Document Title	Specification of CRC Routines
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	016
Document Classification	Standard

Document Version	4.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
14.10.2011	4.2.0	AUTOSAR Administration	• The GetVersionInfo API is always available
19.10.2010	4.1.0	AUTOSAR Administration	• New parameter added to APIs in order to chain CRC computations. • CRC check values corrected and checked values better explained. • CRC magic check added.
02.12.2009	4.0.0	AUTOSAR Administration	• Introduction of a new CRC-8 with the polynomial 2Fh • CRC-8 is now compliant to SAE J1850 • Legal disclaimer revised
23.06.2008	3.0.1	AUTOSAR Administration	• Separated CRC requirements from Memory Services Requirements • CRC8 management added
22.01.2008	3.0.0	AUTOSAR Administration	• Separated CRC requirements from Memory Services Requirements • CRC8 management added
31.10.2007	2.1.2	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
24.01.2007	2.1.1	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
15.12.2006	2.1.0	AUTOSAR Administration	• Crc_CalculateCRC16 and Crc_CalculateCRC32 APIs, Crc_DataPtr parameter : void pointer changed to uint8 pointer • Legal disclaimer revised
28.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template. • UML model introduction • Requirements traceability update • Reentrancy at calculating CRC with hardware support

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
31.05.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview 6
2 Acronyms and abbreviations 7
3 Related documentation 8
3.1 Input documents 8
3.2 Related standards and norms 8
4 Constraints and assumptions 9
4.1 Limitations 9
4.2 Applicability to car domains 9
5 Dependencies to other modules 10
5.1 File structure 10
6 Requirements traceability 11
7 Functional specification 15
7.1 Basic Concepts of CRC Codes 15
7.1.1 Mathematical Description 15
7.1.2 Euclidian Algorithm for Binary Polynomials and Bit-Sequences 17
7.1.3 CRC calculation, Variations and Parameter 18
7.2 Standard parameters 18
7.2.1 8-bit CRC calculation 19
7.2.1.1 8-bit SAE J1850 CRC Calculation 19
7.2.1.2 8-bit 0x2F polynomial CRC Calculation 19
7.2.2 16-bit CRC calculation 20
7.2.3 32-bit CRC calculation 20
7.3 General behavior 21
7.4 Error classification 21
7.5 Error detection 21
7.6 Error notification 21
7.7 Version check 21
7.8 Debugging concept 21
8 API specification 22
8.1 Imported types 22
8.2 Type definitions 22
8.3 Function definitions 22
8.3.1 8-bit CRC Calculation 23
8.3.1.1 8-bit SAE J1850 CRC Calculation 23
8.3.1.2 8-bit 0x2F polynomial CRC Calculation 23
8.3.2 16-bit CRC Calculation 24
8.3.3 32-bit CRC Calculation 25
8.3.4 Crc_GetVersionInfo 26
8.4 Call-back notifications 27
8.5 Scheduled functions 27
8.6 Expected Interfaces 27
8.6.1 Mandatory Interfaces 27

8.6.2 Optional Interfaces 27
8.6.3 Configurable interfaces 27
9 Sequence diagrams 28
9.1 Crc_CalculateCRC8 () 28
9.2 Crc_CalculateCRC8H2F () 28
9.3 Crc_CalculateCRC16() 28
9.4 Crc_CalculateCRC32() 29
10 Configuration specification 30
10.1 How to read this chapter 30
10.1.1 Configuration and configuration parameters 30
10.1.2 Variants 30
10.1.3 Containers 30
10.2 Containers and configuration parameters 31
10.2.1 Variants 31
10.2.2 Crc 31
10.2.3 CrcGeneral 32
10.3 Published Information 34
11 Changes to Release 1 35
11.1 Deleted SWS Items 35
11.2 Replaced SWS Items 35
11.3 Changed SWS Items 35
11.4 Added SWS Items 35
12 Changes during SWS Improvements by Technical Office 36
12.1 Deleted SWS Items 36
12.2 Replaced SWS Items 36
12.3 Changed SWS Items 36
12.4 Added SWS Items 36
13 Changes to Release 2 37
13.1 Deleted SWS Items 37
13.2 Replaced SWS Items 37
13.3 Changed SWS Items 37
13.4 Added SWS Items 37
14 Changes to Release 3 38
14.1 Deleted SWS Items 38
14.2 Replaced SWS Items 38
14.3 Changed SWS Items 38
14.4 Added SWS Items 38
15 Not applicable requirements 39

1 Introduction and functional overview

This specification specifies the functionality, API and the configuration of the AUTOSAR Basic Software module CRC.

The CRC library contains the following routines for CRC calculation:

- CRC8 SAEJ1850
- CRC8 0x2F polynomial
- CRC16
- CRC32

For all routines (CRC8, CRC8H2F, CRC16 and CRC32), the following calculation methods are possible:

- Table based calculation:
  Fast execution, but larger code size (ROM table)
- Runtime calculation:
  Slower execution, but small code size (no ROM table)
- Hardware supported CRC calculation (device specific):
  Fast execution, less CPU time

All routines are re-entrant and can be used by multiple applications at the same time. Hardware supported CRC calculation may be supported by some devices in the future.

2 Acronyms and abbreviations

Acronyms and abbreviations, which have a local scope and therefore are not contained in the AUTOSAR glossary, must appear in a local glossary.

Abbreviation / Acronym:	***Description:***
CRC	Cyclic Redundancy Check
ALU	Arithmetic Logic Unit

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[4] Requirements on Libraries,
AUTOSAR_SRS_Libraries.pdf

[5] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[6] A Painless Guide To CRC Error Detection Algorithms, Ross N. Williams

[7] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[8] Specification of Platform Types,
AUTOSAR_SWS_PlatformTypes.pdf

[9] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[10] Specification of C Implementation Rules,
AUTOSAR_TR_CImplementationRules.pdf

3.2 Related standards and norms

[7] SAE-J1850 8-bit CRC

[8] CCITT 16-bit CRC

[9] IEEE 802.3 Ethernet 32-bit CRC

4 Constraints and assumptions

4.1 Limitations

No known limitations.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

5.1 File structure

[CRC024] ⌈The Crc module shall provide the following files:

- C file `Crc_xxx.c` containing parts of CRC code
- An API interface `Crc.h` providing the function prototypes to access the library CRC functions
- A header file `Crc_Cfg.h` providing specific parameters for the CRC.

