	■	■
IpBase_DelSockAddr	■	
IpBase_CopySockAddr	■	
IpBase_CopyIpV6Addr	■	
IpBase_SockIpAddrIsEqual	■	
IpBase_SockPortIsEqual	■	
IpBase_CalcTcpIpChecksum	■	
IpBase_CalcTcpIpChecksum2	■	
IpBase_CalcTcpIpChecksumAdd	■	

Table 3-4 Development Error Reporting: Assignment of checks to services

3.5.2 Production Code Error Reporting

Not used.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR Classic IpBase into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the IpBase contains the files which are described in the chapters 4.1.1 and 4.1.2.

4.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
IpBase.a		■	IpBase library.
IpBase.c	■		Static source for core.
IpBase.h	■	■	Static header for API.
IpBase_Ber.c	■		Static source for basic encoding rules.
IpBase_Ber.h	■	■	Static header for basic encoding rules.
IpBase_Code.c	■		Static source for coding.
IpBase_Code.h	■	■	Static header for coding.
IpBase_Copy.h	■	■	Static header for copy.
IpBase_PBuf.c	■		Static source for buffer.
IpBase_PBuf.h	■	■	Static header for buffer.
IpBase_Sock.c	■		Static source for socket.
IpBase_Sock.h	■	■	Static header for socket.
IpBase_String.c	■		Static source for string.
IpBase_String.h	■	■	Static header for string.
IpBase_Priv.h	■	■	Static header for internal macro and variable declaration.
IpBase_Types.h	■	■	Static header for type definitions.
_Appl_Rand.c	■	■	Template static source for random functions.
_Appl_Rand.h	■	■	Template static header for random functions.
_Appl_Time.c	■	■	Template static source for time functions.
_Appl_Time.h	■	■	Template static header for time functions.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator Pro.

File Name	Description
IpBase_Cfg.h	Generated header file for pre-compile time configuration data.

Table 4-2 Generated files

4.2 Include Structure

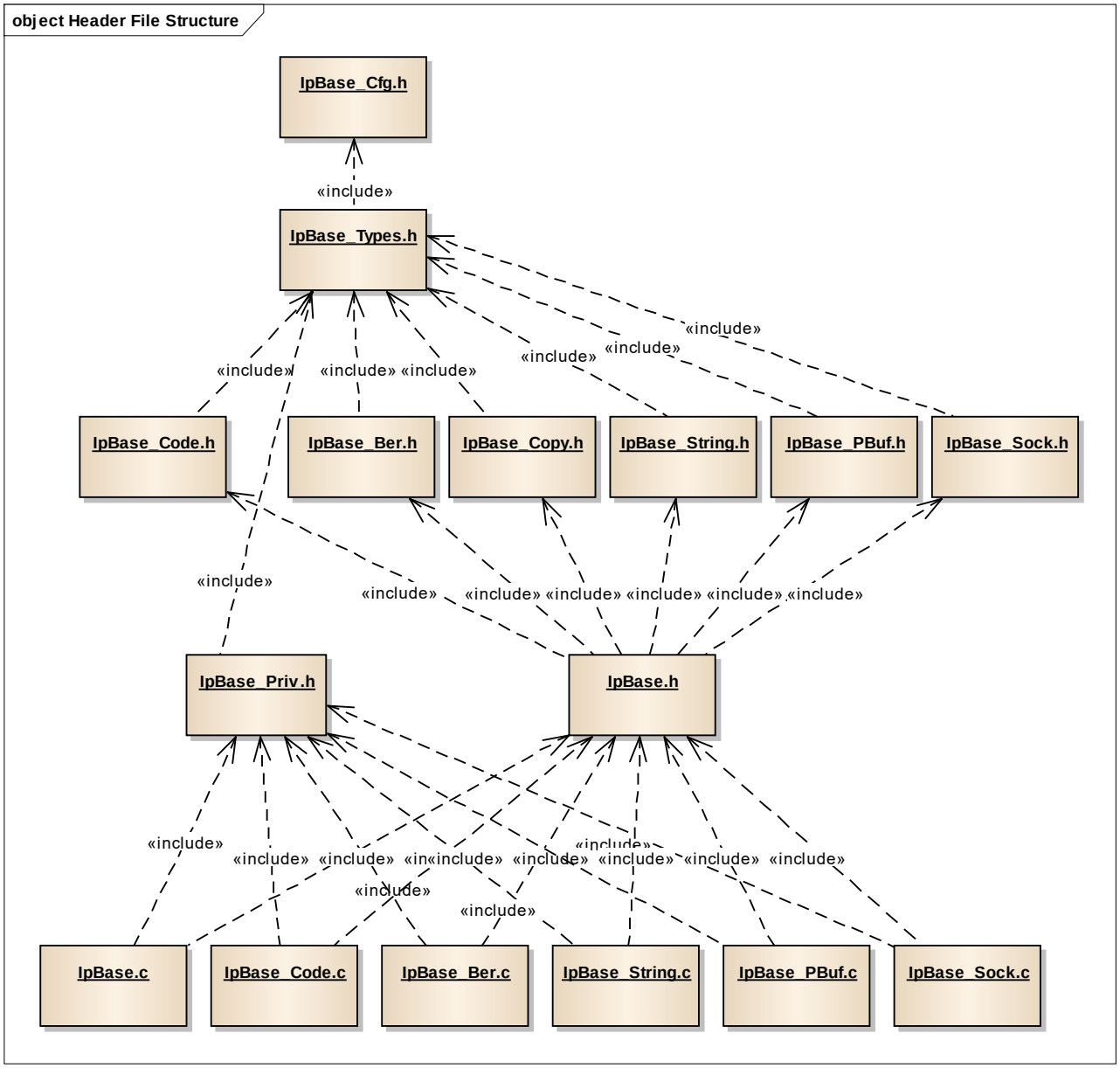

Figure 4-1 Include structure

4.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions of the IpBase and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	IPBASE_VAR_NOINIT	IPBASE_CONST	IPBASE_CODE	IPBASE_PBCFG
IPBASE_START_SEC_PBCFG IPBASE_STOP_SEC_PBCFG				■
IPBASE_START_SEC_CODE IPBASE_STOP_SEC_CODE			■	
IPBASE_START_SEC_CONST_UNSPECIFIED IPBASE_STOP_SEC_ CONST_UNSPECIFIED		■		
IPBASE_START_SEC_CONST_32BIT IPBASE_STOP_SEC_ CONST_32BIT		■		
IPBASE_START_SEC_CONST_16BIT IPBASE_STOP_SEC_ CONST_16BIT		■		
IPBASE_START_SEC_CONST_8BIT IPBASE_STOP_SEC_ CONST_8BIT		■		
IPBASE_START_SEC_VAR_NOINIT_UNSPECIFIED IPBASE_STOP_SEC_VAR_NOINT_UNSPECIFIED	■			
IPBASE_START_SEC_VAR_NOINIT_32BIT IPBASE_STOP_SEC_VAR_NOINT_32BIT	■			
IPBASE_START_SEC_VAR_NOINIT_16BIT IPBASE_STOP_SEC_VAR_NOINT_16BIT	■			
IPBASE_START_SEC_VAR_NOINIT_8BIT IPBASE_STOP_SEC_VAR_NOINT_8BIT	■			

Table 4-3 Compiler abstraction and memory mapping

4.4 Critical Sections

Currently, no critical sections are used.

