MICROSAR ComStackLib

Technical Reference

ComStackLib based BSW generators

Version 2.00.02

Authors	Gunnar Meiss
Status	Released

Document Information

History

Author	Date	Version	Remarks
Gunnar Meiss	2013-03-25	1.00.00	initial version
Gunnar Meiss	2013-08-23	1.01.00	ESCAN00068919 Remove <MSN>UseSignedDataTypesInIndexArrays ESCAN00070017 Remove <MSN>_Resource.xml
Gunnar Meiss	2014-10-06	2.00.00	ESCAN00078776 AR4-698: Post-Build Selectable (Identity Manager)
Gunnar Meiss	2014-12-19	2.00.01	ESCAN00080380 Minor typing and grammar corrections
Gunnar Meiss	2016-03-30	2.00.02	ESCAN00089127 Extend MD_CSL_3355_3356 with the aspects of the PRQA Rule 3358 and 3359 ESCAN00089126 Support a justification for PRQA Rule 310 and PCSymbolicNonDereferenciateablePointers Added chapter Freedom from Interference

Reference Documents

No.	Source	Title	Version
[1]	Vector	Compliance Documentation MISRA-C:2004 / MICROSAR	2.2.0

Scope of the Document

This technical reference describes the general use of the ComStackLib based BSW generators.

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

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00.00	Support of embedded data generation in the IMPLEMENTATION-CONFIG-VARIANT VARIANT-PRE-COMPILE
2.00.00	Support of the IMPLEMENTATION-CONFIG-VARIANT VARIANT-POST-BUILD-LOADABLE
3.00.00	Revision of existing techniques
4.00.00	Revision of existing techniques
5.00.00	AR4-698: Post-Build Selectable (Identity Manager)
6.00.00	Support VTT
7.00.00	Support Techniques to ensure Freedom of Interference
8.00.00	Java 8

Table 1-1 Component history

2 Introduction

This document describes the configuration of ComStackLib based BSW generators.

Supported AUTOSAR Release*:	4
Supported Configuration Variants:	PRE-COMPILE [SELECTABLE] POST-BUILD-LOADABLE [SELECTABLE]

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

The ComStackLib is an embedded data generation engine designed for AUTOSAR based BSW software. Generating embedded software is situated in the context of different aspects.