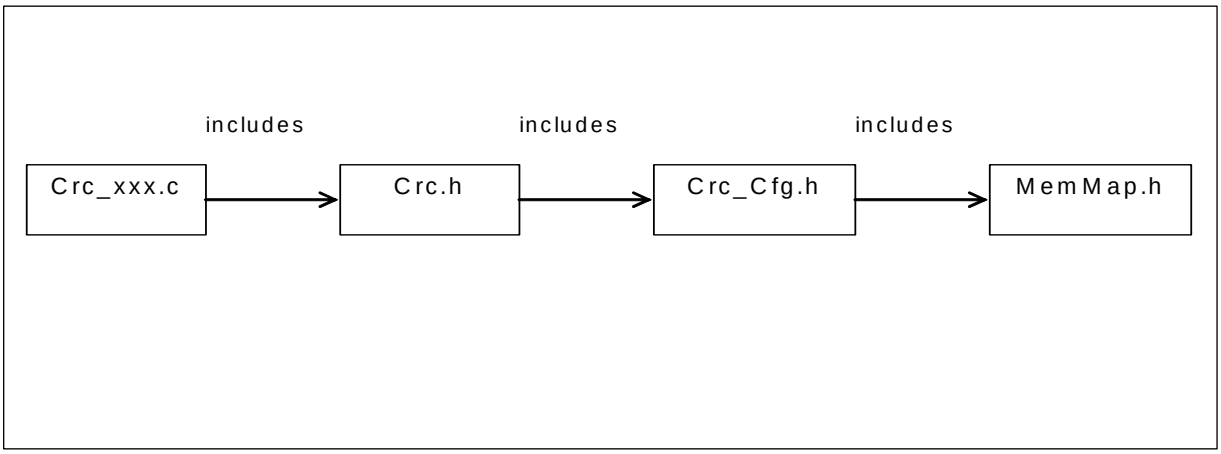


Figure 1: File structure

⌋( )

[CRC022] ⌈The Crc module shall comply with the following include file structure:

- `Crc.h` shall include `Crc_Cfg.h` and `MemMap.h`
- `Crc_xxx.c` shall include `Crc.h`⌋( )

[CRC023] ⌈Users of the Crc module (e.g. NVRAM Manager) shall only include `Crc.`⌋( )

6 Requirements traceability

Requirement	Satisfied by
-	CRC018
-	CRC044
-	CRC036
-	CRC014
-	CRC010
-	CRC015
-	CRC024
-	CRC041
-	CRC037
-	CRC033
-	CRC019
-	CRC031
-	CRC023
-	CRC042
-	CRC013
-	CRC017
-	CRC022
-	CRC020
-	CRC009
-	CRC032
-	CRC038
-	CRC039
-	CRC016
-	CRC045
-	CRC030
-	CRC043
-	CRC021
BSW00302	CRC051
BSW00304	CRC051
BSW00305	CRC051
BSW00306	CRC051
BSW00307	CRC051
BSW00308	CRC051
BSW00309	CRC051
BSW00312	CRC051
BSW00314	CRC051
BSW00321	CRC051
BSW00323	CRC051
BSW00325	CRC051

BSW00326	CRC051
BSW00327	CRC051
BSW00328	CRC051
BSW00330	CRC051
BSW00331	CRC051
BSW00333	CRC051
BSW00334	CRC051
BSW00335	CRC051
BSW00336	CRC051
BSW00337	CRC051
BSW00338	CRC051
BSW00339	CRC051
BSW00341	CRC051
BSW00342	CRC051
BSW00343	CRC051
BSW00344	CRC051
BSW00347	CRC051
BSW00348	CRC051
BSW00350	CRC051
BSW00353	CRC051
BSW00355	CRC051
BSW00358	CRC051
BSW00359	CRC051
BSW00360	CRC051
BSW00361	CRC051
BSW00369	CRC051
BSW00370	CRC051
BSW00371	CRC051
BSW00373	CRC051
BSW00375	CRC051
BSW00376	CRC051
BSW00378	CRC051
BSW00380	CRC051
BSW00383	CRC051
BSW00384	CRC051
BSW00385	CRC051
BSW00386	CRC051
BSW00387	CRC051
BSW00388	CRC051
BSW00389	CRC051
BSW00395	CRC051
BSW00398	CRC051

BSW00399	CRC051
BSW004	CRC005
BSW00400	CRC051
BSW00401	CRC051
BSW00402	CRC005
BSW00404	CRC051
BSW00405	CRC051
BSW00406	CRC051
BSW00407	CRC012, CRC011
BSW00409	CRC051
BSW00410	CRC051
BSW00411	CRC011
BSW00412	CRC051
BSW00414	CRC051
BSW00415	CRC051
BSW00416	CRC051
BSW00417	CRC051
BSW00420	CRC051
BSW00421	CRC051
BSW00422	CRC051
BSW00423	CRC051
BSW00424	CRC051
BSW00425	CRC051
BSW00427	CRC051
BSW00428	CRC051
BSW00429	CRC051
BSW00431	CRC051
BSW00432	CRC051
BSW00433	CRC051
BSW00434	CRC051
BSW005	CRC051
BSW006	CRC051
BSW007	CRC051
BSW009	CRC051
BSW010	CRC051
BSW08525	CRC002, CRC003
BSW101	CRC051
BSW160	CRC051
BSW161	CRC051
BSW162	CRC051
BSW164	CRC051
BSW168	CRC051

BSW170	CRC051
BSW172	CRC051
BSW324	CRC051

7 Functional specification

7.1 Basic Concepts of CRC Codes

7.1.1 Mathematical Description

Let D be a bitwise representation of data with a total number of n bit, i.e.

$D = (d_{n-1}, d_{n-2}, d_{n-3}, \ldots , d_1, d_0)$,

with $d_0, d_1, \ldots = 0b, 1b$. The corresponding *Redundant Code* C is represented by n+k bit as

$C = (D, R) = (d_{n-1}, d_{n-2}, d_{n-3}, \ldots , d_2, d_1, d_0, r_{k-1}, \ldots , r_2, r_1, r_0)$

with $r_0, r_1, \ldots$ = 0b,1b and R = ($r_{k-1}, \ldots , r_2, r_1, r_0$) The code is simply a concatenation of the data and the redundant part. (For our application, we will chose k = 16, 32 and n as a multiple of 16 resp. 32).

CRC-Algorithms are related to *polynomials* with coefficients in the finite *field of two element*, using arithmetic operations ⊕ and * according to the following tables.

The ⊕ operation is identified as the binary operation *exclusive-or*, that is usually available in the ALU of any CPU.

⊕	*0b*	*1b*
0b	0b	1b
1b	1b	0b

*	*0b*	*1b*
0b	0b	0b
1b	0b	1b

For simplicity, we will write ab instead of a*b

We introduce some examples for *polynomials* with coefficients in the *field of two elements* and give the simplified notation of it.