4.5 Template File Integration

4.5.1 Random Template Integration

The file _Appl_Rand.c provides functions which have to be implemented by the customer. The delivered content compiles and links the code to ease integration.

The function ApplRand_Init requires initialization with unique and changing numbers. Examples are provided in the comments. Ideally, a superposition of multiple arbitrary values is used. The function can be used to provide a seed and store it for later use of ApplRand_GetRandNo.

The function ApplRand_GetRandNo requires provisioning of a real random number generator (RNG). Such a random number can be either created in software with a seed or from hardware (HRNG).

Please refer to [4] for details on random numbers in AUTOSAR.

Caution
The delivered templates may not be used in series production. The random function does not provide real random numbers but always the same sequence. The customer has to implement the functions for security reasons.

4.5.2 Time Template Integration

The file _Appl_Time.c provides functions which have to be implemented by the customer. The delivered content compiles and links the code to ease integration.

The function ApplTime_GetTime32 requires provisioning of the time in seconds since 1970.

Caution
The delivered templates may not be used in series production. The time function does not provide the current time but always the same time. The customer has to implement the functions for security reasons.

5 API Description

For interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the IpBase are described in this chapter.

Type Name	C-Type	Description	Value Range
IpBase_AddrInType	uint32	Type used for IP addresses	0x00000000 – 0xFFFFFFFF
IpBase_IPAddressType	uint32	Type used for IP addresses (limited to IPv4)	0x00000000 – 0xFFFFFFFF
IpBase_CopyDataType	uint8	Type used for copy routines	0x00 – 0xFF
IpBase_FamilyType	uint16	Type used for IP address family	0x04 or 0x06
IpBase_PortType	uint16	Type used for port	0x0000 - 0xFFFF
IpBase_EthPhysAddrType	uint8[]	Array used for Ethernet physical address (MAC address)	0x000000000000 – 0xFFFFFFFFFFFF
IpBase_SockIdxType	uint8	Type used for socket index	0x00 – 0xFF
IpBase_ReturnType	uint8	Type used for return values	IPBASE_E_OK 0x00 IPBASE_E_NOT_OK 0x81 IPBASE_E_PENDING 0x82 IPBASE_E_MEM 0x83 IPBASE_E_BER_PARAM 0x84
IpBase_TcpIpEventType	uint8	Type used for TCP/IP events	IPBASE_TCP_EVENT_RESET 0x01 IPBASE_TCP_EVENT_CLOSED 0x02 IPBASE_TCP_EVENT_FIN_RECEIVED 0x03
IpBase_PbufType	struct	Type used for distributed buffers	Payload pointer, total length, segment length
IpBase_IpAddrPortType	struct	Type used for TCP/IP addressing (limited to IPv4)	Port, length, IPv4 address Hint: Type is deprecated due to limitation to IPv4
IpBase_SockAddrType	struct	Type used for TCP/IP addressing	Family, address data abstract base type for IpBase_SockAddrInType and IpBase_SockAddrIn6Type
IpBase_SockAddrInType	struct	Type used for TCP/IP addressing	Family, port, IPv4 address

Type Name	C-Type	Description	Value Range
IpBase_AddrIn6Type	Struct	Type used to store IPv6 address	IPv6 address
IpBase_SockAddrIn6Type	Struct	Type used to store socket address	Family, port, IPv6 address

Table 5-1 Type definitions

IpBase_PbufType

This type is used as array with 1 to n buffer segments where sum of all segment lengths equals total length.

Struct Element Name	C-Type	Description	Value Range
payload	uint8 *	Pointer to the payload data, i.e. buffer segment	0 Lowest memory address
			? ighest memory address
totLen	uint32	Total length in bytes of all buffer segments	0 Lowest memory address
			? Highest memory address
len	uint16	Length in bytes of all buffer segments	0 Lowest memory address
			? Highest memory address

Table 5-2 IpBase_PbufType

IpBase_IpAddrPortType

This type is used as internet protocol socket address with IPv4 address, length and port number.

Struct Element Name	C-Type	Description	Value Range
port	uint16	Port number of internet protocol	0 Port 0
			65535 Port 65535
len	uint8	Length in bytes of address	4 IPv4 address
			16 IPv6 address
addr	uint32	IPv4 internet protocol address	0 Invalid address
			4294967295

Struct Element Name	C-Type	Description	Value Range
			Broadcast address

Table 5-3 IpBase_IpAddrPortType

IpBase_SockAddrType

This type is used as internet protocol socket address with IPv4 address, length and port number.

Struct Element Name	C-Type	Description	Value Range
sa_family	uint16	Internet protocol family (IPv4 or IPv6)	0 Unspecified address family
			65535 Invalid address family
sa_data	uint8[]	Data with dummy length superseded with IPv4 or IPv6 address	0 IPv4 or IPv6 address
			0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF IPv6 broadcast address

Table 5-4 IpBase_SockAddrType

IpBase_SockAddrInType

This type is used as internet protocol socket address with IPv4 address, length and port number.

Struct Element Name	C-Type	Description	Value Range
sin_family	uint16	Internet protocol family (IPv4 or IPv6)	0 Unspecified address family
			65535 Invalid address family
sin_port	uint16	Port number of internet protocol	0 Port 0
			65535 Port 65535
sin_addr	uint32	IPv4 internet protocol address	0 Invalid address
			4294967295 Broadcast address

Table 5-5 IpBase_SockAddrInType

IpBase_AddrIn6Type

This type is used as internet protocol IPv4 or IPv6 address.

Struct Element Name	C-Type	Description	Value Range
addr	uint8[]	Internet protocol family (IPv4 or IPv6)	0 Invalid address
			0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF Broadcast IPv6 address
addr32	uint32[]	Internet protocol family (IPv4 or IPv6)	0 Invalid address
			0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF Broadcast IPv6 address

Table 5-6 IpBase_AddrIn6Type

IpBase_SockAddrIn6Type

This type is used as internet protocol socket address with family plus IPv6 address plus port number.

Struct Element Name	C-Type	Description	Value Range
sin6_family	uint16	Internet protocol family (IPv6)	0 Unspecified address family
			65535 Invalid address family
sin6_port	uint16	Port number of internet protocol	0 Port 0
			65535 Port 65535
sin6_addr	IpBase_AddrIn6Type	IPv6 internet protocol address	0 Invalid address
			0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF Broadcast address

Table 5-7 IpBase_SockAddrIn6Type

5.2 Services provided by IpBase

The IpBase API consists of services, which are realized by function calls or Macros.