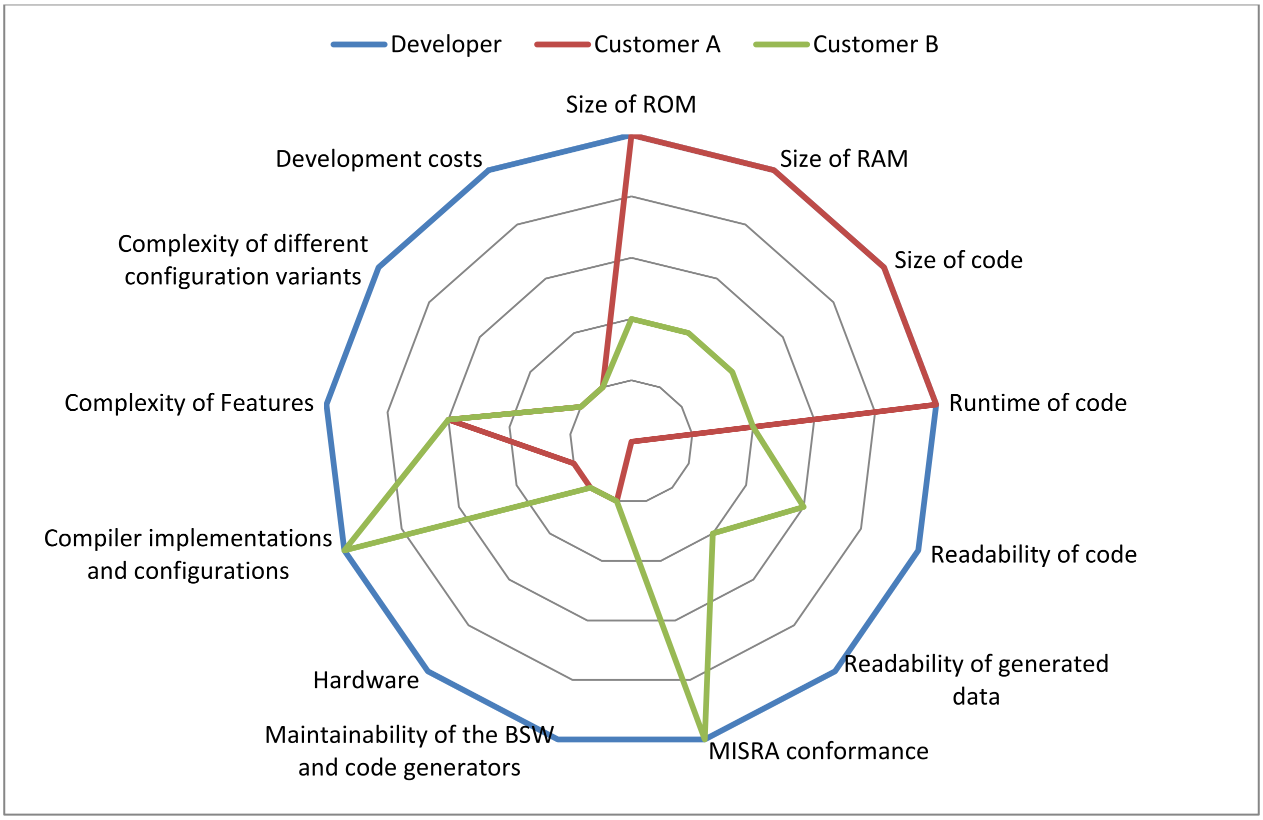

Figure 2-1 Embedded Code Aspects

The number of aspects for embedded software is quite high and they have a various importance from the view of different stakeholders. Some aspects contradict to each other and other aspects cannot be changed at the time of the project. Due to this the ComStackLib has been introduced as scalable embedded data generation engine designed for AUTOSAR.

2.1 Architecture Overview

The following figure shows where the ComStackLib is used in the MICROSAR architecture.