(ex. 1) $p_1(X) = 1b\ X^3 + 0b\ X^2 + 1b\ X^1 + 0b\ X^0 = X^3 + X$

(ex. 2) $p_2(X) = 1b\ X^2 + 1b\ X^1 + 1b\ X^0 = X^2 + X^1 + 1b$

Any code word, represented by n+k bit can be mapped to a polynomial of order n+k-1 with coefficients in the field of two elements. We use the intuitive mapping of the bits i.e.

$C(X) = d_{n-1}X^{k+n-1} + d_{n-2}X^{k+n-2} + \ldots + d_2X^{k+2} + d_1X^{k+1} + d_0\ X^k$
$+ r_{k-1}X^{k-1} + r_{k-2}Xk^{-2} + \ldots r_1\ X + r_0$

$C(X) = X^k(d_{n-1}X^{n-1} + d_{n-2}X^{n-2} + \ldots + d_2X^2 + d_1X^1 + d_0)$
$\mathbf{+ r_{k-1}X^{k-1} + r_{k-2}Xk^{-2} + \ldots r_1\ X + r_0}$

$C(X) = X^k D(X) \oplus R(X)$

This mapping is one-to-one.

A certain space CRC_G of *Cyclic Redundant Code Polynomials* is defined to be a multiple of a given *Generator Polynomial* $G(X) = X^k + g_{k-1} X^{k-1} + g_{k-2} X^{k-2} + \ldots + g_2 X^2 + g_1 X + g_0$. By definition, for any code polynomial C(X) in CRC_G there is a polynomial M(X) with

$$C(X) = G(X)\ M(X).$$

For a fixed irreducible (i.e. prime-) polynomial G(X), the mapping M(X) → C(X) is one-to-one. Now, how are data of a given codeword verified? This is basically a division of polynomials, using the *Euclidian Algorithm*. In practice, we are not interested in M(X), but in the *remainder* of the division, C(X) mod G(X). For a correct code word C, this remainder has to be *zero,* C(X) mod G(X) = 0. If this is not the case – there is an error in the codeword. Given G(X) has some additional algebraic properties, one can determine the error-location and correct the codeword.

Calculating the code word from the data can also be done with the Euclidian Algorithm. For a given data polynomial $D(x) = d_{n-1}X^{n-1} + d_{n-2}X^{n-2} + \ldots + d_1X^1 + d_0$ and the corresponding code polynomial C(X) we have

$$C(X) = X^k\ D(X) \oplus R(X) = M(X)\ G(X)$$

Performing the operation "mod G(X)" on both sides, one obtains

$$0 = C(X) \bmod G(X) = [X^k\ D(X)] \bmod G(X) \oplus R(X) \bmod G(X) \quad (*)$$

We denote that the order of the Polynomial R(X) is less than the order of G(X), so the modulo division gives zero with remainder R(X):

$$R(X) \bmod G(X) = R(X).$$

For polynomial R(X) with coefficients in the finite field with two elements we have the remarkable property R(X) + R(X) = 0. If we add R(X) on both sides of equation (*****) we obtain

$$R(X) = X^k\ D(X) \bmod G(X).$$

The important implication is that the redundant part of the requested code can be determined by using the Euclidian Algorithm for polynomials. At present, any CRC calculation method is a more or less sophisticated variation of this basic algorithm.

Up to this point, the propositions on CRC Codes are summarized as follows:

1. The construction principle of CRC Codes is based on polynomials with coefficients in the finite field of two elements. The ⊕ operation of this field is identical to the binary operation "xor" (exclusive or)
2. There is a natural mapping of bit-sequences into this space of polynomials.

3. Both calculation and verification of the CRC code polynomial is based on division modulo a given generator polynomial.
4. This generator polynomial has to have certain algebraic properties in order to achieve error-detection and eventually error-correction.

7.1.2 Euclidian Algorithm for Binary Polynomials and Bit-Sequences

Given a Polynomial $P_n(X) = p_nX^n + p_{n-1}X^{n-1} + \ldots + p_2X^2 + p_1X + p_0$ with coefficients in the finite field of two elements. Let $Q(X) = X^k + q_{k-1} X^{k-1} + q_{k-2} X^{k-2} + \ldots + q_2 X^2 + q_1 X + q_0$ be another polynomial of exact order k>0. Let $R_n(X)$ be the remainder of the polynomial division of maximum order k-1 and $M_n(X)$ corresponding so that

$R_n(X) \oplus M_n(X)\, Q(X) = P_n(X)$.

Euclidian Algorithm - Recursive

(Termination of recursion)
If n < k, then choose $R_n(X) = P_n(X)$ and $M_n = 0$.

(Recursion n+1 → n)
Let $P_{n+1}(X)$ be of maximum order n+1.
If n+1>= k calculate $P_n(X) = P_{n+1}(X) - p_{n+1}\, Q(X)\, X^{n-k+1}$. This polynomial is of maximum order n. Then

$$P_{n+1}(X) \bmod Q(X) = P_n(X) \bmod Q(X).$$

Proof of recursion

Choose $R_{n+1}(X) = P_{n+1}(X) \bmod Q(X)$ and $M_{n+1}(X)$ so that

$$R_{n+1}(X) \oplus M_{n+1}(X)\, Q(X) = P_{n+1}(X).$$

Then $R_{n+1}(X) - R_n(X) = P_{n+1}(X) - M_{n+1}(X)\, Q(X) - P_n(X) \oplus M_n(X)\, Q(X)$.

With $P_{n+1}(X) - P_n(X) = p_{n+1}\, Q(X)\, X^{n-k+1}$ we obtain

$R_{n+1}(X) - R_n(X) = p_{n+1}\, Q(X)\, X^{n-k+1} + M_n(X)\, Q(X) - M_{n+1}(X)\, Q(X)$

$R_{n+1}(X) - R_n(X) = Q(X)\, [p_{n+1}\, X^{n-k+1} + M_n(X) - M_{n+1}(X)]$

On the left side, there is a polynomial of maximum order k-1. On the right side Q(X) is of exact order k. This implies that both sides are trivial and equal to zero. One obtains

$R_{n+1}(X) = R_n(X)$ (1)
$M_{n+1}(X) = M_n(X) + p_{n+1}\, X^{n-k+1}$ (2)

(end of proof)

Example

$P(X) = P_4(X) = X^4 + X^2 + X + 1b$; $Q(X) = X^2 + X + 1b$; n = 4; k = 2
$P_3(X) = X^4 + X^2 + X + 1b - 1b(X^2 + X + 1b)\ X^2 = X^3 + X + 1b.$
$P_2(X) = X^3 + X + 1b - 1b\ X\ (X^2 + X + 1b) = X^2 + 1b.$
$P_1(X) = X^2 + 1 - 1b\ (X^2 + X + 1) = X$
$R(X) = P(X) \bmod Q(X) = R_1(X) = P_1(X) = X.$

7.1.3 CRC calculation, Variations and Parameter

Based on the Euclidian Algorithm, some variations have been developed in order to improve the calculation performance. All these variations do not improve the capability to detect or correct errors – the so-called Hamming Distance of the resulting code is determined only by the generator polynomial. Variations simply optimize for different implementing ALUs.