5.2.1 IpBase_GetVersionInfo

Prototype	
`void IpBase_GetVersionInfo (Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
`VersionInfoPtr`	Pointer for version information
Return code	
void	none
Functional Description	
Get version information. Returns version information, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
task	

Table 5-8 IpBase_GetVersionInfo

5.2.2 IPBASE_BYTE_SWAP16

Prototype	
`IPBASE_BYTE_SWAP16 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	16bit unsigned integer with all bytes swapped (1,2->2,1)
Functional Description	
Swaps the bytes of a 16bit unsigned integer. The sequence of all bytes is swapped.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-9 IPBASE_BYTE_SWAP16

5.2.3 IPBASE_BYTE_SWAP32

Prototype	
`IPBASE_BYTE_SWAP32 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	32bit unsigned integer with all bytes swapped (1,2,3,4->4,3,2,1)
Functional Description	
Swaps the bytes of a 32bit unsigned integer. The sequence of all bytes is swapped.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-10 IPBASE_BYTE_SWAP32

5.2.4 IPBASE_HTON16

Prototype	
`IPBASE_HTON16 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in network byte order
Functional Description	
Swaps all bytes of a 16bit unsigned integer from host to network byte order, i.e. swaps if host is big endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-11 IPBASE_HTON16

5.2.5 IPBASE_HTON32

Prototype	
`IPBASE_HTON32 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in network byte order
Functional Description	
Swaps all bytes of a 32bit unsigned integer from host to network byte order, i.e. swaps if host is big endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-12 IPBASE_HTON32

5.2.6 IPBASE_NTOH16

Prototype	
`IPBASE_NTOH16 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in host byte order
Functional Description	
Swaps all bytes of a 16bit unsigned integer from network to host byte order, i.e. swaps if host is big endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-13 IPBASE_NTOH16

5.2.7 IPBASE_NTOH32

Prototype	
`IPBASE_NTOH32 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in host byte order
Functional Description	
Swaps all bytes of a 32bit unsigned integer from network to host byte order, i.e. swaps if host is big endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-14 IPBASE_NTOH32

5.2.8 IPBASE_LE2HE16

Prototype	
`IPBASE_LE2HE16 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in host byte order
Functional Description	
Swaps all bytes of a 16bit unsigned integer from little endian to host endian, i.e. swaps if host is little endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-15 IPBASE_LE2HE16

5.2.9 IPBASE_LE2HE32

Prototype	
`IPBASE_LE2HE32 (Data)`	
Parameter	
Data	Value in original byte order
Return code	
Data	Bytes in host byte order
Functional Description	
Swaps all bytes of a 32bit unsigned integer from little endian to host endian, i.e. swaps if host is little endian.	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro.	
Expected Caller Context	
interrupt or task level	

Table 5-16 IPBASE_LE2HE32

5.2.10 IpBase_ByteSwap16

Prototype	
`IpBase_ByteSwap16 (uint16 Data)`	
Parameter	
Data	Value in original byte order
Return code	
uint16	16bit unsigned integer with all bytes swapped (1,2->2,1)
Functional Description	
Swaps the bytes of a 16bit unsigned integer. The sequence of all bytes is swapped.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-17 IpBase_ByteSwap16

5.2.11 IpBase_ByteSwap32

Prototype	
`IpBase_ByteSwap32 (uint32 Data)`	
Parameter	
Data	Value in original byte order
Return code	
uint32	32bit unsigned integer with all bytes swapped (1,2,3,4->4,3,2,1)
Functional Description	
Swaps the bytes of a 32bit unsigned integer. The sequence of all bytes is swapped.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-18 IpBase_ByteSwap32

5.2.12 IpBase_ByteSwap64

Prototype	
`IpBase_ByteSwap64 (uint64 Data)`	
Parameter	
Data	Value in original byte order
Return code	
uint32	64bit unsigned integer with all bytes swapped (1,2,3,4,5,6,7,8->8,7,6,5,4,3,2,1)
Functional Description	
Swaps the bytes of a 64bit unsigned integer. The sequence of all bytes is swapped.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-19 IpBase_ByteSwap64

5.2.13 IpBase_PutUint8

Prototype	
`void IpBase_PutUint8 (uint8 *BufferPtr, uint32_least Offset, uint8 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written
Return code	
void	None
Functional Description	
Writes a one-byte unsigned integer into a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-20 IpBase_PutUint8

5.2.14 IpBase_PutUint16

Prototype	
`void IpBase_PutUint16 (uint8 *BufferPtr, uint32_least Offset, uint16 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written
Return code	
void	None
Functional Description	
Writes a two-byte unsigned integer into a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	

interrupt or task level

Table 5-21 IpBase_PutUint16

5.2.15 IpBase_PutUint24

Prototype	
`void IpBase_PutUint24 (uint8 *BufferPtr, uint32_least Offset, uint32 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written
Return code	
void	None
Functional Description	
Writes a three-byte unsigned integer into a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-22 IpBase_PutUint24

5.2.16 IpBase_PutUint32

Prototype	
`void IpBase_PutUint32 (uint8 *BufferPtr, uint32_least Offset, uint32 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written
Return code	
void	None
Functional Description	
Writes a four-byte unsigned integer into a buffer.	

Particularities and Limitations
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.
Expected Caller Context
interrupt or task level

Table 5-23 IpBase_PutUint32

5.2.17 IpBase_PutUint48

Prototype	
`void IpBase_PutUint48 (uint8 *BufferPtr, uint32_least Offset, uint64 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written
Return code	
void	None
Functional Description	
Writes a six-byte unsigned integer into a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-24 IpBase_PutUint48

5.2.18 IpBase_PutUint64

Prototype	
`void IpBase_PutUint64 (uint8 *BufferPtr, uint32_least Offset, uint64 Value)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Value	Value to be written

Return code	
void	None
Functional Description	
Writes an eight-byte unsigned integer into a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-25 IpBase_PutUint64

5.2.19 IpBase_GetUint8

Prototype	
`uint8 IpBase_GetUint8 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Return code	
uint8	Read value
Functional Description	
Reads a one byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-26 IpBase_GetUint8

5.2.20 IpBase_GetUint16

Prototype	
`uint16 IpBase_GetUint16 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer

Return code	
uint16	Read value
Functional Description	
Reads a two-byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-27 IpBase_GetUint16

5.2.21 IpBase_GetUint24

Prototype	
`uint32 IpBase_GetUint24 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Return code	
uint32	Read value
Functional Description	
Reads a three-byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-28 IpBase_GetUint24

5.2.22 IpBase_GetUint32

Prototype	
`uint32 IpBase_GetUint32 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer

Return code	
uint32	Read value
Functional Description	
Reads a four-byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-29 IpBase_GetUint32

5.2.23 IpBase_GetUint48

Prototype	
`uint64 IpBase_GetUint48 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer
Return code	
uint64	Read value
Functional Description	
Reads a six-byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-30 IpBase_GetUint48

5.2.24 IpBase_GetUint64

Prototype	
`uint64 IpBase_GetUint64 (uint8 *BufferPtr, uint32_least Offset)`	
Parameter	
BufferPtr	Pointer to the buffer
Offset	Byte offset within the buffer