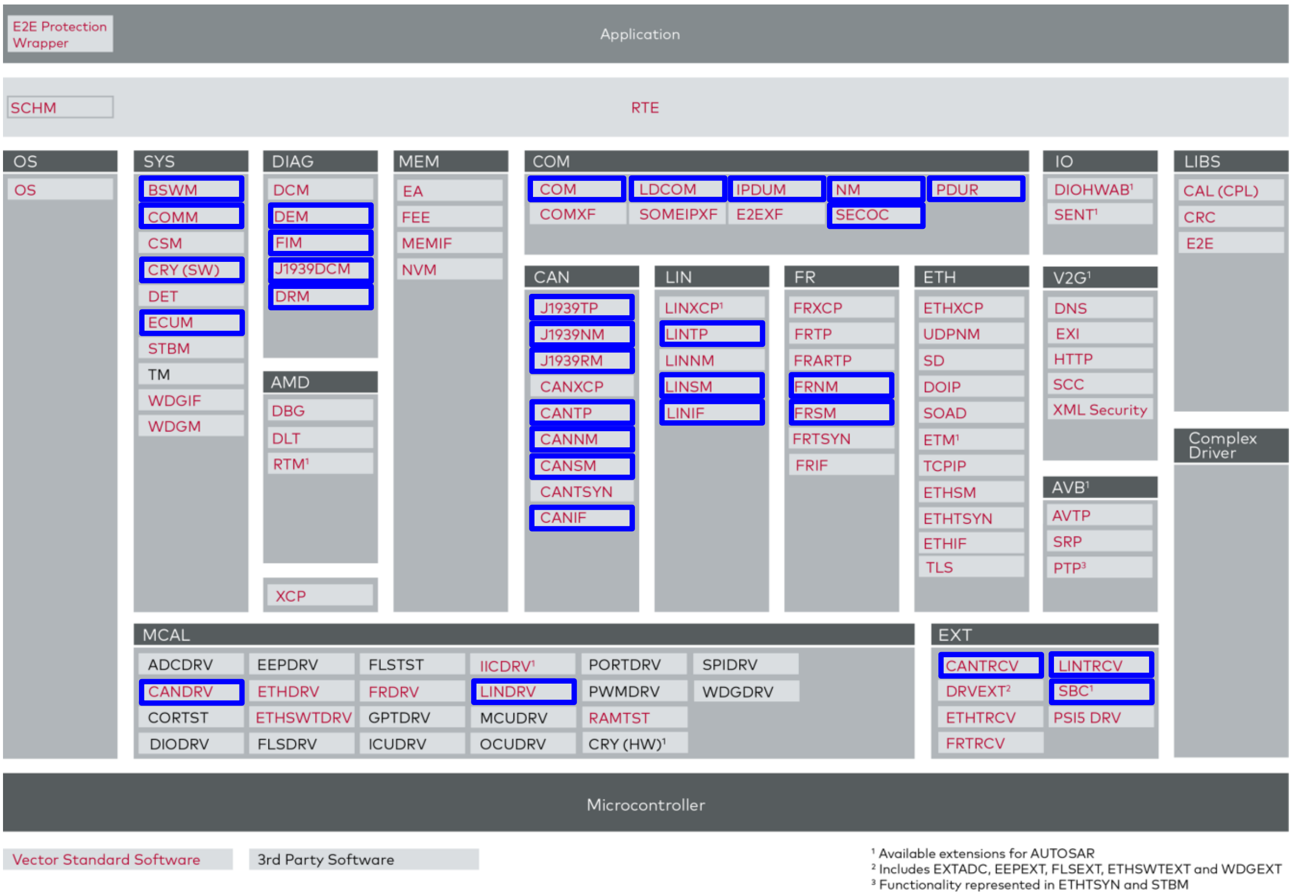

Figure 2-2 AUTOSAR 4.2 Architecture Overview

3 Functional Description

This chapter gives necessary information for the tailoring of the MICROSAR ComStackLib based software into your environment. Figure 3-1 Resources in compiler optimization variants shows the resource consumption of two different ECUs combined with different compiler optimization levels. The compiler is not able to influence the size of CONST and VAR data. The embedded software developer is in charge to reduce the CONST and VAR data consumption.

