MICROSAR Crypto Abstraction Library

Technical Reference

Version 2.2

Authors	Wladimir Gerber, Markus Schneider
Status	Released

Document Information

History

Author	Date	Version	Remarks
Wladimir Gerber	2012-10-01	1.0	Initial version
Markus Schneider	2013-04-03	2.0	Updated for AUTOSAR 4; Adapting chapter 3.1.3, 5 and chapter 7.1
Markus Schneider	2013-10-14	2.1	Update of chapter ‘5 Configuration’
Markus Schneider	2014-10-15	2.2	Added SymEncrypt, KeyDerive and KeyExchange services; Removed chapter “Component History”

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_CryptoAbstractionLibrary.pdf	V1.2.0 R4.0 Rev 3
[2]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0 R4.0 Rev 3

Scope of the Document

This technical reference describes the general use of the Microsar Crypto Abstraction Library (CAL) software.

Contents

Illustrations

Tables

1 Introduction

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

Supported AUTOSAR Release:	4.0 Release 3	
Supported Configuration Variants:	Pre-compile time	
Vendor ID:	CAL_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CAL_MODULE_ID	206 decimal (according to ref. see [2])

This module provides a standardized interface to some primitives for cryptographic functionality. The CAL module itself does not implement the crypto functions, but utilizes the CPL module. Thus, the actual implementation of symmetrical or asymmetrical encryption or decryption depends on the functionality and capability of the underlying CPL module. The CAL just provides a common interface to these routines according to AUTOSAR standards.

The Cryptographic Primitive Library (CPL) is a separate package and not part of the CAL module.

The CAL offers C functions that can be called from source code, i.e. from BSW modules, from SWC or from Complex Device Drivers.

As the CAL is a library, it is not related to a special layer of the AUTSOAR Layered Software Architecture. The services of the CAL are always executed in the context of the calling function.

The module implementation related to this document provides a subset of the functionality specified in [1].

The current version covers the following functionality:

- **Symmetrical Decryption/Encryption Interface:** Symmetrical decryption and encryption of data with arbitrary length.
- **Random Interface:** A self-shrinking generator is provided for generating pseudo random numbers.
- **Signature Verify Interface:** Verification of a digital signature.
- **Hash Interface:** Computation of a hash value.
- **Symmetrical Key Extract Interface:** Extracting a key from a given char array.
- **Symmetrical Block Interface:** Symmetrical block decryption and encryption.

> **MAC Generate and Verify Interface:** Generation and verification of a Message Authentication Code (MAC).

> **Key Derivation Interface:** Derive one or more secret keys from a secret value.

> **Key Exchange Interface:** Key exchange service.

1.1 Architecture Overview

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

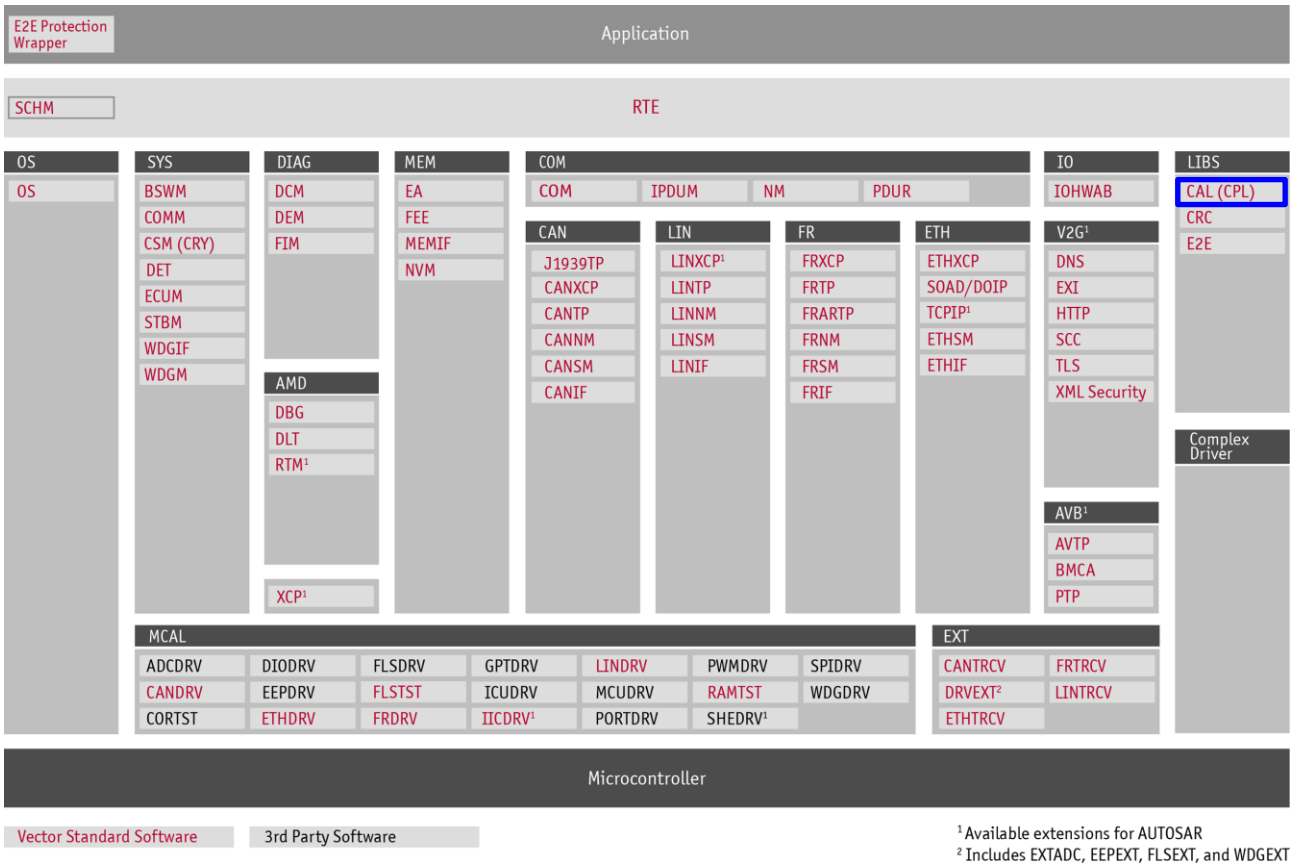

Figure 1-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the CAL. These interfaces are described in chapter 4.