Return code	
uint64	Read value
Functional Description	
Reads an eight-byte unsigned integer from a buffer.	
Particularities and Limitations	
> Service ID: implemented as inline function and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as inline function.	
Expected Caller Context	
interrupt or task level	

Table 5-31 IpBase_GetUint64

5.2.25 IpBase_Encode

Prototype	
`Std_ReturnType` **`IpBase_Encode`** `(uint8 Code, uint8 *TgtDataPtr, const uint8 *SrcDataPtr, uint32 *TgtLenBytePtr, uint32 SrcLenByte)`	
Parameter	
Code	defines the code used for encoding
TgtDataPtr	pointer for the encoded data
SrcDataPtr	pointer to the raw data
TgtLenBytePtr	pointer for the encoded data length in bytes
SrcLenByte	raw data length in bytes
Return code	
Std_ReturnType	E_OK data encoded E_NOT_OK encoding failed
Functional Description	
Encodes the given data using the specified code.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
task	

Table 5-32 IpBase_Encode

5.2.26 IpBase_Decode

Prototype	
`Std_ReturnType IpBase_Decode (uint8 Code, uint8 *TgtDataPtr, const uint8 *SrcDataPtr, uint32 *TgtLenBytePtr, uint32 SrcLenByte)`	
Parameter	
Code	defines the code used for decoding
TgtDataPtr	pointer for the decoded data
SrcDataPtr	pointer to the raw data
TgtLenBytePtr	pointer for the decoded data length in bytes
SrcLenByte	raw data length in bytes
Return code	
Std_ReturnType	E_OK data decoded E_NOT_OK decoding failed
Functional Description	
Decodes the given data using the specified code.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
task	

Table 5-33 IpBase_Decode

5.2.27 IpBase_BerInitWorkspace

Prototype	
`void IpBase_BerInitWorkspace (IpBase_BerWorkspaceType * const WorkspacePtr, IpBase_BerStackElementType * const StackPtr, const uint8 Depth)`	
Parameter	
WorkspacePtr	the workspace to initialize
StackPtr	the stack to use
Depth	the depth of the stack
Return code	
void	none
Functional Description	
Initializes the ASN.1/BER parser workspace.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different workspace parameter.
Expected Caller Context
task

Table 5-34 IpBase_BerInitWorkspace

5.2.28 IpBase_BerGetElement

Prototype
`IpBase_ReturnType` **`IpBase_BerGetElement`** `(IpBase_BerWorkspaceType * const WorkspacePtr, IpBase_BerElementType *ElementPtr, const uint8 *ElementNrPtr, const uint8 ElementDepth, const uint8 *DataPtr, const uint32 DataSize)`

Parameter	
WorkspacePtr	the internally used workspace
ElementPtr	the found element
ElementNrPtr	the element number (chapter.section.subsection. ...)
ElementDepth	the depth of the element (chapter = 1, chapter.section = 2, ...)
DataPtr	the data
DataSize	the size of the data

Return code	
IpBase_ReturnType	IPBASE_E_OK element found IPBASE_E_NOT_OK element not found IPBASE_E_INV_PARAM data corrupt IPBASE_E_MEM memory exceeded

Functional Description
Get an ASN.1/BER element with a given number out of ASN.1/BER encoded data.

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different workspace parameter.
Expected Caller Context
task

Table 5-35 IpBase_BerGetElement

5.2.29 IpBase_Copy

Prototype
`IpBase_Copy` `(TgtDataPtr, SrcDataPtr, LenByte)`

Parameter	
TgtDataPtr	pointer for target data
SrcDataPtr	pointer to source data
LenByte	data length in bytes
Return code	
void	none
Functional Description	
Copy data (memcpy).	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro and using VStdLib_MemCpy.	
Expected Caller Context	
interrupt or task level	

Table 5-36 IpBase_Copy

5.2.30 IpBase_Fill

Prototype	
`void IpBase_Fill (TgtDataPtr, Pattern, LenByte)`	
Parameter	
TgtDataPtr	pointer for target data
Pattern	fill pattern
LenByte	data length in bytes
Return code	
void	none
Functional Description	
Fill data (memset).	
Particularities and Limitations	
> Service ID: implemented as macro and thus no Service ID > This function is synchronous. > This function is reentrant. The function is implemented as macro and using VStdLib_MemSet.	
Expected Caller Context	
interrupt or task level	

Table 5-37 IpBase_Fill

5.2.31 IpBase_StrCmpPBuf

Prototype	
`uint8 IpBase_StrCmpPBuf (const IpBase_PbufType **SrcPBufPtr, const sint8 *PatternPtr, uint16 *CurByteIdxPtr, uint32 *TotByteIdxPtr, uint32 *RestLenBytePtr)`	
Parameter	
SrcPBufPtr	pointer to source data
PatternPtr	string pattern
CurByteIdxPtr	local start index
TotByteIdxPtr	total start index
RestLenBytePtr	unread sniplet
Return code	
uint8	IPBASE_CMP_EQUAL string pattern found IPBASE_CMP_NOT_EQUAL string pattern not found
Functional Description	
Compare string with PBuf.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. Supports PBuf, only for short string comparisons (bytewise access).	
Expected Caller Context	
interrupt or task level	

Table 5-38 IpBase_StrCmpPBuf

5.2.32 IpBase_IncPBuf

Prototype	
`void IpBase_IncPBuf (IpBase_PbufType **PBufPtr, uint16 *CurByteIdxPtr, uint32 *TotByteIdxPtr)`	
Parameter	
PBufPtr	pointer to PBuf struct
CurByteIdxPtr	pointer to current byte idx within PBuf struct
TotByteIdxPtr	pointer to total byte idx within PBuf struct
Return code	
void	none
Functional Description	
Increment PBuf.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. Increments the pbuf byte access. Switches to next PBuf at end of segment.
Expected Caller Context
interrupt or task level

Table 5-39 IpBase_IncPBuf

5.2.33 IpBase_CopyString2PbufAt

Prototype
`Std_ReturnType` **`IpBase_CopyString2PbufAt`** `(const uint8 *StrPtr, uint16 StrLen, IpBase_PbufType *PbufPtr, uint32 StartPos)`

Parameter	
StrPtr	pointer to source string
StrLen	length of the source string [byte]
PbufPtr	pointer to destination Pbuf struct
StartPos	start position in Pbuf
Return code	
Std_ReturnType	E_OK string could be copied E_NOT_OK string could not be copied

Functional Description
Copy a string to a pbuf at a defined position. Length will be limited to PBuf size and string length.