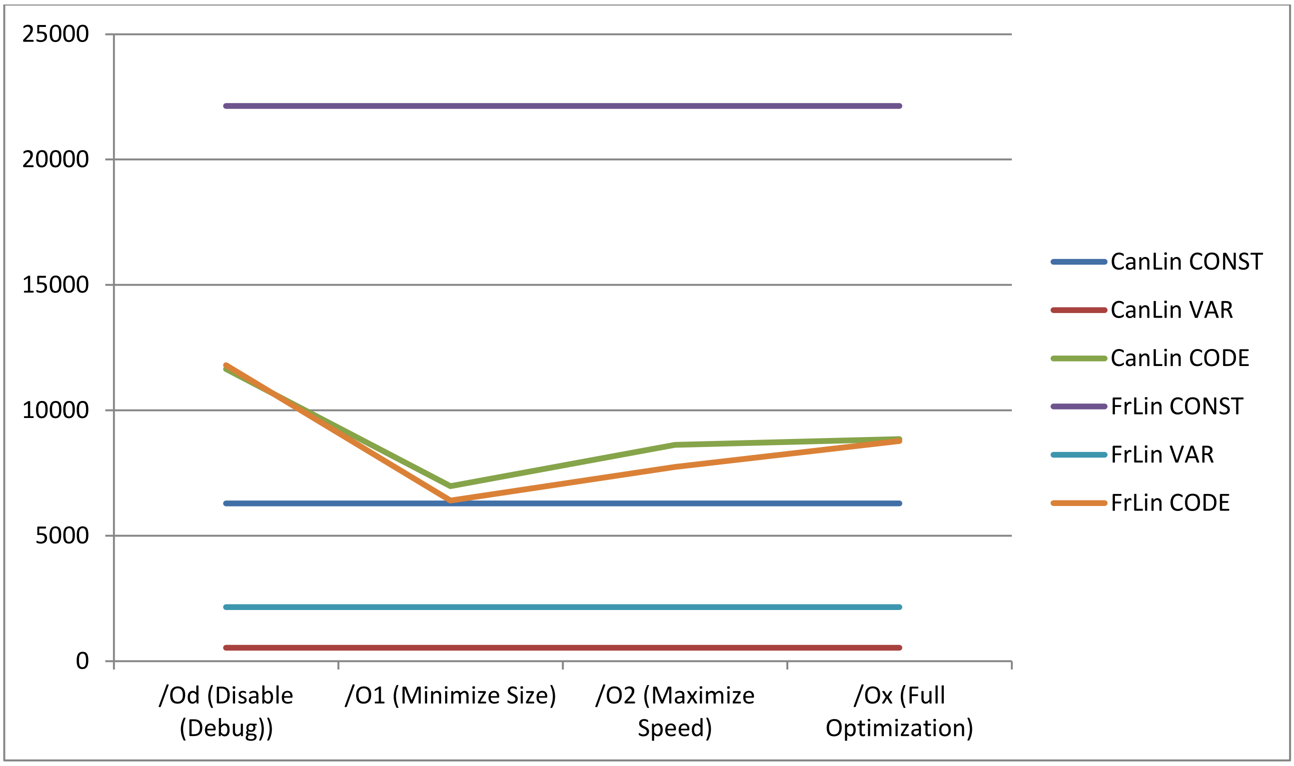

Figure 3-1 Resources in compiler optimization variants

3.1 CONFIG-CLASS of Data

The code generator has an internal knowledge of the required CONFIG-CLASS. Due to this data can be moved dependent on the IMPLEMENTATION-CONFIG-VARIANT. See chapter 4.2 IMPLEMENTATION-CONFIG-VARIANT dependent Data.

3.2 CONFIG-CLASS PRE-COMPILE Optimizations

3.2.1 Optimize Const Data to Defines

Set the configuration parameters <MSN>OptimizeConstVars2Define and <MSN>OptimizeConstArrays2Define to TRUE to optimize automatically CONST data in the CONFIG-CLASS PRE-COMPILE to a define.