CRC-Calculation methods are characterized as follows:

1. Rule for Mapping of Data to a bit sequence (d_{n-1}, d_{n-2}, d_{n-3}, … , d_1, d_0) and the corresponding data polynomial D(X) (standard or reflected data).
2. Generator polynomial G(X)
3. Start value and corresponding Polynomial S(X)
4. Appendix A(X), also called XOR-value for modifying the final result.
5. Rule for mapping the resulting CRC-remainder R(X) to codeword. (Standard or reflected data)

The calculation itself is organized in the following steps

- Map Data to D(X)
- Perform Euclidian Algorithm on $X^k\ D(X) + X^{n-k-1}\ S(X) + A(X)$ and determine $R(X) = [\ X^k\ D(X) + X^{n-k-1}\ S(X) + A(X)] \bmod G(X)$
- Map D(X), R(X) to codeword

7.2 Standard parameters

This section gives a rough overview on the standard parameters that are commonly used for 8-bit, 16-bit and 32-bit CRC calculation.

- CRC result width: Defines the result data width of the CRC calculation.
- Polynomial: Defines the generator polynomial which is used for the CRC algorithm.
- Initial value: Defines the start condition for the CRC algorithm.
- Input data reflected: Defines whether the bits of each input byte are reflected before being processed.

- Result data reflected: Similar to “Input data reflected” this parameter defines whether the bits of the CRC result are reflected.
- XOR value: This Value is XORed to the final register value before the value is returned as the official checksum.
- Check: This field is a check value that can be used as a weak validator of implementations of the algorithm. The field contains the checksum obtained when the ASCII values '1' '2' '3' '4' '5' '6' '7' '8' '9' corresponding to values 31h 32h 33h 34h 35h 36h 37h 38h 39h is fed through the specified algorithm.
- Magic check: The CRC checking process calculates the CRC over the entire data block, including the CRC result. An error-free data block will always result in the unique constant polynomial (magic check) - representing the CRC-result XORed with 'XOR value'- regardless of the data block content.

Example of magic check: calculation of SAE-J1850 CRC8 (see detailed parameters in CRC030) over data bytes 00h 00h 00h 00h:

- CRC generation: CRC over 00h 00h 00h 00h, start value FFh:
  - CRC-result = 59h
- CRC check: CRC over 00h 00h 00h 00h 59h, start value FFh:
  - CRC-result = 3Bh
  - Magic check = CRC-result XORed with ‘XOR value’:
    C4h = 3Bh xor FFh

7.2.1 8-bit CRC calculation

7.2.1.1 8-bit SAE J1850 CRC Calculation

[CRC030] ⌈The Crc_CalculateCRC8() function of the CRC module shall implement the CRC8 routine based on the SAE-J1850 CRC8 Standard:

CRC result width:	8 bits
Polynomial:	1Dh
Initial value:	FFh
Input data reflected:	No
Result data reflected:	No
XOR value:	FFh
Check:	4Bh
Magic check:	C4h

⌋( )

7.2.1.2 8-bit 0x2F polynomial CRC Calculation

[CRC042] ⌈The Crc_CalculateCRC8H2F() function of the CRC module shall implement the CRC8 routine based on the generator polynomial 0x2F:

CRC result width:	8 bits
Polynomial:	2Fh
Initial value:	FFh
Input data reflected:	No
Result data reflected:	No
XOR value:	FFh
Check:	DFh
Magic check:	42h

⌋( )

7.2.2 16-bit CRC calculation

[CRC002] ⌈The CRC module shall implement the CRC16 routine based on the CCITT CRC16 Standard:

CRC result width:	16 bits
Polynomial:	1021h
Initial value:	FFFFh
Input data reflected:	No
Result data reflected:	No
XOR value:	0000h
Check:	29B1h
Magic check:	0000h

⌋(BSW08525)

7.2.3 32-bit CRC calculation

[CRC003] ⌈The CRC module shall implement the CRC32 routine based on the IEEE-802.3 CRC32 Ethernet Standard:

CRC result width:	32 bits
Polynomial:	04C11DB7h
Initial value:	FFFFFFFFh
Input data reflected:	Yes
Result data reflected:	Yes
XOR value:	FFFFFFFFh
Check:	CBF43926h
Magic check*:	DEBB20E3h

***Important note:** To match the magic check value, the CRC must be appended in little endian format, i.e. low significant byte first. This is due to the reflections of the input and the result.⌋(BSW08525)

7.3 General behavior

Data blocks are passed to the CRC routines using the parameters “start address”, “size” and “start value”. The return value is the CRC result.

7.4 Error classification

The CRC library functions do not provide any error classification. CRC recalculation and comparison must be done by each module in the upper layer (e.g. NVRAM Manager).

7.5 Error detection

The CRC library functions do not provide any error detection mechanism.

7.6 Error notification

The CRC library functions do not provide any error notification.

7.7 Version check

[CRC005] ⌈`Crc.c` shall check if the correct version of `Crc.h` is included. This shall be done by a preprocessor check of the version number `CRC_MAJOR_VERSION` and `CRC_MINOR_VERSION.`⌋ (BSW00402, BSW004)

7.8 Debugging concept

[CRC036] ⌈Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable.⌋ ( )

[CRC037] ⌈All type definitions of variables that shall be debugged shall be accessible by the header file CrC.h.⌋ ( )

[CRC038] ⌈The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"sizeof".⌋ ( )

[CRC039] ⌈Variables available for debugging shall be described in the respective Basic Software Module Description.⌋ ( )

8 API specification

8.1 Imported types

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

[CRC018] ⌈

Module	*Imported Type*
Std_Types	Std_VersionInfoType

⌋ **( )**

8.2 Type definitions

None.