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.
Expected Caller Context
interrupt or task level

Table 5-40 IpBase_CopyString2PbufAt

5.2.34 IpBase_CopyPbuf2String

Prototype
`Std_ReturnType` **`IpBase_CopyPbuf2String`** `(uint8 *StrPtr, const IpBase_PbufType *PbufPtr, uint16 StrLen, uint32 StartPos)`

Parameter	
StrPtr	pointer to string
PbufPtr	pointer to Pbuf struct
StrLen	length of the string [byte]

StartPos	absolute start position in Pbuf
Return code	
Std_ReturnType	E_OK string was copied E_NOT_OK string was not copied
Functional Description	
Copy PBuf content into string starting at a given position within PBuf.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
interrupt or task level	

Table 5-41 IpBase_CopyPbuf2String

5.2.35 IpBase_FindStringInPbuf

Prototype
`Std_ReturnType` **`IpBase_FindStringInPbuf`** `(const uint8 *StrPtr, const IpBase_PbufType *PbufPtr, uint16 StrLen, uint32 StartPos, uint32 *StrPosPtr)`

Parameter	
StrPtr	pointer to search string
PbufPtr	pointer to Pbuf struct
StrLen	length of the search string [byte]
StartPos	start position for search in Pbuf
StrPosPtr	index in Pbuf where the searched string starts
Return code	
Std_ReturnType	E_OK string was found E_NOT_OK string was not found or API parameters are invalid
Functional Description	
Find a string in a PBuf starting at a given position within PBuf.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
interrupt or task level	

Table 5-42 IpBase_FindStringInPbuf

5.2.36 IpBase_CheckStringInPbuf

Prototype	
`Std_ReturnType` **`IpBase_CheckStringInPbuf`** `(const uint8 *StrPtr, const IpBase_PbufType *PbufPtr, uint16 StrLen, uint32 StartPos)`	
Parameter	
StrPtr	pointer to search string
PbufPtr	pointer to Pbuf struct
StrLen	length of the search string [byte]
StartPos	start position for search in Pbuf
Return code	
Std_ReturnType	E_OK string was found E_NOT_OK string was not found or API parameters are invalid
Functional Description	
Check whether a string is found in a PBuf at the given position	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
interrupt or task level	

Table 5-43 IpBase_CheckStringInPbuf

5.2.37 IpBase_ReadByteInPbuf

Prototype	
`Std_ReturnType` **`IpBase_ReadByteInPbuf`** `(const IpBase_PbufType *PbufPtr, uint32 BytePos, uint8 *SingleBytePtr)`	
Parameter	
PbufPtr	pointer to Pbuf struct
BytePos	absolute byte position in Pbuf
SingleBytePtr	pointer where the byte shall be copied to
Return code	
Std_ReturnType	E_OK byte was copied E_NOT_OK byte was not copied
Functional Description	
Read a byte from PBuf structure.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	

Expected Caller Context
interrupt or task level

Table 5-44 IpBase_ReadByteInPbuf

5.2.38 IpBase_DelSockAddr

Prototype	
`Std_ReturnType` **`IpBase_DelSockAddr`** `(IpBase_SockAddrType *SockPtr, uint16 Family)`	
Parameter	
SockPtr	socket address
Family	supported family
Return code	
Std_ReturnType	E_OK SockAddr could be deleted E_NOT_OK deletion failed
Functional Description	
Delete socket address i.e. reset values (incl. family, port, ip-addr).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via socket address pointer has to be large enough to reset depending on the address family either IPv4 (4 bytes) or IPv6 (16 bytes) address.	
Expected Caller Context	
interrupt or task level	

Table 5-45 IpBase_DelSockAddr

5.2.39 IpBase_CopySockAddr

Prototype	
`Std_ReturnType` **`IpBase_CopySockAddr`** `(IpBase_SockAddrType *TgtSockPtr, const IpBase_SockAddrType *SrcSockPtr)`	
Parameter	
TgtSockPtr	target socket address
SrcSockPtr	source socket address
Return code	
Std_ReturnType	E_OK SockAddr could be copied E_NOT_OK copy failed
Functional Description	
Copy socket address (incl. family, port, ip-addr) from Src to Tgt.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via source and target socket address pointer has to be large enough to store depending on source address family either IPv4 (4 bytes) or IPv6 (16 bytes) address plus 2 bytes for the address family.
Expected Caller Context
interrupt or task level

Table 5-46 IpBase_CopySockAddr

5.2.40 IpBase_CopyIpV6Addr

Prototype
`Std_ReturnType` **`IpBase_CopyIpV6Addr`** `(IpBase_AddrIn6Type *TgtIpAddrPtr, const IpBase_AddrIn6Type *SrcIpAddrPtr)`

Parameter	
TgtIpAddrPtr	target IP address
SrcIpAddrPtr	source IP address

Return code	
Std_ReturnType	E_OK IP addr could be copied E_NOT_OK copy failed

Functional Description
Copy IPv6 socket address (incl. family, port, ip-addr) from Src to Tgt.

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via source and target socket address pointer has to be large enough to store IPv6 (16 bytes).
Expected Caller Context
interrupt or task level

Table 5-47 IpBase_CopyIpV6Addr

5.2.41 IpBase_SockIpAddrIsEqual

Prototype
`boolean` **`IpBase_SockIpAddrIsEqual`** `(const IpBase_SockAddrType *SockAPtr, const IpBase_SockAddrType *SockBPtr)`

Parameter	
SockAPtr	socket address A
SockBPtr	socket address B

Return code	
boolean	TRUE IP address is equal FALSE IP address is not equal
Functional Description	
Check if IP address of sockets is equal.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via source and target socket address pointer has to be large enough to store depending on source address family either IPv4 (4 bytes) or IPv6 (16 bytes) address plus 2 bytes for the address family.	
Expected Caller Context	
interrupt or task level	

Table 5-48 IpBase_SockIpAddrIsEqual

5.2.42 IpBase_SockPortIsEqual

Prototype	
`boolean IpBase_SockPortIsEqual (const IpBase_SockAddrType *SockAPtr, const IpBase_SockAddrType *SockBPtr)`	
Parameter	
SockAPtr	target socket address
SockBPtr	source socket address
Return code	
boolean	TRUE port is equal FALSE port is not equal
Functional Description	
Check if port of sockets is equal.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via source and target socket address pointer has to be large enough to store address family (2 bytes) plus 2 bytes for the address port.	
Expected Caller Context	
interrupt or task level	

Table 5-49 IpBase_SockPortIsEqual

5.2.43 IpBase_CalcTcpIpChecksum

Prototype	
`uint16 IpBase_CalcTcpIpChecksum (const uint8 *DataPtr, uint32 LenByte)`	
Parameter	
DataPtr	pointer to the data
LenByte	data length in bytes
Return code	
uint16	calculated checksum
Functional Description	
This API calculates the checksum over a given data range. The checksum is TcpIp specific. I.e. it expects 16bit data chunks and uses one's complement checksum algorithm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via data pointer has to be large enough to read data length parameter bytes. > Deprecated	
Expected Caller Context	
interrupt or task level	

Table 5-50 IpBase_CalcTcpIpChecksum

5.2.44 IpBase_CalcTcpIpChecksum2

Prototype	
`uint16 IpBase_CalcTcpIpChecksum2 (const uint8 *DataPtr, uint32 LenByte, const uint8 *PseudoHdrPtr, uint32 PseudoHdrLenByte)`	
Parameter	
DataPtr	pointer to the data
LenByte	data length in bytes
PseudoHdrPtr	pointer to the pseudo header
PseudoHdrLenByte	pseudo header length in bytes
Return code	
uint16	calculated checksum
Functional Description	
This API calculates the checksum over two given data ranges. The checksum is TcpIp specific. I.e. it expects 16bit data chunks and uses one's complement checksum algorithm.	