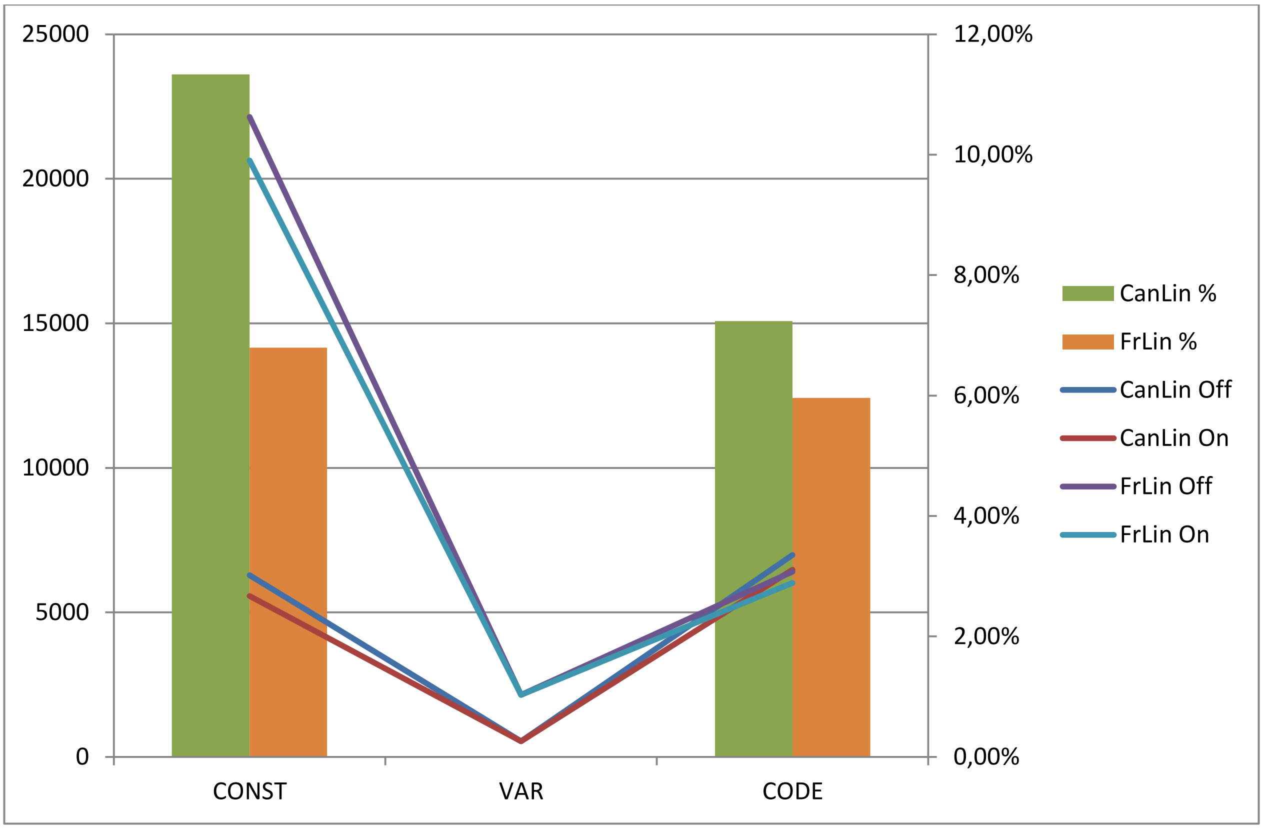

Figure 3-2 Using defines for CONST data

The optimization effect depends on the available configuration data. CONST and CODE size in the ECU can be reduced.

3.2.2 Optimize Data Types

Every generated data element generated with the ComStackLib has an own C data type. Due to this, the data type itself can be calculated automatically as small as possible. Set <MSN>MinimizeNumericalDataTypes to MINIMIZE_NUMERICAL_DATA_TYPES to calculate the data types as small as possible.