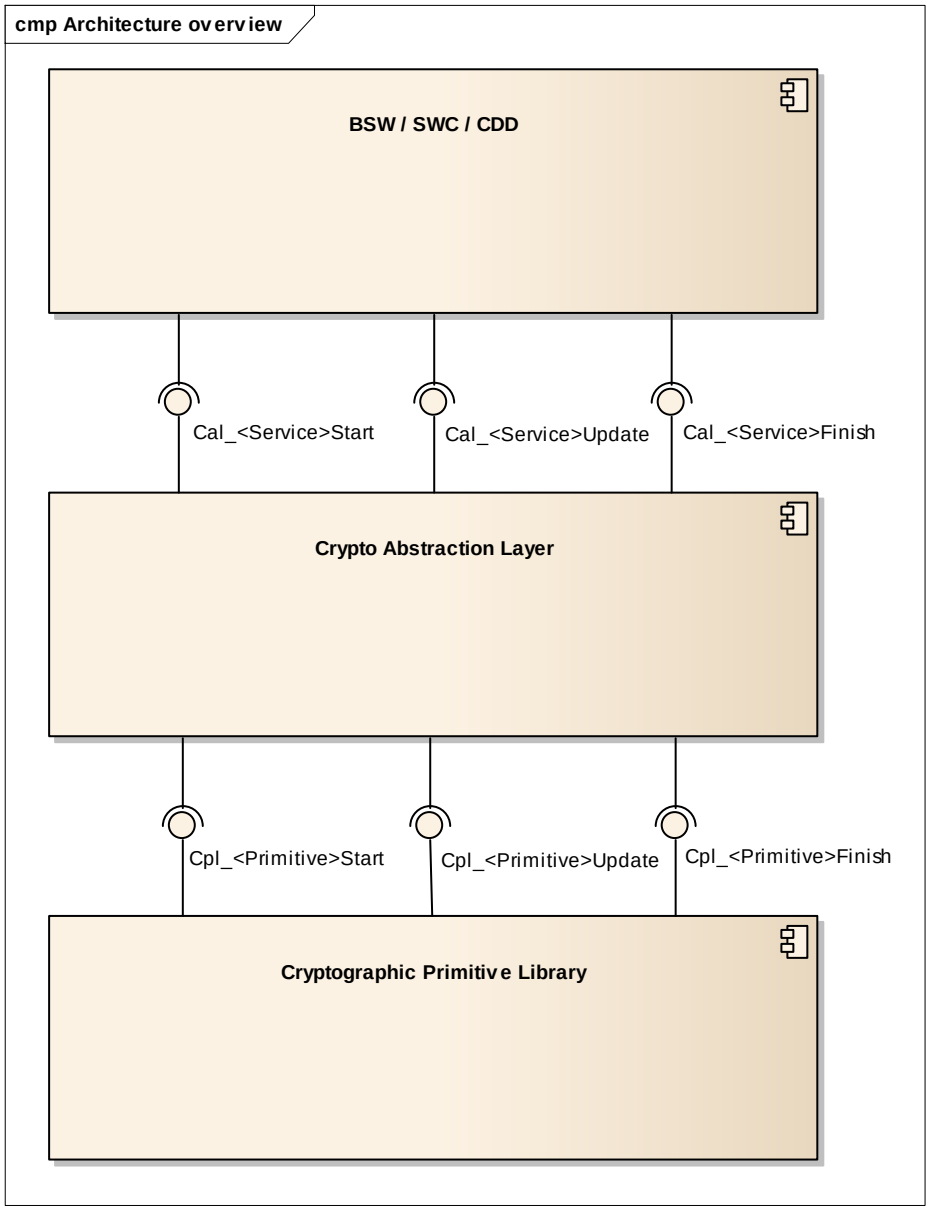

Figure 1-2 Interfaces to adjacent modules of the CAL

2 Functional Description

2.1 Features

The features listed in the following tables cover a subset of the functionality specified for the CAL.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

- Table 2-1 Supported AUTOSAR standard conform features
- Table 2-2 Not supported AUTOSAR standard conform features

For further information of not supported features see also chapter 5.

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Symmetrical Interface (Encryption and Decryption)
Hash Interface
Symmetrical Block Interface (Encryption and Decryption)
Signature Interface (Verification)
Symmetrical Key Extract Interface
MAC Interface (Generate and Verify)
Random Interface
Key Derivation Interface
Key Exchange Interface

Table 2-1 Supported AUTOSAR standard conform features

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Asymmetrical Interface
Checksum Interface
Symmetrical Key Wrapping Interface
Asymmetrical Key Extract Interface
Asymmetrical Key Wrapping Interface

Table 2-2 Not supported AUTOSAR standard conform features

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
None

Table 2-3 Features provided beyond the AUTOSAR standard

2.2 Initialization

There is no specific module initialization necessary. Each primitive instance is initialized separately in the specific Cal_<service>Start function before the functionality itself can be used.

2.3 States

The CAL module provides two types of functionalities. One type is a simple user interface, where the functionality can be accessed directly. The other type uses a streaming approach, where an arbitrary amount of data can be applied to the service primitive. In the latter case, the functionality will be accessed through 3 functions: the **Cal_<service>Start()**, the **Cal_<service>Update()** and the **Cal_<service>Finish** function. Each service contains a state machine, saved in the context buffer, which allows the call of each function only in its appropriate state. The internal state diagram of each of the function can be depicted from the following figure:

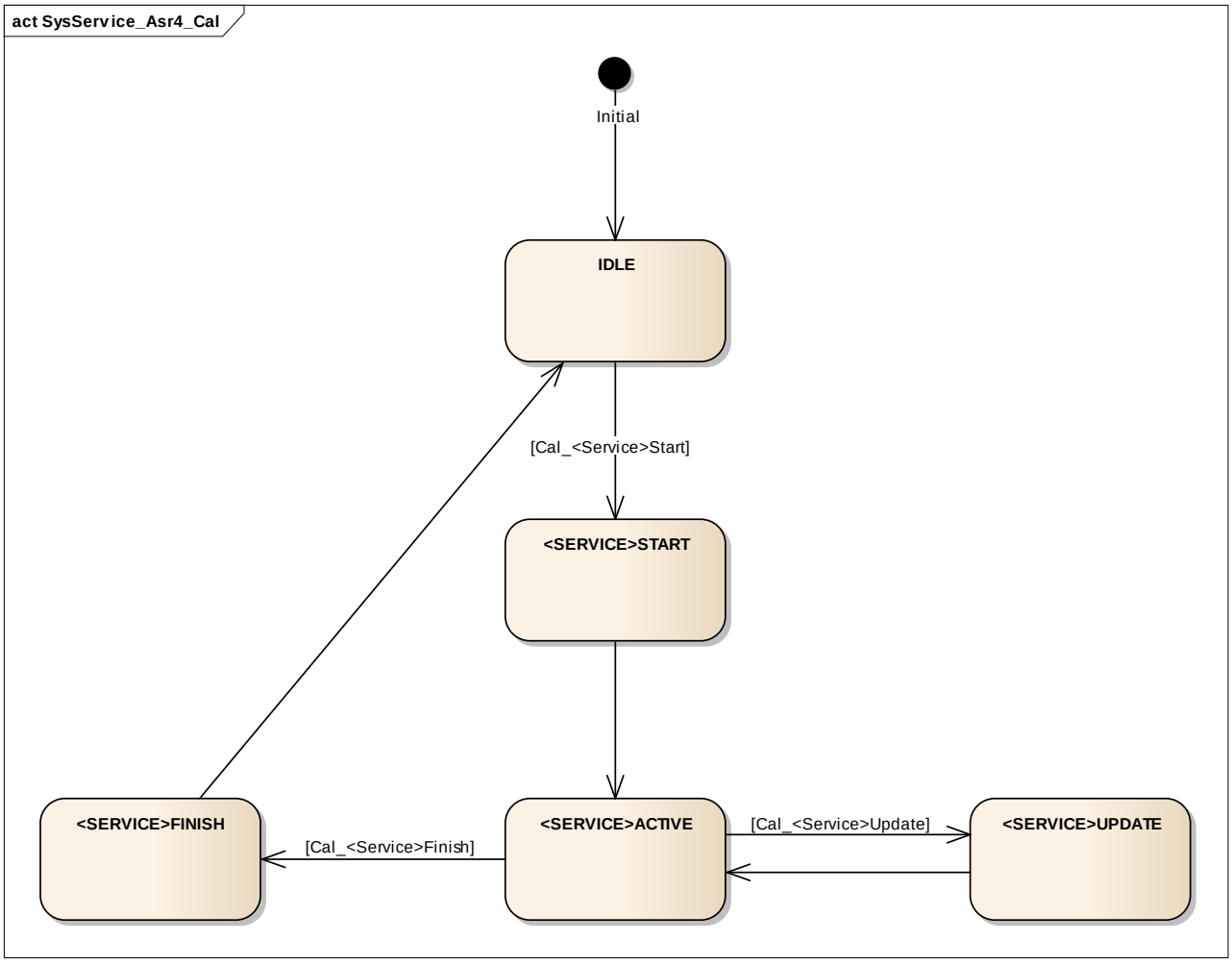

Figure 2-1 Sequence Diagram

The CAL module does not provide any main function.

2.3.1 Streaming Approach

The implementation of those CAL services which expect arbitrary amounts of user data are based on the streaming approach with start, update and finish functions, see Figure 2-1.

2.4 Error Handling

The module does not provide any error detections according to AUTOSAR.

2.4.1 Development Error Reporting

The CAL is not performing any error checking at development level and also does not use the DET. There are no configuration settings necessary for error reporting.

2.4.2 Production Code Error Reporting

The module does not generate production code error reporting.

3 Integration

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

3.1 Scope of Delivery

The delivery of the CAL contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
Cal.h	■		This is the header file of the Hash Interface. It contains all function declarations.
Cal.c	■		This source file contains Cal_GetVersionInfo
Cal_Hash.c	■		This source file contains the definition of the Hash Inteface.
Cal_KeyDerive.c	■		This source file contains the definition of the Key Derive Interface.
Cal_KeyExchange	■		This source file contains the definition of the Key Exchange Interface.
Cal_MacGenerate.c	■		This source file contains the definition of the MAC Generate Inteface.
Cal_MacVerify.c	■		This source file contains the definition of the MAC Verification Inteface.
Cal_Random.c	■		This source file contains the definition of the Random Inteface.
Cal_SignatureVerify.c	■		This source file contains the definition of the Signature Verify Inteface.
Cal_SymBlockDecrypt.c	■		This source file contains the definition of the Symmetrical Block Decrypt Inteface.
Cal_SymBlockEncrypt.c	■		This source file contains the definition of the Symmetrical Block Encrypt Inteface.
Cal_SymDecrypt.c	■		This source file contains the definition of the Symmetrical Decrypt Inteface.
Cal_SymEncrypt.c	■		This source file contains the definition of the Symmetrical Encrypt Interface.
Cal_SymKeyExtract.c	■		This source file contains the definition of the Symmetrical Key Extract Inteface.
Cal_Types.h	■		This header file contains the declaration of all CAL specific Types.

File Name	Source Code Delivery	Object Code Delivery	Description
Cpl.lib[1]		■	This is the library of the Cryptographic primitive function.

Table 3-1 Static files

The files for the CAL module come along with the CPL library functions. Thus, integrating the CAL requires also add the CPL.lib to your linker command file. The CAL uses the services of the CPL when calling the CAL. The CPL functions are not described here explicitly.

3.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator. For more information about the configuration see chapter 5.2 ‘Configuration with DaVinci Configurator’.

File Name	Description
Cal_Cfg.h	This is header file contains the service type configurations and name handles for the configuration IDs
Cal_Cfg.c	This source file contains the service configurations.

Table 3-2 Dynamic files

[1] The CPL is an additional module.