Particularities and Limitations
> Service ID: see table 'Service IDs'
> This function is synchronous.
> This function is reentrant.
> The memory addressed via data pointer has to be large enough to read data length parameter bytes. The memory addressed via data pointer has to be large enough to read data length parameter bytes. The memory addressed via pseudo header pointer has to be large enough to read pseudo header length parameter bytes.
> Deprecated
Expected Caller Context
interrupt or task level

Table 5-51 IpBase_CalcTcpIpChecksum2

5.2.45 IpBase_CalcTcpIpChecksumAdd

Prototype	
`uint16 IpBase_CalcTcpIpChecksumAdd (const uint8 *DataPtr, uint32 LenByte, uint32 Checksum, boolean Stop)`	
Parameter	
DataPtr	pointer to the data
LenByte	data length in bytes
Checksum	checksum from last call, 0 for first call
Stop	true for last call, false for all but last call of this function to finalize checksum
Return code	
uint16	calculated checksum
Functional Description	
This API adds a range to TcpIp checkusm calculation. The checksum is TcpIp specific. I.e. it expects 16bit data chunks and uses one's complement checksum algorithm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'	
> This function is synchronous.	
> This function is reentrant.	
> The memory addressed via data pointer has to be large enough to read data length parameter bytes.	
Expected Caller Context	
interrupt or task level	

Table 5-52 IpBase_CalcTcpIpChecksumAdd

5.2.46 IpBase_TcpIpChecksumAdd

Prototype
`uint32 IpBase_TcpIpChecksumAdd (const uint8 *DataPtr, uint32 LenByte, uint32 Checksum, boolean Stop)`

Parameter	
DataPtr	Pointer to data.
LenByte	Data length in bytes. Must be a multiple of 2 for all intermediate data blocks, can be odd for the last data block.
Checksum	Current 32-bit checksum in host byte order. 0 to start a new checksum calculation.
Stop	Build the one's complement to finalize the checksum.
Return code	
uint32	Calculated 32-bit checksum in host byte order (Stop==FALSE) or 16-bit checksum in network byte order (Stop==TRUE).
Functional Description	
Add a range to TcpIp checksum calculation. The checksum is TcpIp specific. I.e. it expects 16-bit data chunks and uses one's complement checksum algorithm.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via the data pointer has to be large enough to read data length parameter bytes.	
Expected Caller Context	
interrupt or task level	

Table 5-53 IpBase_TcpIpChecksumAdd

5.2.47 IpBase_TcpIpChecksumCopyAdd

Prototype	
`uint32 IpBase_TcpIpChecksumCopyAdd (uint8 * TgtDataPtr, const uint8 *SrcDataPtr, uint32 LenByte, uint32 Checksum, boolean Stop)`	
Parameter	
TgtDataPtr	Pointer to target data.
SrcDataPtr	Pointer to source data.
LenByte	Data length in bytes. Must be a multiple of 2 for all intermediate data blocks, can be odd for the last data block that is copied. The memory addressed via the pointers TgtDataPtr and SrcDataPtr has to be large enough to copy 'LenByte' bytes.
Checksum	Current 16-bit checksum in host byte order. 0 to start a new checksum calculation.
Stop	Build the one's complement to finalize the checksum.
Return code	
uint32	Calculated 16-bit checksum in host byte order (Stop==FALSE) or 16-bit checksum in network byte order (Stop==TRUE).

Functional Description
Copies a source array of any alignment to a target array of any alignment and calculates the additive TcpIp checksum of the data during copy. Processing 32-bit aligned data is the fastest.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via the data pointers has to be large enough to copy 'LenByte' bytes (SMI-1134760).
Expected Caller Context
interrupt or task level

Table 5-54 IpBase_TcpIpChecksumCopyAdd

5.2.48 IpBase_StrCpy

Prototype	
`uint8 IpBase_StrCpy (uint8 *TgtPtr, const uint8 *SrcPtr)`	
Parameter	
TgtPtr	pointer for target string
SrcPtr	pointer to source string
Return code	
uint8	number of copied bytes
Functional Description	
String copy for zero terminated strings.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The source string has to be terminated by '\0', the memory addressed by the target pointer has to be large enough to copy the entire string including the trailing '\0'. > The string length is limited to 0xFFFF to detect missing '\0'. > Deprecated: Please switch to IpBase_StrCpyMaxLen	
Expected Caller Context	
task level	

Table 5-55 IpBase_StrCpy

5.2.49 IpBase_StrCpyMaxLen

Prototype
`uint8 IpBase_StrCpyMaxLen (uint8 *TgtPtr, const uint8 *SrcPtr, uint32 MaxLen)`

Parameter	
TgtPtr	pointer for target string
SrcPtr	pointer to source string
MaxLen	maximum length
Return code	
uint8	number of copied bytes
Functional Description	
String copy (zero terminated strings) with length limitation.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The source string has to be terminated by '\0', the memory addressed by the target pointer has to be as large as given by maximum length parameter.	
Expected Caller Context	
interrupt or task level	

Table 5-56 IpBase_StrCpyMaxLen

5.2.50 IpBase_StrCmp

Prototype	
`uint8 IpBase_StrCmp (const uint8 *Str1Ptr, const uint8 *Str2Ptr)`	
Parameter	
Str1Ptr	pointer to first string
Str2Ptr	pointer to second string
Return code	
uint8	IPBASE_CMP_EQUAL strings are equal IPBASE_CMP_NOT_EQUAL string pattern not found
Functional Description	
Compare 2 strings until end of the shorter string.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The strings have to be terminated by '\0'. String subsets are accepted (i.e. "Hello" == "Hello World"). > The string length is limited to 0xFFFF to detect missing ‘\0’.	
Expected Caller Context	
interrupt or task level	

Table 5-57 IpBase_StrCmp

5.2.51 IpBase_StrCmpLen

Prototype	
`uint8 IpBase_StrCmpLen (const uint8 *Str1Ptr, const uint8 *Str2Ptr, uint16 StrLen)`	
Parameter	
Str1Ptr	pointer to string 1
Str2Ptr	pointer to string 2
StrLen	length of the search string [byte]
Return code	
uint8	IPBASE_CMP_EQUAL strings are equal IPBASE_CMP_NOT_EQUAL strings are not equal, or other error condition occurred
Functional Description	
Compare 2 strings with limited length.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed has to be as large as given by string length parameter.	
Expected Caller Context	
interrupt or task level	