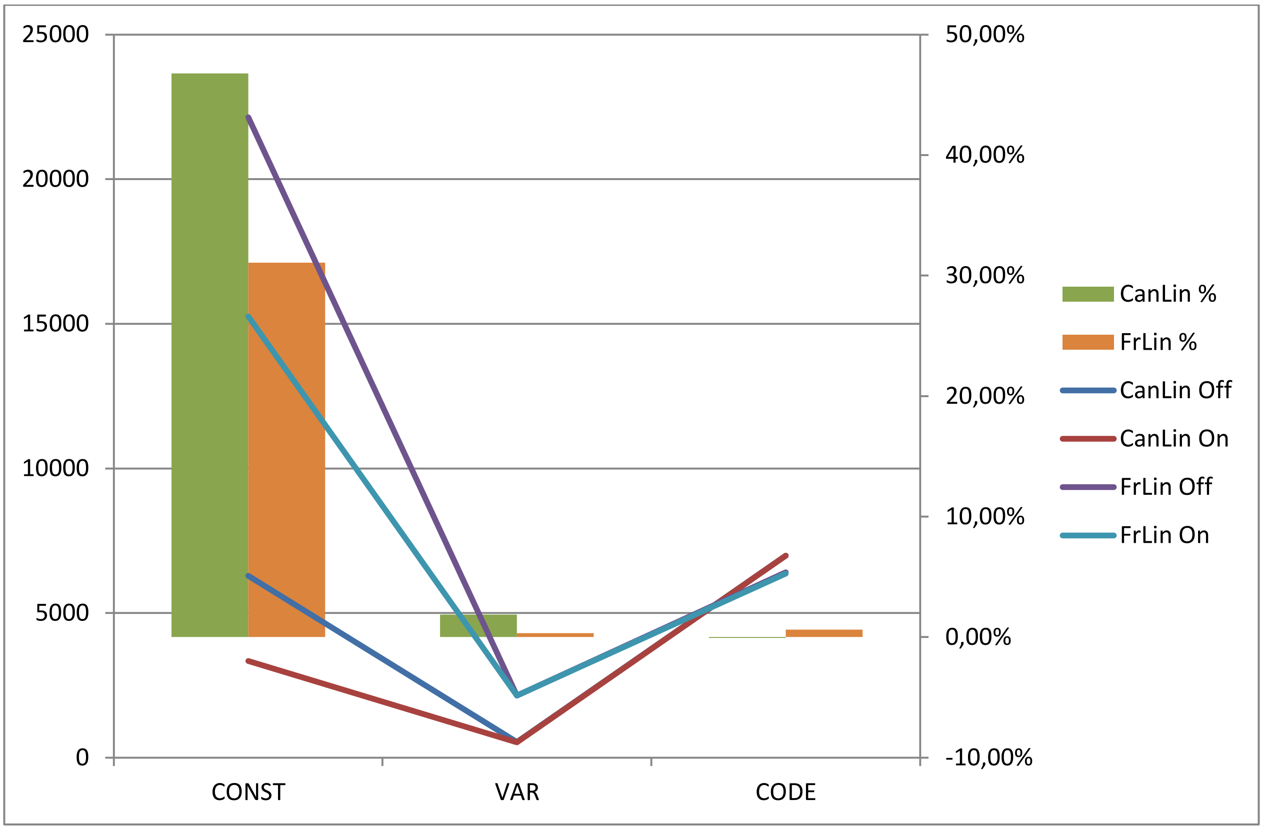

Figure 3-3 Data type minimization

The usage of data type minimization saves CONST, VAR and CODE size.

3.2.3 Optimize Bool Data in Structs

Boolean data can be represented differently in C structs. Due to this, the generation of boolean data can be configured with <MSN>StructBoolDataUsage as BOOLEAN, BITFIELD and BITMASKING. There is nearly no difference between the usage of different bit data types.