8.3 Function definitions

[CRC013] ⌈If CRC routines are to be used as a library, the CRC modules' implementer shall develop the CRC module in a way that only those parts of the CRC code that are used by other modules are linked into the final binary.⌋ ( )

[CRC041] ⌈When calculating a CRC-result in a single call, the call should use 'Initial value' as start value.⌋ ( )

[CRC014] ⌈When calculating a CRC-result using multiple calls, the first call should use 'Initial value' as start value and then for subsequent calls, the start value shall be the result of the previous call XORed with 'XOR value' and reflected if the parameter '*Result data reflected*' is '*Yes*'.
The function is specified in a way the user will not have to perform the rework of the start value but will indicate if it is the first call or the subsequent ones.
⌋ ( )

Example: calculation of CRC32 Ethernet Standard (see detailed parameters in CRC003) over data bytes 01h 02h 03h 04h 05h 06h 07h 08h:

- In one function call, CRC over 01h 02h 03h 04h 05h 06h 07h 08h, start value FFFFFFFFh:
  - CRC-result = 3FCA88C5h (final value)
- In two function calls:
  - CRC over 01h 02h 03h 04h, start value FFFFFFFFh:
    - CRC-result of first call = B63CFBCDh (intermediate value)
  - CRC over 05h 06h 07h 08h, start value: B63CFBCDh xor *XOR value* (FFFFFFFFh) = 49C30432h and after reflection: 4C20C392h
    - CRC-result of final call = 3FCA88C5h (final value)

The following C-code example shows that the caller modifies the start value by using the previous result (without any rework) and indicates that it is no more the first call:

```
InterResult = Crc_CalculateCRC32(&Array12345678[0], 4, 0xFFFFFFFF, TRUE);
result = Crc_CalculateCRC32(&Array12345678[4], 4, InterResult, FALSE);
```

8.3.1 8-bit CRC Calculation

8.3.1.1 8-bit SAE J1850 CRC Calculation

[CRC031] ⌈

Service name:	Crc_CalculateCRC8	

Syntax:	`uint8 Crc_CalculateCRC8(`
	`  const uint8* Crc_DataPtr,`
	`  uint32 Crc_Length,`
	`  uint8 Crc_StartValue8,`
	`  boolean Crc_IsFirstCall`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Crc_DataPtr	Pointer to start address of data block to be calculated.
	Crc_Length	Length of data block to be calculated in bytes.
	Crc_StartValue8	Start value when the algorithm starts.
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8. FALSE: Subsequent call in a call sequence; Crc_StartValue8 is interpreted to be the return value of the previous function call.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	uint8	8 bit result of CRC calculation.
Description:	This service makes a CRC8 calculation on Crc_Length data bytes, with SAE J1850 parameters	

⌋()

[CRC032] ⌈The function `Crc_CalculateCRC8` shall perform a CRC8 calculation on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue8.` ⌋()

[CRC033] ⌈If the CRC calculation within the function `Crc_CalculateCRC8` is performed by hardware, then the CRC module's implementer shall ensure reentrancy of this function by implementing a (software based) locking mechanism. ⌋()

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC8` in order to decrease the calculation time.

The function `Crc_CalculateCRC8` requires specification of configuration parameters defined in CRC006.

8.3.1.2 8-bit 0x2F polynomial CRC Calculation

[CRC043] ⌈

Service name:	Crc_CalculateCRC8H2F

Syntax:	`uint8 Crc_CalculateCRC8H2F(`
	`  const uint8* Crc_DataPtr,`
	`  uint32 Crc_Length,`
	`  uint8 Crc_StartValue8H2F,`
	`  boolean Crc_IsFirstCall`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Crc_DataPtr	Pointer to start address of data block to be calculated.
	Crc_Length	Length of data block to be calculated in bytes.
	Crc_StartValue8H2F	Start value when the algorithm starts.
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue8H2F. FALSE: Subsequent call in a call sequence; Crc_StartValue8H2F is interpreted to be the return value of the previous function call.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	uint8	8 bit result of CRC calculation.
Description:	This service makes a CRC8 calculation with the Polynomial 0x2F on Crc_Length	

⌋()

[CRC044] ⌈The function `Crc_CalculateCRC8H2F` shall perform a CRC8 calculation with the polynomial 0x2F on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue8H2F.` ⌋()

[CRC045] ⌈If the CRC calculation within the function `Crc_CalculateCRC8H2F` is performed by hardware, then the CRC module's implementer shall ensure reentrancy of this function by implementing a (software based) locking mechanism. ⌋()

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC8H2F` in order to decrease the calculation time.

The function `Crc_CalculateCRC8H2F` requires specification of configuration parameters defined CRC006.

8.3.2 16-bit CRC Calculation

[CRC019]⌈

Service name:	Crc_CalculateCRC16

Syntax:	`uint16 Crc_CalculateCRC16(`
	`  const uint8* Crc_DataPtr,`
	`  uint32 Crc_Length,`
	`  uint16 Crc_StartValue16,`
	`  boolean Crc_IsFirstCall`
	`)`

Service ID[hex]:	0x02
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	Crc_DataPtr	Pointer to start address of data block to be calculated.
	Crc_Length	Length of data block to be calculated in bytes.
	Crc_StartValue16	Start value when the algorithm starts.
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue16. FALSE: Subsequent call in a call sequence; Crc_StartValue16 is interpreted to be the return value of the previous function call.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	uint16	16 bit result of CRC calculation.
Description:	This service makes a CRC16 calculation on Crc_Length data bytes.	

⌋()

[CRC015] ⌈The function `Crc_CalculateCRC16` shall perform a CRC16 calculation on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue16.` ⌋()

[CRC009] ⌈If the CRC calculation within the function `Crc_CalculateCRC16` is performed by hardware, then the CRC module's implementer shall ensure reentrancy of this function by implementing a (software based) locking mechanism. ⌋()

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC16` in order to decrease the calculation time.

The function `Crc_CalculateCRC16` requires specification of configuration parameters defined in CRC006.

8.3.3 32-bit CRC Calculation

[CRC020] ⌈

Service name:	Crc_CalculateCRC32	

Syntax:	`uint32 Crc_CalculateCRC32(`
	`  const uint8* Crc_DataPtr,`
	`  uint32 Crc_Length,`
	`  uint32 Crc_StartValue32,`
	`  boolean Crc_IsFirstCall`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Crc_DataPtr	Pointer to start address of data block to be calculated.
	Crc_Length	Length of data block to be calculated in bytes.
	Crc_StartValue32	Start value when the algorithm starts.
	Crc_IsFirstCall	TRUE: First call in a sequence or individual CRC calculation; start from initial value, ignore Crc_StartValue32. FALSE: Subsequent call in a call sequence; Crc_StartValue32 is interpreted to be the return value of the previous function call.
Parameters (in-out):	None	

Parameters (out):	None	
Return value:	uint32	32 bit result of CRC calculation.
Description:	This service makes a CRC32 calculation on Crc_Length data bytes.	

⌋( )

[CRC016] ⌈The function `Crc_CalculateCRC32` shall perform a CRC32 calculation on `Crc_Length` data bytes, pointed to by `Crc_DataPtr`, with the starting value of `Crc_StartValue32.` ⌋( )

[CRC010] ⌈If the CRC calculation within the function `Crc_CalculateCRC32` is performed by hardware, then the CRC module's implementer shall ensure reentrancy of this function by implementing a (software based) locking mechanism. ⌋( )

Note: If large data blocks have to be calculated (>32 bytes, depending on performance of processor platform), the table based calculation method should be configured for the function `Crc_CalculateCRC32` in order to decrease the calculation time.