3.1.3 Include Structure

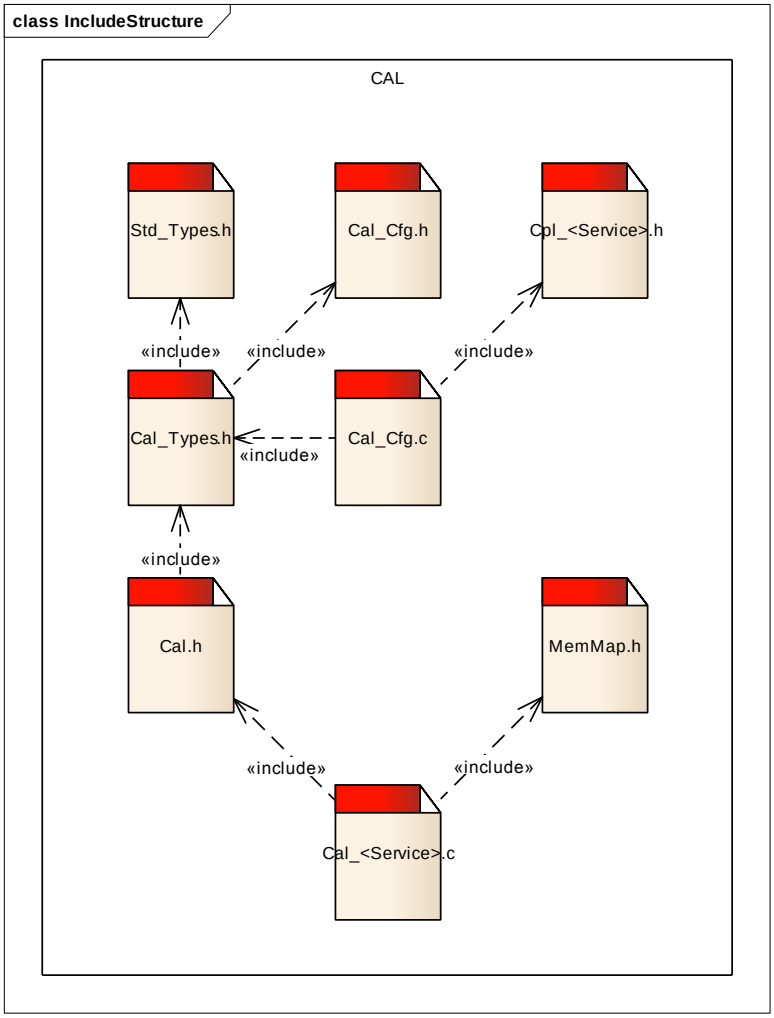

Figure 3-1 Include structure of the MSR CAL

Due to the adjustable context buffer sizes, the include structure of the CAL differs from the AUTOSAR Specification. The ‘Cal_Types.h’ includes the ‘Cal_Cfg.h’ than the other way round. Furthermore the ‘Cal_Cfg.c’ has to include the specific ‘Cpl_<Service>.h’ headers to solve the linked symbols.

3.2 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 CAL and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	CAL_CODE	CAL_CONST
CAL_START_SEC_CONST_UNSPECIFIED CAL_STOP_SEC_CONST_UNSPECIFIED		■
CAL_START_SEC_CODE CAL_STOP_SEC_CODE	■	

Table 3-3 Compiler abstraction and memory mapping

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
Cal_ReturnType	Enum	Defines the different return values from the CAL functions	CAL_E_OK
			CAL_E_NOT_OK
			CAL_E_SMALL_BUFFER
			CAL_E_ENTROPY_EXHAUSTION
Cal_ConfigIdType	Enum	The configuration ID of the cryptographic primitive.	Every configuration gets its own configuration id.
Cal_ActionType	Enum	Internal state of the stream functions	CAL_ACT_IDLE
			CAL_ACT_ACTIVE

Table 4-1 Type definitions

Cal_<Service>ConfigType

Struct Element Name	C-Type	Description	Value Range
ConfigId	Cal_ConfigIdType	ID of this configuration	Range of enum Cal_ConfigIdType
PrimitiveStartFct	Function pointer	Pointer to underlying Cpl_<primitive>Start	Depends on the AUTOSAR compiler abstraction definition for function pointers.
PrimitiveUpdateFct	Function pointer	Pointer to underlying Cpl_<primitive>Update	Depends on the AUTOSAR compiler abstraction definition for function pointers.
PrimitiveFinishFct	Function pointer	Pointer to underlying Cpl_<primitive>Finish	Depends on the AUTOSAR compiler abstraction definition for function pointers.
PrimitiveConfigPtr	Const pointer	ID of the underlying primitive configuration.	Depends on the AUTOSAR compiler abstraction definition for pointers to constant data.

Table 4-2 Cal_<Service>ConfigType

Cal_<Primitive>ConfigType

Struct Element Name	C-Type	Description	Value Range
ConfigId	Cal_Co	ID of this configuration	Range of enum Cal_ConfigIdType

Struct Element Name	C-Type	Description	Value Range
PrimitiveFct	Function pointer	Pointer to underlying Cpl_<primitive>	Depends on the AUTOSAR compiler abstraction definition for pointers to constant data.
PrimitiveConfigPtr	Const pointer	ID of the underlying primitive configuration	Depends on the AUTOSAR compiler abstraction definition for pointers to constant data.

Table 4-3 Cal_<Primitive>ConfigType

Cal_<Service>CtxBufType[]

As the CAL is a library, it is not allowed to store any internal states in the library. When calling a service of the CAL, the application provides a pointer to a buffer, in which the CAL and the underlying CPL can store all context and status information that is necessary to process the service, see [1], [CAL0730].

4.2 Services provided by CAL

4.2.1 Cal_SymDecryptStart

Prototype	
`Cal_ReturnType` **`Cal_SymDecryptStart`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr, const uint8 *InitVectorPtr, uint32 InitVectorLength)`	
Parameter	
cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical decryption computation.
keyPtr	Holds a pointer to the key which has to be used during the symmetrical decryption computation.
InitVectorPtr	Holds a pointer to the initialization vector which has to be used during the symmetrical decryption computation.
InitVectorLength	Holds the length of the initialization vector which has to be used during the symmetrical decryption computation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the symmetrical decrypt service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-4 Cal_SymDecryptStart

4.2.2 Cal_SymDecryptUpdate

Prototype	
`Cal_ReturnType` **`Cal_SymDecryptUpdate`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, const uint8 *cipherTextPtr, uint32 cipherTextLength, uint8 *plainTextPtr, uint32 *plainTextLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical decryption computation.
cipherTextPtr	Holds a pointer to the constant cipher text that shall be decrypted.
cipherTextLength	Contains the length of the cipher text in bytes.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
plainTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER - The provided buffer is too small to store the result.
Functional Description	
This function shall be used to feed the symmetrical decryption service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-5 Cal_SymDecryptUpdate

4.2.3 Cal_SymDecryptFinish

Prototype	
`Cal_ReturnType` **`Cal_SymDecryptFinish`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, uint8 *plainTextPtr, uint32 *plainTextLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical decryption computation.

contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
plainTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
Return code	
Cal_ReturnType	CAL_E_OK - success, decryption data has been accepted. CAL_E_NOT_OK - Wrong parameter or calling sequence CAL_E_SMALL_BUFFER - Input buffer size too small (parameter in *plainTextLengthPtr)
Functional Description	
This function shall be used to finish the symmetrical decryption service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-6 Cal_SymDecryptFinish

4.2.4 Cal_SymEncryptStart

Prototype	
`Cal_ReturnType` **`Cal_SymEncryptStart`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr, const uint8 *InitVectorPtr, uint32 InitVectorLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical encryption computation.
keyPtr	Holds a pointer to the key which has to be used during the symmetrical encryption computation.
InitVectorPtr	Holds a pointer to the initialization vector which has to be used during the symmetrical encryption computation.
InitVectorLength	Holds the length of the initialization vector which has to be used during the symmetrical encryption computation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the symmetrical encrypt service of the CAL module.	

Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-7 Cal_SymEncryptStart

4.2.5 Cal_SymEncryptUpdate

Prototype
`Cal_ReturnType` **`Cal_SymEncryptUpdate`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, const uint8 *plainTextPtr, uint32 plainTextLength, uint8 *cipherTextPtr, uint32 *cipherTextLengthPtr)`

Parameter	
cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical encryption computation.
plainTextPtr	Holds a pointer to the plain text that shall be encrypted.
plainTextLength	Contains the length of the plain text in bytes.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
cipherTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER - The provided buffer is too small to store the result.
Functional Description	
This function shall be used to feed the symmetrical encryption service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-8 Cal_SymEncryptUpdate

4.2.6 Cal_SymEncryptFinish

Prototype	
`Cal_ReturnType` **`Cal_SymEncryptFinish`** `(Cal_ConfigIdType cfgId, Cal_SymDecryptCtxBufType contextBuffer, uint8 *cipherTextPtr, uint32 *cipherTextLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the constant CAL module configuration which has to be used during the symmetrical encryption computation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
cipherTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return code	
Cal_ReturnType	CAL_E_OK - success, decryption data has been accepted. CAL_E_NOT_OK - Wrong parameter or calling sequence CAL_E_SMALL_BUFFER - Input buffer size too small (parameter in *plainTextLengthPtr)
Functional Description	
This function shall be used to finish the symmetrical encryption service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-9 Cal_SymEncryptFinish

4.2.7 Cal_KeyDeriveStart

Prototype	
`Cal_ReturnType` **`Cal_KeyDeriveStart`** `(Cal_ConfigIdType cfgId, Cal_KeyDeriveCtxBufType contextBuffer, uint32 keyLength, uint32 iterations)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
keyLength	Holds the length of the key to be derived by the underlying key derivation primitive.
iterations	Holds the number of iterations to be performed by the underlying key derivation primitive.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the key derivation service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-10 Cal_KeyDeriveStart

4.2.8 Cal_KeyDeriveUpdate

Prototype	
`Cal_ReturnType` **`Cal_SymEncryptUpdate`** `(Cal_ConfigIdType cfgId, Cal_KeyDeriveCtxBufType contextBuffer, const uint8 *passwordPtr, uint32 passwordLength, const uint8 *saltPtr, uint32 saltLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
passwordPtr	Holds a pointer to the password, i.e. the original key, from which to derive a new key.
passwordLength	Holds the length of the password in bytes.
saltPtr	Holds a pointer to the cryptographic salt, i.e. a random number, for the underlying primitive.
saltLength	Holds the length of the salt in bytes.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the key derivation service with input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-11 Cal_KeyDeriveUpdate

4.2.9 Cal_KeyDeriveFinish

Prototype	
`Cal_ReturnType` **`Cal_KeyDeriveFinish`** `(Cal_ConfigIdType cfgId, Cal_KeyDeriveCtxBufType contextBuffer, uint8 *keyPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr	Holds a pointer to the memory location which will hold the result of the key derivation.
Return code	
Cal_ReturnType	CAL_E_OK - success, decryption data has been accepted. CAL_E_NOT_OK - Wrong parameter or calling sequence
Functional Description	
This function shall be used to finish the key derive service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-12 Cal_KeyDeriveFinish

4.2.10 Cal_KeyExchangeCalcSecretStart

Prototype	
`Cal_ReturnType` **`Cal_KeyExchangeCalcSecretStart`** `(Cal_ConfigIdType cfgId, Cal_KeyExchangeCalcSecretCtxBufType contextBuffer, const Cal_KeyExchangeBaseType basePtr, const Cal_KeyExchangePrivateType privateValuePtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
basePtr	Holds a pointer to the base information known to both users of the key exchange protocol.
privateValuePtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by publicValuePtr. On returning from this function the actual length of the calculated public value shall be stored.

Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to start the public value calculation service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-13 Cal_KeyExchangeCalcSecretStart

4.2.11 Cal_KeyExchangeCalcSecretUpdate

Prototype	
`Cal_ReturnType` **`Cal_KeyExchangeCalcSecretUpdate`** `(Cal_ConfigIdType cfgId, Cal_KeyExchangeCalcSecretCtxBufType contextBuffer, const uint8 *partnerPublicValuePtr, uint32 partnerPublicValueLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
partnerPublicValuePtr	Holds a pointer to the data representing the public value of the key exchange partner.
partnerPublicValueLength	Contains the length of the partner value in bytes.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the key exchange service with the public value coming from the partner of the key exchange protocol.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-14 Cal_KeyExchangeCalcSecretUpdate

4.2.12 Cal_KeyExchangeCalcSecretFinish

Prototype	
`Cal_ReturnType` **`Cal_KeyExchangeCalcSecretFinish`** `(Cal_ConfigIdType cfgId, Cal_KeyExchangeCalcSecretCtxBufType contextBuffer, uint8 * sharedSecretPtr, uint32 * sharedSecretLengthPtr, boolean TruncationIsAllowed)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key derivation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
sharedSecretPtr	Holds a pointer to the memory location which will hold the result of the key exchange.If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
sharedSecretLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by sharedSecretPtr. On returning from this function the actual length of the computed value shall be stored.
TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not.
Return code	
Cal_ReturnType	CAL_E_OK - success, decryption data has been accepted. CAL_E_NOT_OK - Wrong parameter or calling sequence
Functional Description	
This function shall be used to finish the key exchange service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-15 Cal_KeyExchangeCalcSecretFinish