Table 5-58 IpBase_StrCmpLen

5.2.52 IpBase_StrCmpNoCase

Prototype	
`uint8 IpBase_StrCmpNoCase (const uint8 *Str1Ptr, const uint8 *Str2Ptr)`	
Parameter	
Str1Ptr	pointer to first string
Str2Ptr	pointer to second string
Return code	
uint8	IPBASE_CMP_EQUAL strings are equal IPBASE_CMP_NOT_EQUAL string pattern not found
Functional Description	
Compare 2 strings until end of the shorter string ignoring the case (accepting sub strings).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The strings have to be terminated by '\0'. String subsets are accepted (i.e. "Hello"=="Hello World") case is ignored (i.e. "Hello" == "HELLO"). > The string length is limited to 0xFFFF to detect missing '\0'.
Expected Caller Context
interrupt or task level

Table 5-59 IpBase_StrCmpNoCase

5.2.53 IpBase_StrCmpLenNoCase

Prototype
`uint8 IpBase_StrCmpLenNoCase (const uint8 *Str1Ptr, const uint8 *Str2Ptr, uint16 StrLen)`

Parameter	
Str1Ptr	pointer to first string
Str2Ptr	pointer to second string
StrLen	length of the search string [byte]
Return code	
uint8	IPBASE_CMP_EQUAL strings are equal IPBASE_CMP_NOT_EQUAL string pattern not found

Functional Description
Compare 2 strings until end of the shorter string ignoring the case.

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > String subsets are accepted (i.e. "Hello" == "Hello World") case is ignored (i.e. "Hello" == "HELLO").
Expected Caller Context
interrupt or task level

Table 5-60 IpBase_StrCmpLenNoCase

5.2.54 IpBase_StrFindSubStr

Prototype
`uint32 IpBase_StrFindSubStr (const uint8 *StrPtr, const uint8 *SubStrPtr, uint16 StrLen, uint16 SubStrLen)`

Parameter	
StrPtr	pointer to string
SubStrPtr	pointer to sub string

StrLen	length of the string [byte]
SubStrLen	length of the sub string [byte]
Return code	
uint32	PosByte position in string where the sub-string starts IPBASE_STR_LEN_INVALID sub-string not found or error
Functional Description	
Search for the first occurrence of sub-string within a string (e.g. string "hello world", sub-string "world").	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Both strings have to be terminated by '\0', the memory addressed has to be as large as given by string length parameter.	
Expected Caller Context	
interrupt or task level	

Table 5-61 IpBase_StrFindSubStr

5.2.55 IpBase_StrLen

Prototype
`uint32 IpBase_StrLen (const uint8 *StrPtr, uint32 MaxLen)`
Parameter
StrPtr: pointer to string MaxLen: maximum length of the search string [byte]
Return code
uint32: 0..MaxLen-1 length of the string
Functional Description
Check the length of the string.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The string has to be terminated by '\0', the memory addressed has to be as large as given by string length parameter.
Expected Caller Context
interrupt or task level

Table 5-62 IpBase_StrLen

5.2.56 IpBase_ConvInt2String

Prototype	
`Std_ReturnType` **`IpBase_ConvInt2String`** `(uint32 IntVal, uint8 **StrPtr, uint8 *StrLenPtr)`	
Parameter	
IntVal	integer number
StrPtr	pointer to string
StrLenPtr	pointer to length of the string [byte]
Return code	
Std_ReturnType	E_OK integer converted E_NOT_OK integer conversion failed
Functional Description	
Convert an integer number to an ASCII string (dec).	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter.	
Expected Caller Context	
interrupt or task level	

Table 5-63 IpBase_ConvInt2String

5.2.57 IpBase_ConvInt2HexString

Prototype	
`Std_ReturnType` **`IpBase_ConvInt2HexString`** `(uint32 IntVal, uint8 **StrPtr, uint8 *StrLenPtr)`	
Parameter	
IntVal	integer number
StrPtr	pointer to string (hex coded)
StrLenPtr	pointer to length of the string [byte]
Return code	
Std_ReturnType	E_OK integer converted E_NOT_OK integer conversion failed
Functional Description	
Convert an integer number to an ASCII string (hex).	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter.
Expected Caller Context
interrupt or task level

Table 5-64 IpBase_ConvInt2HexString

5.2.58 IpBase_ConvInt2StringBase

Prototype
`Std_ReturnType` **`IpBase_ConvInt2StringBase`** `(uint32 IntVal, uint8 *StrPtr, uint8 StrLen)`

Parameter	
IntVal	integer number
StrPtr	pointer to string
StrLenPtr	pointer to length of the string [byte]

Return code	
Std_ReturnType	E_OK integer converted E_NOT_OK integer conversion failed

Functional Description
Convert an integer number to an ASCII string (dec) without incrementing StrPtr but '\0' at end.

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter.
Expected Caller Context
interrupt or task level

Table 5-65 IpBase_ConvInt2StringBase

5.2.59 IpBase_ConvInt2StringFront

Prototype
`Std_ReturnType` **`IpBase_ConvInt2StringFront`** `(uint32 IntVal, uint8 **StrPtr, uint8 *StrLenPtr)`

Parameter	
IntVal	integer number
StrPtr	pointer to string
StrLenPtr	pointer to length of the string [byte]

Return code	
Std_ReturnType	E_OK integer converted E_NOT_OK integer conversion failed
Functional Description	
Convert an integer number to an ASCII string (dec) without incrementing StrPtr but '\0' at end.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter.	
Expected Caller Context	
interrupt or task level	

Table 5-66 IpBase_ConvInt2StringFront

5.2.60 IpBase_ConvArray2HexStringBase

Prototype	
`Std_ReturnType IpBase_ConvArray2HexStringBase (const uint8 *ArrayPtr, uint16 ArrayLen, uint8 *StrPtr)`	
Parameter	
ArrayPtr	pointer to array
ArrayLen	array length [byte]
StrPtr	pointer to string (has to provide (ArrayLen*2)+1 chars)
Return code	
Std_ReturnType	E_OK array converted E_NOT_OK integer conversion failed
Functional Description	
Convert an array number to an ASCII string (hex), omits leading '00'.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Array [a0a1], ArrayLen 2 -> 'A0A1'. Array [00a1], ArrayLen 2 -> 'A1'. The memory addressed via string pointer has to be large enough to store array length elements. The memory addressed via array pointer has to be large enough to read out array length elements.	
Expected Caller Context	
interrupt or task level	

Table 5-67 IpBase_ConvArray2HexStringBase

5.2.61 IpBase_ConvString2Int

Prototype	
`Std_ReturnType` **`IpBase_ConvString2Int`** `(const uint8 *StrPtr, uint8 StrLen, uint32 *IntValPtr)`	
Parameter	
StrPtr	pointer to string
StrLen	length of the string [byte], length has to be 10 or less (uint32 limit)
IntValPtr	pointer to integer number
Return code	
Std_ReturnType	E_OK string could be converted to integer E_NOT_OK string could not be converted to integer
Functional Description	
Convert an ASCII string (dec values) to an integer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter. The memory addressed via integer number pointer has to point to 4byte size integer (uint32). The string content is not checked to contain valid integer numbers (i.e. 0-9).	
Expected Caller Context	
interrupt or task level	