The function `Crc_CalculateCRC32` requires specification of configuration parameters defined in CRC006.

8.3.4 Crc_GetVersionInfo

[CRC021] ⌈

Service name:	Crc_GetVersionInfo	
Syntax:	`void Crc_GetVersionInfo( Std_VersionInfoType* Versioninfo )`	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	Versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module.	

⌋( )

[CRC011] ⌈The function `Crc_GetVersionInfo` shall return the version information of the CRC module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋(BSW00407, BSW00411)

[CRC017] ⌈If source code for caller and callee of the function Crc_GetVersionInfo is available, the CRC module should realize this function as a macro, defined in the modules header file. ⌋(BSW00407, BSW00411)

8.4 Call-back notifications

None.

8.5 Scheduled functions

This chapter lists all functions called directly by the Basic Software Module Scheduler.

The Crc module does not have scheduled functions.

8.6 Expected Interfaces

In this chapter, all interfaces required from other modules are listed.

8.6.1 Mandatory Interfaces

None

8.6.2 Optional Interfaces

None.

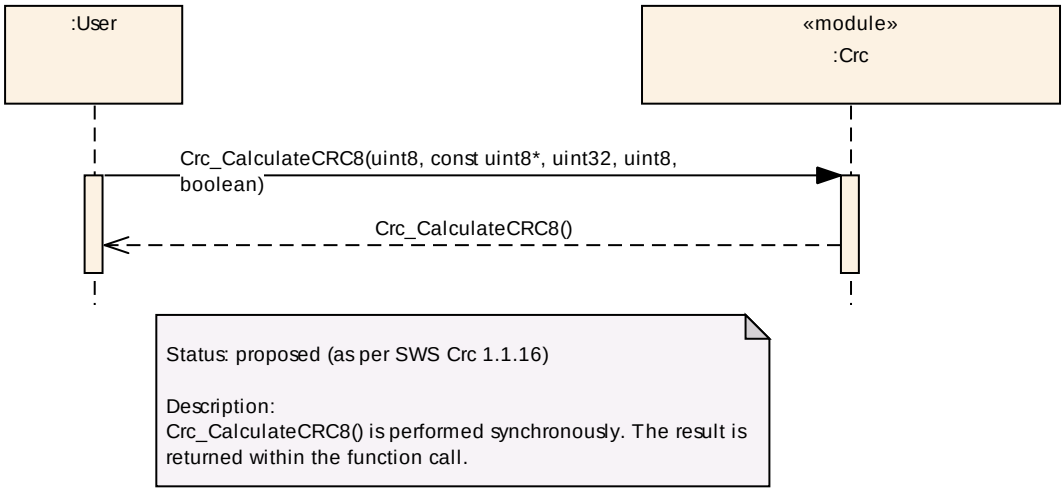
8.6.3 Configurable interfaces

None.

9 Sequence diagrams

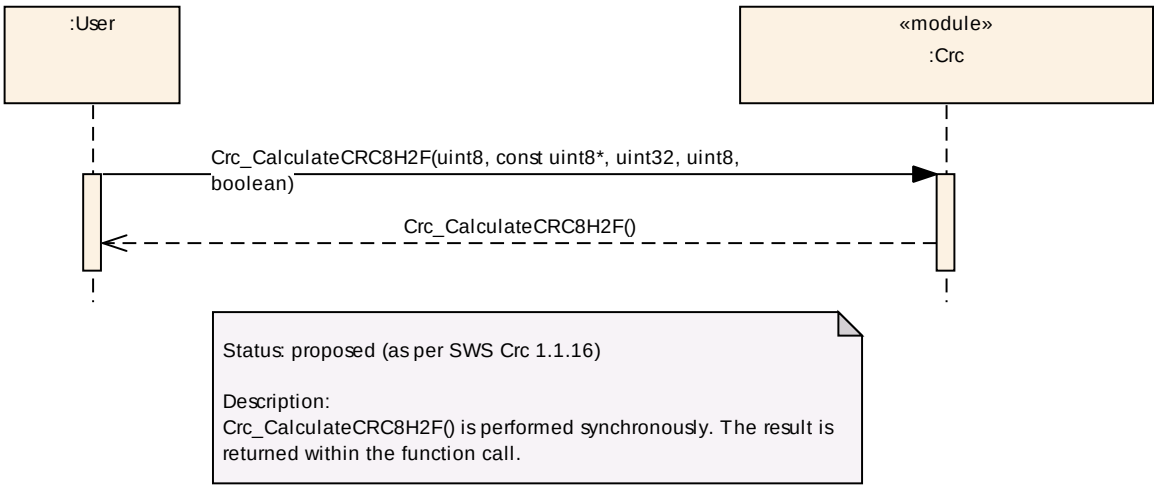
9.1 Crc_CalculateCRC8 ()

The following diagram shows the synchronous function call `Crc_CalculateCRC8`.



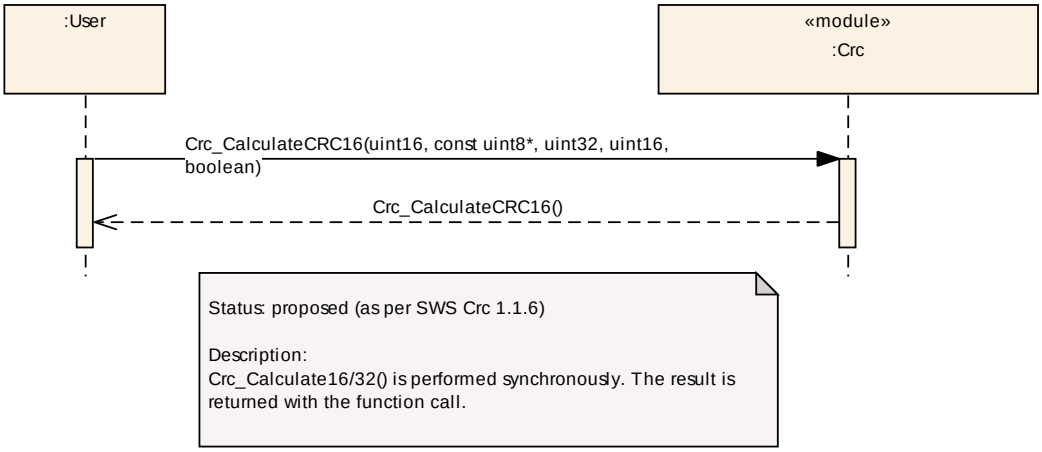
9.2 Crc_CalculateCRC8H2F ()

The following diagram shows the synchronous function call `Crc_CalculateCRC8H2F`.