4.2.13 Cal_KeyExchangeCalcPubVal

Prototype	
`Cal_ReturnType` **`Cal_KeyExchangeCalcPubVal`** `(Cal_ConfigIdType cfgId, const * basePtr, const * privateValuePtr, uint8 * publicValuePtr, uint32 * publicValueLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the key exchange.
basePtr	Holds a pointer to the base information known to both users of the key exchange protocol.
privateValuePtr	Holds a pointer to the private information known only to the current user of the key exchange protocol.

publicValuePtr	Holds a pointer to the memory location which will hold the public value of the key exchange protocol.
publicValueLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by publicValuePtr. On returning from this function the actual length of the calculated public value shall be stored.
Return code	
Cal_ReturnType	CAL_E_OK - success, decryption data has been accepted. CAL_E_NOT_OK - Wrong parameter or calling sequence
Functional Description	
This function shall be used to start the public value calculation service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-16 Cal_KeyExchangeCalcPubVal

4.2.14 Cal_RandomSeedStart

Prototype
`Cal_ReturnType` **`Cal_RandomSeedStart`** `(Cal_ConfigIdType cfgId, Cal_RandomCtxBufType contextBuffer)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the seeding of the random number generator.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to initialize the random seed service of the CAL module.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-17 Cal_RandomSeedStart

4.2.15 Cal_RandomSeedUpdate

Prototype	
`Cal_ReturnType Cal_RandomSeedUpdate (Cal_ConfigIdType cfgId, Cal_RandomCtxBufType contextBuffer, const uint8 *seedPtr, uint32 seedLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the seeding of the random number generator.
seedPtr	Holds a pointer to the seed for the random number generator.
seedLength	Contains the length of the seed in bytes.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed a seed to the random number generator.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-18 Cal_RandomSeedUpdate

4.2.16 Cal_RandomSeedFinish

Prototype	
`Cal_ReturnType Cal_RandomSeedFinish (Cal_ConfigIdType cfgId, Cal_RandomCtxBufType contextBuffer)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during random number generator.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to finish the random seed service.	
Particularities and Limitations	
none	

Call Context
Context of the calling function

Table 4-19 Cal_RandomSeedFinish

4.2.17 Cal_RandomGenerate

Prototype	
`Cal_ReturnType` **`Cal_RandomGenerate`** `(Cal_ConfigIdType cfgId, Cal_RandomCtxBufType contextBuffer, uint8 *resultPtr, uint32 resultLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during random number generator.
resultLength	Holds the amount of random bytes which should be generated.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored. If a seed is needed, this must be the same context buffer that has been used for the call of the RandomSeed interfaces.
resultPtr	Holds a pointer to the memory location which will hold the result of the random number generation. The memory location must have at least the size "resultLength".
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_ENTROPY_EXHAUSTION - Request failed, entropy of random number generator is exhausted.
Functional Description	
This function shall be used to start the random number generation service of the CAL module.	
Particularities and Limitations	
Context of the calling function	
Call Context	
none	

Table 4-20 Cal_RandomGenerate

4.2.18 Cal_SignatureVerifyStart

Prototype
`Cal_ReturnType` **`Cal_SignatureVerifyStart`** `(Cal_ConfigIdType cfgId, Cal_SignatureVerifyCtxBufType contextBuffer, const Cal_AsymPublicKeyType *keyPtr)`

Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
keyPtr	Holds a pointer to the key necessary for the signature verification.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the signature verify service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-21 Cal_SignatureVerifyStart

4.2.19 Cal_SignatureVerifyUpdate

Prototype
`Cal_ReturnType` **`Cal_SignatureVerifyUpdate`** `(Cal_ConfigIdType cfgId, Cal_SignatureVerifyCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
dataPtr — Holds a pointer to the signature which shall be verified.
dataLength — Contains the length of the signature to verify in bytes.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to feed the signature verification service with the input data.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-22 Cal_SignatureVerifyUpdate

4.2.20 Cal_SignatureVerifyFinish

Prototype	
`Cal_ReturnType` **`Cal_SignatureVerifyFinish`** `(Cal_ConfigIdType cfgId, Cal_SignatureVerifyCtxBufType contextBuffer, const uint8 *signaturePtr, uint32 signatureLength, Cal_VerifyResultType *resultPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the signature computation/verification.
signaturePtr	Holds a pointer to the memory location which holds the signature to be verified.
signatureLength	Holds the length of the Signature to be verified.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
resultPtr	Holds a pointer to the memory location which will hold the result of the signature verification.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to finish the signature verification service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-23 Cal_SignatureVerifyFinish

4.2.21 Cal_HashStart

Prototype	
`Cal_ReturnType` **`Cal_HashStart`** `(Cal_ConfigIdType cfgId, Cal_HashCtxBufType contextBuffer)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.
contextBuffer	Buffer pointer to the context data.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed

Functional Description
This function shall be used to initialize the hash service of the CAL module.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-24 Cal_HashStart

4.2.22 Cal_HashUpdate

Prototype	
`Cal_ReturnType Cal_HashUpdate (Cal_ConfigIdType cfgId, Cal_HashCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.
dataPtr	Holds a pointer to the data to be hashed.
dataLength	Contains the number of bytes to be hashed.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the hash service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-25 Cal_HashUpdate

4.2.23 Cal_HashFinish

Prototype
`Cal_ReturnType Cal_HashFinish (Cal_ConfigIdType cfgId, Cal_HashCtxBufType contextBuffer, uint8 *resultPtr, uint32 *resultLengthPtr, boolean truncationIsAllowed)`

Parameter	
cfgId	Holds the identifier of the CAL module configuration that has to be used during the hash value computation.
resultLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. On returning from this function the actual length of the computed value shall be stored.
resultPtr	Holds a pointer to the memory location which will hold the result of the hash value computation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER - The provided buffer is too small to store the result, and truncation was not allowed.
Functional Description	
This function shall be used to finish the hash service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-26 Cal_HashFinish

4.2.24 Cal_SymKeyExtractStart

Prototype
`Cal_ReturnType` **`Cal_SymKeyExtractStart`** `(Cal_ConfigIdType cfgId, Cal_SymKeyExtractCtxBufType contextBuffer)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the key extraction.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to initialize the symmetrical key extraction service of the CAL module.

Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-27 Cal_SymKeyExtractStart

4.2.25 Cal_SymKeyExtractUpdate

Prototype	
`Cal_ReturnType Cal_SymKeyExtractUpdate (Cal_ConfigIdType cfgId, Cal_SymKeyExtractCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
dataPtr	Holds a pointer to the data which contains the key in a format which cannot be used directly by the CAL. From this data the key will be extracted in a CAL-conforming format.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the symmetrical key extraction service with input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-28 Cal_SymKeyExtractUpdate

4.2.26 Cal_SymKeyExtractFinish

Prototype	
`Cal_ReturnType Cal_SymKeyExtractFinish (Cal_ConfigIdType cfgId, Cal_SymKeyExtractCtxBufType contextBuffer, Cal_SymKeyType *keyPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the key extraction.

contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
keyPtr	Holds a pointer to a structure where the result (i.e. the symmetrical key) is stored in.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to finish the symmetrical key extraction service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-29 Cal_SymKeyExtractFinish

4.2.27 Cal_SymBlockEncryptStart

Prototype
`Cal_ReturnType` **`Cal_SymBlockEncryptStart`** `(Cal_ConfigIdType cfgId, Cal_SymBlockEncryptCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
keyPtr — Holds a pointer to the key which has to be used during the symmetrical block encryption computation.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to initialize the symmetrical block encrypt service of the CAL module.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-30 Cal_SymBlockEncryptStart

4.2.28 Cal_SymBlockEncryptUpdate

Prototype	
`Cal_ReturnType` **`Cal_SymBlockEncryptUpdate`** `(Cal_ConfigIdType cfgId, Cal_SymBlockEncryptCtxBufType contextBuffer, const uint8 *plainTextPtr, uint32 plainTextLength, uint8 *cipherTextPtr, uint32 *cipherTextLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
plainTextPtr	Holds a pointer to the plain text that shall be encrypted.
plainTextLength	Contains the length of the plain text in bytes.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
cipherTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the symmetrical block encryption service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-31 Cal_SymBlockEncryptUpdate

4.2.29 Cal_SymBlockEncryptFinish

Prototype	
`Cal_ReturnType` **`Cal_SymBlockEncryptFinish`** `(Cal_ConfigIdType cfgId, Cal_SymBlockEncryptCtxBufType contextBuffer)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block encryption computation.
plainTextPtr	Holds a pointer to the plain text that shall be encrypted.
plainTextLength	Contains the length of the plain text in bytes.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

cipherTextLengthPtr	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by cipherTextPtr. On returning from this function the amount of data that has been encrypted shall be stored.
cipherTextPtr	Holds a pointer to the memory location which will hold the encrypted text.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER The provided buffer is too small to store the result.
Functional Description	
This function shall be used to finish the symmetrical block encryption service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-32 Cal_SymBlockEncryptFinish

4.2.30 Cal_SymBlockDecryptStart

Prototype
`Cal_ReturnType` **`Cal_SymBlockDecryptStart`** `(Cal_ConfigIdType cfgId, Cal_SymBlockDecryptCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the symmetrical block decryption computation.
keyPtr — Holds a pointer to the key which has to be used during the symmetrical block decryption computation.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to initialize the symmetrical block decrypt service of the CAL module.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-33 Cal_SymBlockDecryptStart

4.2.31 Cal_SymBlockDecryptUpdate

Prototype	
`Cal_ReturnType Cal_SymBlockDecryptUpdate (Cal_ConfigIdType cfgId, Cal_SymBlockDecryptCtxBufType contextBuffer, const uint8 *cipherTextPtr, uint32 cipherTextLength, uint8 *plainTextPtr, uint32 *plainTextLengthPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block decryption computation.
plainTextPtr	Holds a pointer to the memory location which will hold the decrypted text.
cipherTextLength	Contains the length of the cipher text in bytes.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
plainTextLength	Holds a pointer to a memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by plainTextPtr. On returning from this function the amount of data that has been decrypted shall be stored.
cipherTextPtr	Holds a pointer to the constant cipher text that shall be decrypted.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the symmetrical block decryption service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-34 Cal_SymBlockDecryptUpdate

4.2.32 Cal_SymBlockDecryptFinish

Prototype	
`Cal_ReturnType Cal_SymBlockDecryptFinish (Cal_ConfigIdType cfgId, Cal_SymBlockDecryptCtxBufType contextBuffer)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the symmetrical block decryption computation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.

Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER The provided buffer is too small to store the result.
Functional Description	
This function shall be used to finish the symmetrical block decryption service.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-35 Cal_SymBlockDecryptFinish

4.2.33 Cal_MacGenerateStart

Prototype
`Cal_ReturnType` **`Cal_MacGenerateStart`** `(Cal_ConfigIdType cfgId, Cal_MacGenerateCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`

Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
keyPtr	Holds a pointer to the key necessary for the MAC generation.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the MAC generate service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-36 Cal_MacGenerateStart

4.2.34 Cal_MacGenerateUpdate

Prototype
`Cal_ReturnType` **`Cal_MacGenerateUpdate`** `(Cal_ConfigIdType cfgId, Cal_MacGenerateCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`

Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
dataPtr	Holds a pointer to the data for which a MAC shall be computed.
dataLength	Contains the number of bytes for which the MAC shall be computed.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the MAC generate service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-37 Cal_MacGenerateUpdate

4.2.35 Cal_MacGenerateFinish

Prototype
`Cal_ReturnType` **`Cal_MacGenerateFinish`** `(Cal_ConfigIdType cfgId, Cal_MacGenerateCtxBufType contextBuffer, uint8 *resultPtr, uint32 *resultLengthPtr, boolean TruncationIsAllowed)`

Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC computation.
TruncationIsAllowed	This parameter states whether a truncation of the result is allowed or not. TRUE: Truncation is allowed. FALSE: Truncation is not allowed.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
resultLengthPtr	Holds a pointer to the memory location in which the length information is stored. On calling this function this parameter shall contain the size of the buffer provided by resultPtr. On returning from this function the actual length of the computed MAC shall be stored.
resultPtr	Holds a pointer to the memory location which will hold the result of the MAC generation. If the result does not fit into the given buffer, and truncation is allowed, the result shall be truncated.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed CAL_E_SMALL_BUFFER - The provided buffer is too small to store the result, and truncation was not allowed.