Table 5-68 IpBase_ConvString2Int

5.2.62 IpBase_ConvString2IntDyn

Prototype	
`Std_ReturnType` **`IpBase_ConvString2IntDyn`** `(const uint8 **StrPtr, uint8 *StrLenPtr, uint32 *IntValPtr)`	
Parameter	
StrPtr	pointer to string
StrLen	length of the string [byte], length has to be 10 or less (uint32 limit)
IntValPtr	pointer to integer number
Return code	
Std_ReturnType	E_OK string could be converted to integer E_NOT_OK string could not be converted to integer
Functional Description	
Convert an ASCII string (dec values) to an integer with dynamic length.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Str '12', StrLen 2 -> 12. Str '12', StrLen 1 -> 1. The memory addressed via string pointer has to be as large as given by string length parameter. The memory addressed via integer number pointer has to point to 4byte size integer (uint32).
Expected Caller Context
interrupt or task level

Table 5-69 IpBase_ConvString2IntDyn

5.2.63 IpBase_ConvStringHex2Int

Prototype	
`Std_ReturnType` **`IpBase_ConvStringHex2Int`** `(const uint8 *StrPtr, uint8 StrLen, uint32 *IntValPtr)`	
Parameter	
StrPtr	pointer to string
StrLen	length of the string [byte] , length has to be 8 or less (uint32 limit)
IntValPtr	pointer to integer number
Return code	
Std_ReturnType	E_OK string could be converted to integer E_NOT_OK string could not be converted to integer
Functional Description	
Convert an ASCII string (hex values) to an integer.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The memory addressed via string pointer has to be as large as given by string length parameter. The memory addressed via integer number pointer has to point to 4byte size integer (uint32). The string content is not checked to contain valid hexadecimal numbers (i.e. 0-9|A-F).	
Expected Caller Context	
interrupt or task level	

Table 5-70 IpBase_ConvStringHex2Int

5.2.64 IpBase_ConvStringHex2IntDyn

Prototype	
`Std_ReturnType` **`IpBase_ConvStringHex2IntDyn`** `(const uint8 **StrPtr, uint8 *StrLenPtr, uint32 *IntValPtr)`	
Parameter	
StrPtr	pointer to string

StrLen	length of the string [byte], length has to be 8 or less (uint32 limit)
IntValPtr	pointer to integer number
Return code	
Std_ReturnType	E_OK string could be converted to integer E_NOT_OK string could not be converted to integer
Functional Description	
Convert an ASCII string (hex values) to an integer with dynamic length.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Str '12', StrLen 2 -> 0x12. Str '12', StrLen 1 -> 0x1. The memory addressed via string pointer has to be as large as given by string length parameter. The memory addressed via integer number pointer has to point to 4byte size integer (uint32).	
Expected Caller Context	
interrupt or task level	

Table 5-71 IpBase_ConvStringHex2IntDyn

5.2.65 IpBase_ConvString2IntBase

Prototype		
`uint32 IpBase_ConvString2IntBase (const uint8 *StrPtr, uint8 StrMaxLen)`		
Parameter		
StrPtr	pointer to string	
StrMaxLen	max length of the string [byte], length has to be 10 or less (uint32 limit)	
Return code		
uint32	0-4294967295	string converted to integer
	IPBASE_ULONG_MAX	string could not be converted to integer
Functional Description		
Convert an ASCII string (dec values) to an integer.		
Particularities and Limitations		
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Str '12', StrMaxLen 2 -> 12. Str '12', StrMaxLen 1 -> 1. The memory addressed via string pointer has to be as large as given by string length parameter.		
Expected Caller Context		
interrupt or task level		

Table 5-72 IpBase_ConvString2IntBase

5.2.66 IpBase_ConvString2SignedIntBase

Prototype		
`sint32 IpBase_ConvString2SignedIntBase (const uint8 *StrPtr, uint8 StrMaxLen)`		
Parameter		
StrPtr	pointer to string	
StrMaxLen	max length of the string [byte], length has to be 10 or less (uint32 limit)	
Return code		
sint32	-2147483646-+2147483646	string converted to integer
	IPBASE_SLONG_MAX	string could not be converted to integer
Functional Description		
Convert an ASCII string (decimal values) to a signed integer.		
Particularities and Limitations		
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Str '12', StrMaxLen 2 -> 12. Str '12', StrMaxLen 1 -> 1. Str 'a' -> IPBASE_SLONG_MAX. The memory addressed via string pointer has to be as large as given by string length parameter.		
Expected Caller Context		
interrupt or task level		

Table 5-73 IpBase_ConvString2SignedIntBase

5.2.67 IpBase_ConvHexString2ArrayBase

Prototype	
`Std_ReturnType IpBase_ConvHexString2ArrayBase (uint8 *ArrayPtr, uint16 ArrayLen, const uint8 * const StrPtr)`	
Parameter	
ArrayPtr	pointer to array
ArrayLen	array length [byte]
StrPtr	pointer to string
Return code	
Std_ReturnType	E_OK array converted E_NOT_OK array conversion failed
Functional Description	
Convert an ASCII hex string to an array number, omits leading '00'. Array is filled up with 0.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > Array [a0a1], ArrayLen 2 -> 'A0A1'. Array [00a1], ArrayLen 2 -> 'A1'. The memory addressed via array pointer has to be as large as given by array length parameter. The memory addressed via string pointer has to be large enough for array length parameter characters.
Expected Caller Context
interrupt or task level

Table 5-74 IpBase_ConvHexString2ArrayBase

5.3 Configurable Interfaces

5.3.1 Notifications

IpBase defines no notifications that can be mapped to callback functions provided by other modules.

6 Configuration

In the IpBase attributes can be configured according to the tool DaVinci Configurator Pro.

6.1 Configuration Variants

The IpBase supports the configuration variants

> VARIANT-PRE-COMPILE

The configuration classes of the IpBase parameters depend on the supported configuration variants. For their definitions please see the IpBase_bswmd.arxml file.

6.2 Configuration with IpBase_Cfg.h

The IpBase configuration is written to the file IpBase_Cfg.h.

6.2.1 Component Configuration

The following attributes can be configured for IpBase, if it is delivered as source code.

Attribute Name	Value Type	Values The default value is written in bold	Description
Enable Development Error Report	Boolean	STD_ON, **STD_OFF**	Enables the Development Error Tracing (DET).

Table 6-1 Configuration parameter descriptions

6.2.2 User Configuration

Not relevant.

7 AUTOSAR Standard Compliance

Currently, the component is not considered in AUTOSAR. However, the component is designed based on AUTOSAR principles.

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
SysService_IpBase	Vector Informatik component name of the IPBase module
DaVinci Configurator Pro	DaVinci Configurator Pro 5 generation tool for MICROSAR Classic components

Table 7-1 Glossary

8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
MICROSAR Classic	Microcontroller Open System Architecture (the Vector AUTOSAR Classic solution)

Table 7-2 Abbreviations

9 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector-informatik.com