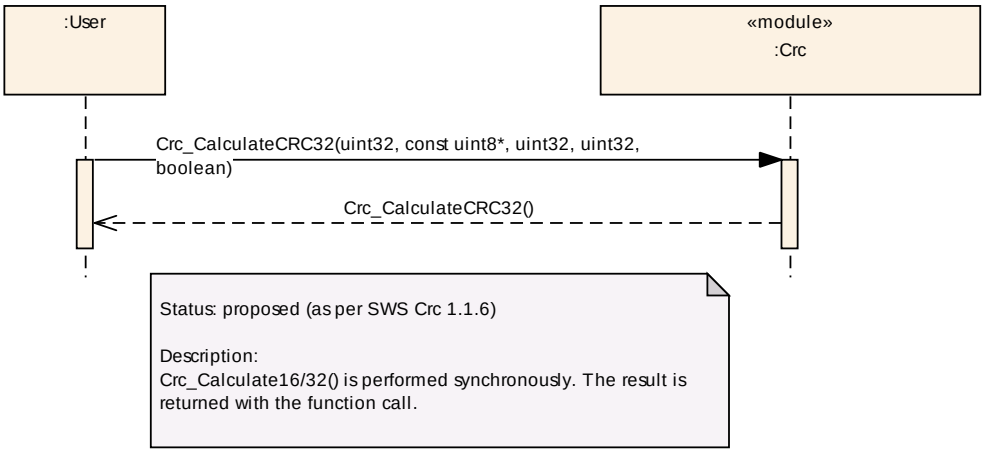
9.3 Crc_CalculateCRC16()

The following diagram shows the synchronous function call `Crc_CalculateCRC16`.



9.4 Crc_CalculateCRC32()

The following diagram shows the synchronous function call `Crc_CalculateCRC32`.



10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Architecture [2]
- AUTOSAR ECU Configuration Specification [5]:
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

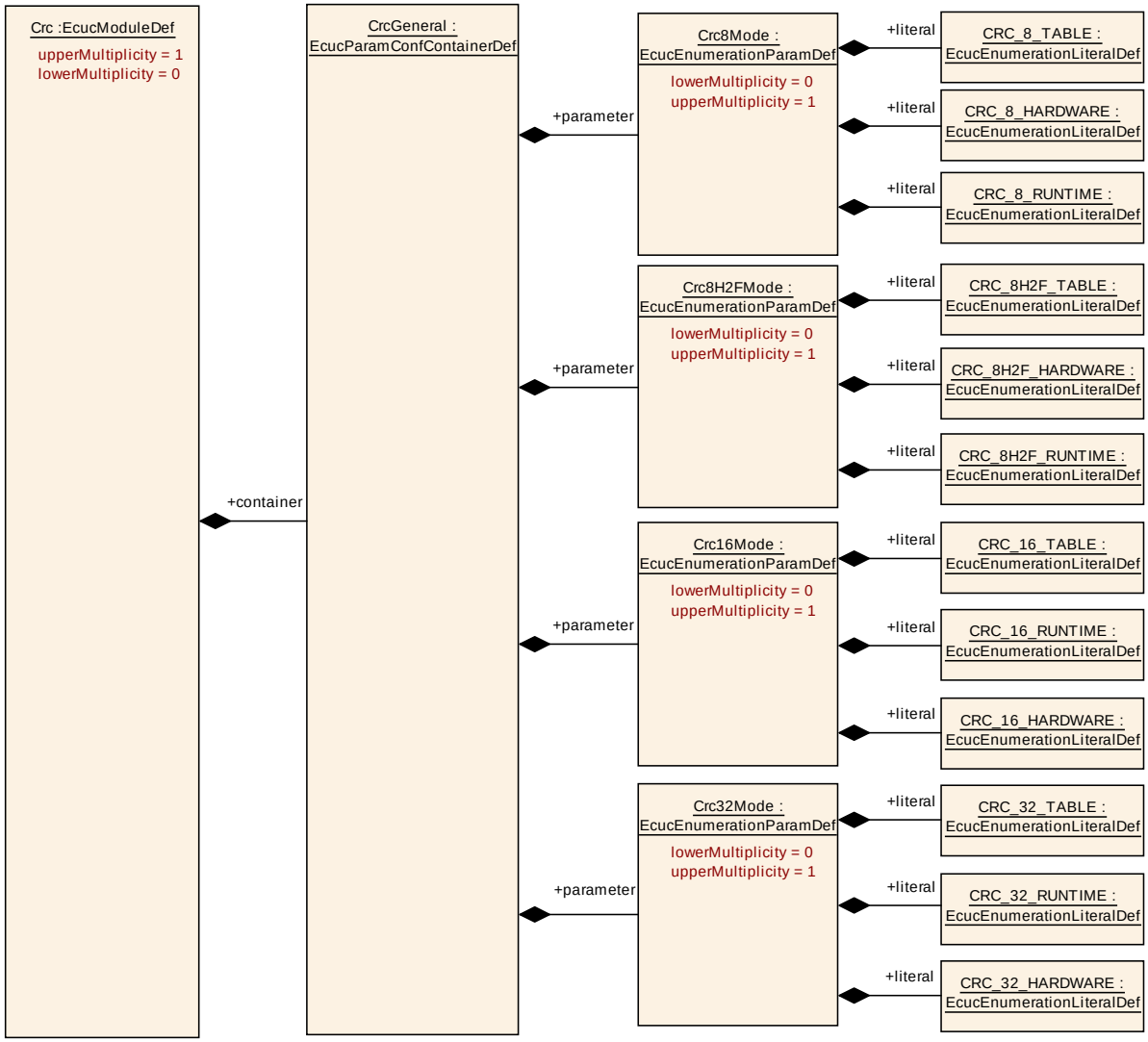
10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapters 7 and Chapter 8.



10.2.1 Variants

[CRC040] ⌈VARIANT-PRE-COMPILE: Only parameters with "Pre-compile time" configuration are allowed in this variant.⌋ ( )

10.2.2 Crc

Module Name	*Crc*
Module Description	Configuration of the Crc (Crc routines) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CrcGeneral	1	General configuration of CRC module

10.2.3 CrcGeneral

SWS Item	**CRC006_Conf :**
Container Name	CrcGeneral{CRC_COMMON}
Description	General configuration of CRC module
Configuration Parameters	