Functional Description
This function shall be used to finish the MAC generation service.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-38 Cal_MacGenerateFinish

4.2.36 Cal_MacVerifyStart

Prototype	
`Cal_ReturnType` **`Cal_MacVerifyStart`** `(Cal_ConfigIdType cfgId, Cal_MacVerifyCtxBufType contextBuffer, const Cal_SymKeyType *keyPtr)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC verification.
keyPtr	Holds a pointer to the key necessary for the MAC verification.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to initialize the MAC verify service of the CAL module.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-39 Cal_MacVerifyStart

4.2.37 Cal_MacVerifyUpdate

Prototype	
`Cal_ReturnType` **`Cal_MacVerifyUpdate`** `(Cal_ConfigIdType cfgId, Cal_MacVerifyCtxBufType contextBuffer, const uint8 *dataPtr, uint32 dataLength)`	
Parameter	
cfgId	Holds the identifier of the CAL module configuration which has to be used during the MAC verification.

dataPtr	Holds a pointer to the data for which a MAC shall be verified.
dataLength	Contains the number of bytes for which the MAC shall be verified.
contextBuffer	Holds the pointer to the buffer in which the context of this service can be stored.
Return code	
Cal_ReturnType	CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description	
This function shall be used to feed the MAC verification service with the input data.	
Particularities and Limitations	
none	
Call Context	
Context of the calling function	

Table 4-40 Cal_MacVerifyUpdate

4.2.38 Cal_MacVerifyFinish

Prototype
`Cal_ReturnType` **`Cal_MacVerifyFinish`** `(Cal_ConfigIdType cfgId, Cal_MacVerifyCtxBufType contextBuffer, const uint8 *MacPtr, uint32 MacLength, Cal_VerifyResultType *resultPtr)`
Parameter
cfgId — Holds the identifier of the CAL module configuration which has to be used during the MAC verification.
MacPtr — Holds a pointer to the memory location which will hold the MAC to verify.
MacLength — Holds the length of the MAC to be verified.
contextBuffer — Holds the pointer to the buffer in which the context of this service can be stored.
resultPtr — Holds a pointer to the memory location which will hold the result of the MAC verification.
Return code
Cal_ReturnType — CAL_E_OK - Request successful CAL_E_NOT_OK - Request failed
Functional Description
This function shall be used to finish the MAC verification service.
Particularities and Limitations
none
Call Context
Context of the calling function

Table 4-41 Cal_MacVerifyFinish

4.3 API used by CAL

In the following table services provided by other components, which are used by the CAL are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
CPL	Cpl_<Primitive>Start
CPL	Cpl_<Primitive>Update
CPL	Cpl_<Primitive>Finish
CPL	Cpl_<Primitive>

Table 4-42 Services used by the CAL

4.4 Callback Functions

The CAL functions do not provide any callback functions.

4.5 Configurable Interfaces

4.5.1 Notifications

The CAL module does not provide notifications for upper layers. The CAL routines are all synchronous.

4.5.2 Callout Functions

Further on the CAL module supports no callout function.

4.5.3 Hook Functions

The CAL supports no hook functions.

5 Configuration

In the CAL the attributes can be configured with the following methods:

> Configuration in DaVinci Configurator, for detailed description see 5.2

5.1 Configuration Variants

The CAL supports the configuration variants

> VARIANT-PRE-COMPILE

5.2 Configuration with DaVinci Configurator

5.2.1 Common Properties

Attribute Name	Values Default value is typed bold	Description
CalVersionInfoApi	**STD_ON** STD_OFF	Enables or disables Cal_GetVersionInfo API.
CalDisableNotConfiguredApis	**STD_ON** STD_OFF	If enabled, APIs of not configured services are unavailable.
CalMaxAlignScalarType	8 16 **32**	The scalar type which has the maximum alignment restrictions on the given platform. Valid values are: 8, 16, 32 Bit.

Table 5-1 Common Properties

5.2.2 Service Type related Properties

Depending on the type of service, the following parameters may configurable:

Attribute Name	Values Default value is typed bold	Description
Cal<ServiceType>MaxKeySize	1.. 4294967295	This is the maximum size over all key lengths used in all CPL primitives, which implement an <ServiceType> Service. Please note that the calling application has to provide the key buffer. So, it has to be ensured that the size of this buffer matches with the configured value here.
Cal<ServiceType>MaxCtxBufByteSize	1.. 4294967295	This is the maximum size over all context buffers used in all CPL primitives, which implement an <ServiceType> Service. Please note that the calling application has to provide the context buffer. So, it has to be ensured that the size of this buffer matches with the configured value here.

Table 5-2 Service Type related Properties

Note
For the methods internal storage, the buffer size will be always one max align type larger than entered.

5.2.3 Service specific Properties

Each service configuration has the following adjustable parameters:

Attribute Name	Description
Configuration Short Name	This container comprises the configuration of one <ServiceType> service. The container name serves as a symbolic name for the identifier of a service configuration.
<ServiceType> Init Configuration	This is the name of the C symbol, which contains the configuration of the underlying cryptographic primitive. Usually, this symbol represents a structure provided by the CPL module.
Name of <ServiceType> Primitive	This is the name of the cryptographic primitive to use. This name will be used to form the function pointers to the Start, Update and Finish functions of the corresponding cryptographic primitive according to the following rule: <name>_[Start|Update|Finish] Usually these functions are provided by the CPL module.
<ServiceType> Include File	Name of the Cpl header that shall be included for the specific service.
<ServiceType> CPL Reference[2]	Selecting a reference will populate the configuration fields.

Table 5-3 Service specific Properties

5.3 Configuration of CPL

The attributes of the CPL configuration have to be either set by generator or manually. For the definition and declaration of the configuration please read the specific CPL manual.

[2] This option is only available for selected services

6 AUTOSAR Standard Compliance

At the current version not all services specified by AUTOSAR are implemented, for details see Table 2-1 and Table 2-2.

6.1 Deviation from the AUTOSAR Software Specification

The MSR CAL ‘Include File Structure’ slightly differs to the AUTOSAR SWS, due to the adjustable context buffer sizes, the ‘Cal_Types.h’ have to include the generated ‘Cal_Cfg.h’. Furthermore the ‘Cal_Cfg.c’ has to include the specific ‘Cpl_<Service>.h’ headers to solve the linked symbols (cf. Chapter 3.1.3).

7 Glossary and Abbreviations

7.1 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
CAL	Crypto Abstraction Library
CPL	Crypto Primitive Library
RTE	Runtime Environment
SWC	Software Component
SWS	Software Specification

Table 7-1 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com