SWS Item	**CRC025_Conf :**		
Name	Crc16Mode {CRC_16_MODE}		
Description	Switch to select one of the available CRC 16-bit (CCITT) calculation methods		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CRC_16_HARDWARE	hardware based CRC16 calculation	
	CRC_16_RUNTIME	runtime based CRC16 calculation	
	CRC_16_TABLE	table based CRC16 calculation (default selection)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**CRC026_Conf :**		
Name	Crc32Mode {CRC_32_MODE}		
Description	Switch to select one of the available CRC 32-bit (IEEE-802.3 CRC32 Ethernet Standard) calculation methods		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CRC_32_HARDWARE	hardware based CRC32 calculation	
	CRC_32_RUNTIME	runtime based CRC32 calculation	
	CRC_32_TABLE	table based CRC32 calculation (default selection)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**CRC031_Conf :**	
Name	Crc8H2FMode {CRC_8H2F_MODE}	
Description	Switch to select one of the available CRC 8-bit (2Fh polynomial) calculation methods	
Multiplicity	0..1	
Type	EcucEnumerationParamDef	
Range	CRC_8H2F_HARDWARE	hardware based CRC8H2F calculation
	CRC_8H2F_RUNTIME	runtime based CRC8H2F calculation
	CRC_8H2F_TABLE	table based

		CRC8H2F calculation (default selection)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**CRC030_Conf :**		
Name	Crc8Mode {CRC_8_MODE}		
Description	Switch to select one of the available CRC 8-bit (SAE J1850) calculation methods		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CRC_8_HARDWARE	hardware based CRC8 calculation	
	CRC_8_RUNTIME	runtime based CRC8 calculation	
	CRC_8_TABLE	table based CRC8 calculation (default selection)	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 Published Information

[CRC050] ⌈The standardized common published parameters as required by BSW00402 in the SRS General on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].⌋ ( )

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

SWS Item	CRC048	
Information elements		
Information element name	***Type / Range***	***Information element description***
Crc_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
Crc _MODULE_ID	#define/ uint8	Module ID of this module from Module List
Crc _AR_MAJOR_VERSION	#define/ uint8	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
Crc _AR_MINOR_VERSION	#define/ uint8	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
Crc _AR_PATCH_VERSION	#define/ uint8	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
Crc _SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
Crc _SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
Crc _SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

11 Changes to Release 1

11.1 Deleted SWS Items

No deleted SWS items to Release 1.

11.2 Replaced SWS Items

No replaced SWS items to Release 1.

11.3 Changed SWS Items

No changed SWS items to Release 1.

11.4 Added SWS Items

SWS Item	*Rationale*
CRC009	
CRC010	
CRC011	Added due to adaptation to new SWS template.
CRC012	Added due to adaptation to new SWS template.
CRC013	
CRC014	
CRC015	
CRC016	
CRC016	
CRC016	
CRC016	

12 Changes during SWS Improvements by Technical Office

12.1 Deleted SWS Items

SWS Item	*Rationale*
CRC001	SWS Improvement: No requirement on CRC module but generic information.
CRC007	SWS Improvement: No requirement on CRC module but generic information.
CRC008	SWS Improvement: No requirement.

12.2 Replaced SWS Items

No replaced SWS items.

12.3 Changed SWS Items

No changed SWS items.

12.4 Added SWS Items

SWS Item	*Rationale*
CRC017	Hint for Crc_GetVersionInfo
CRC018	UML Model linking of imported types
CRC019	UML Model linking of Crc_CalculateCRC16
CRC020	UML Model linking of Crc_CalculateCRC32
CRC021	UML Model linking of Crc_GetVersionInfo
CRC022	Gave ID to existing text
CRC023	Gave ID to existing text
CRC024	Gave ID to existing text
CRC025	Gave ID to existing text
CRC026	Gave ID to existing text
CRC027	Gave ID to existing text
CRC028	Gave ID to existing text
CRC029	Gave ID to existing text

13 Changes to Release 2

13.1 Deleted SWS Items

No deleted SWS items.

13.2 Replaced SWS Items

No replaced SWS items.

13.3 Changed SWS Items

No changed SWS items.

13.4 Added SWS Items

SWS Item	*Rationale*
CRC026	Add support for CRC8
CRC027	Add support for CRC8
CRC028	Add support for CRC8
CRC029	Add support for CRC8

14 Changes to Release 3

14.1 Deleted SWS Items

No deleted SWS items.

14.2 Replaced SWS Items

SWS Item	*Rationale*
CRC026 by CRC030	CRC026 item identifier was duplicated (bug #25136)
CRC027 by CRC031	CRC027 item identifier was duplicated (bug #25136)
CRC028 by CRC032	CRC028 item identifier was duplicated (bug #25136)
CRC029 by CRC033	CRC029 item identifier was duplicated (bug #25136)

14.3 Changed SWS Items

SWS Item	*Rationale*
CRC021	Asterisk is missing for pointer parameter in Crc_GetVersionInfo() (bug #25131)
CRC030	CRC8 parameters changed (bug #23034)
CRC014	CRC calculation with splitted blocks example (bug #32417)
CRC034	Crc8Mode range definitions needs to be changed (bug #32578)

14.4 Added SWS Items

SWS Item	*Rationale*
CRC034	Gave ID to existing text
CRC035	Gave ID to existing text
CRC036	Add Debugging concept
CRC037	Add Debugging concept
CRC038	Add Debugging concept
CRC039	Add Debugging concept
CRC040	Change of Variant requirement descriptions required (bug #22603)
CRC041	CRC calculation on a single call
CRC042	Crc_CalculateCRC8H2F polynomial parameters
CRC043	Crc_CalculateCRC8H2F API description
CRC044	Crc_CalculateCRC8H2F function description
CRC045	Crc_CalculateCRC8H2F hardware implementation description
CRC046	Crc8H2FMode configuration parameter description
CRC047	CrcInitialValue8H2F configuration parameter description
CrC001_PI	Rework of Published Information

15 Not applicable requirements

[CRC051] ⌈ These requirements are not applicable to this specification.⌋ (BSW00344, BSW00404, BSW00405, BSW170, BSW00380, BSW00412, BSW00383, BSW00384, BSW00387, BSW00388, BSW00389, BSW00395, BSW00398, BSW00399, BSW00400, BSW00401, BSW00375, BSW101, BSW00416, BSW00406, BSW168, BSW00423, BSW00424, BSW00425, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00336, BSW00337, BSW00338, BSW00369, BSW00339, BSW00421, BSW00422, BSW00420, BSW00417, BSW00323, BSW00409, BSW00385, BSW00386, BSW161, BSW162, BSW324, BSW005, BSW00415, BSW164, BSW00325, BSW00326, BSW00342, BSW00343, BSW160, BSW007, BSW00347, BSW00305, BSW00307, BSW00373, BSW00327, BSW00335, BSW00350, BSW00410, BSW00314, BSW00370, BSW00348, BSW00353, BSW00361, BSW00302, BSW00328, BSW00312, BSW006, BSW00304, BSW00355, BSW00378, BSW00306, BSW00308, BSW00309, BSW00371, BSW00358, BSW00414, BSW00376, BSW00359, BSW00360, BSW00330, BSW00331, BSW009, BSW172, BSW010, BSW00333, BSW00321, BSW00341, BSW00334)