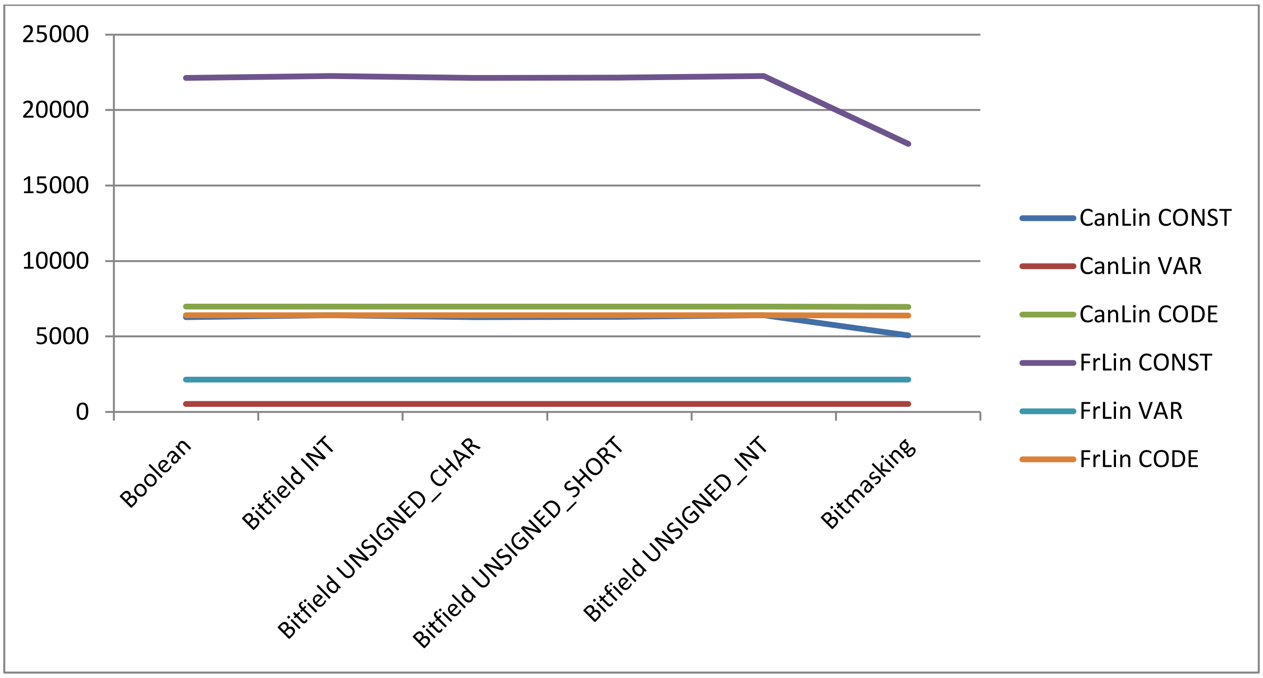

Figure 3-4 Boolean struct data variants

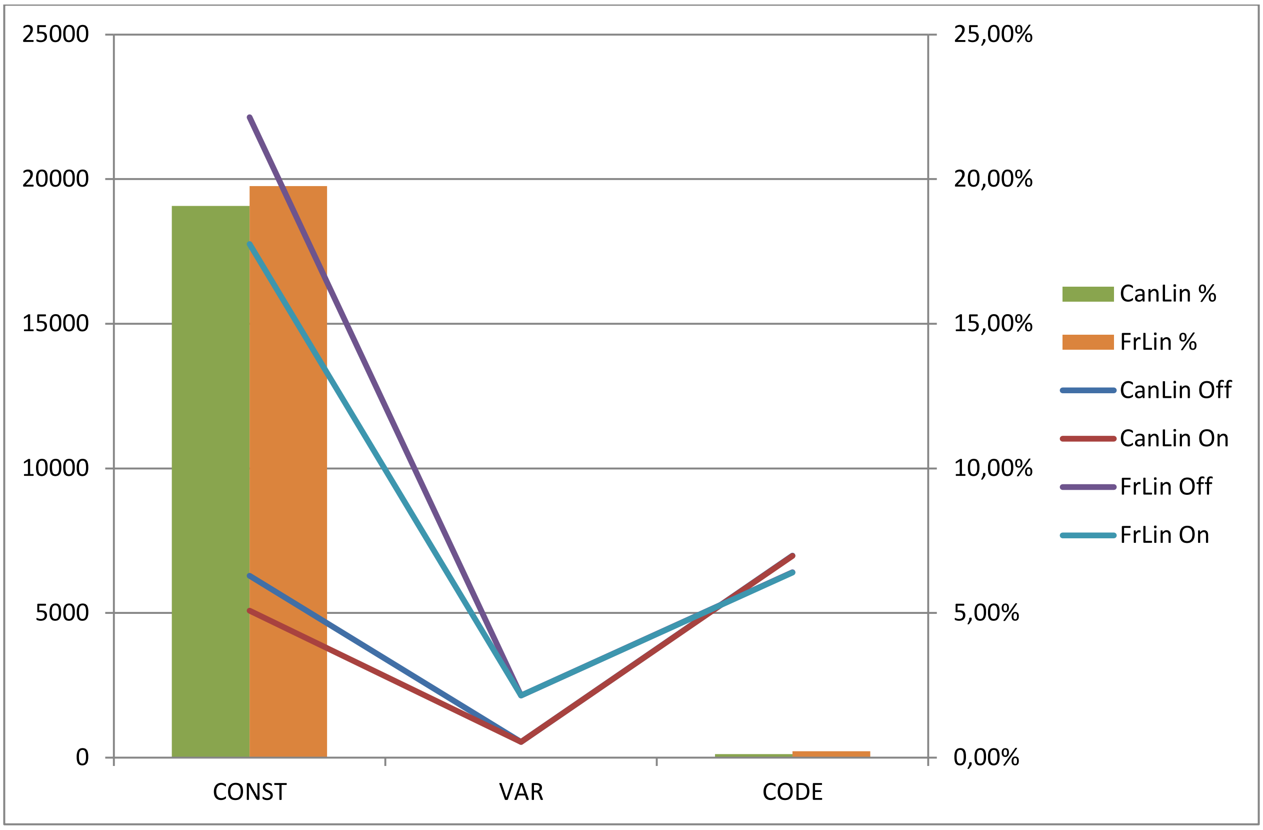

Figure 3-5 Boolean struct data versus Bitmasking

The usage of BITMASKING reduces the CONST size. The increase of the CODE size is so tiny, that it can be omitted.

3.2.4 Data Deduplication and Reduction

Data deduplication and reduction is a typical way to reduce the amount of data. The ComStackLib provides generic algorithms which implement typical data deduplication mechanisms.

3.2.4.1 Equal Data

Identical data can be deduplicated by redirection of the data access to other data. There is no influence to the runtime of the embedded software.

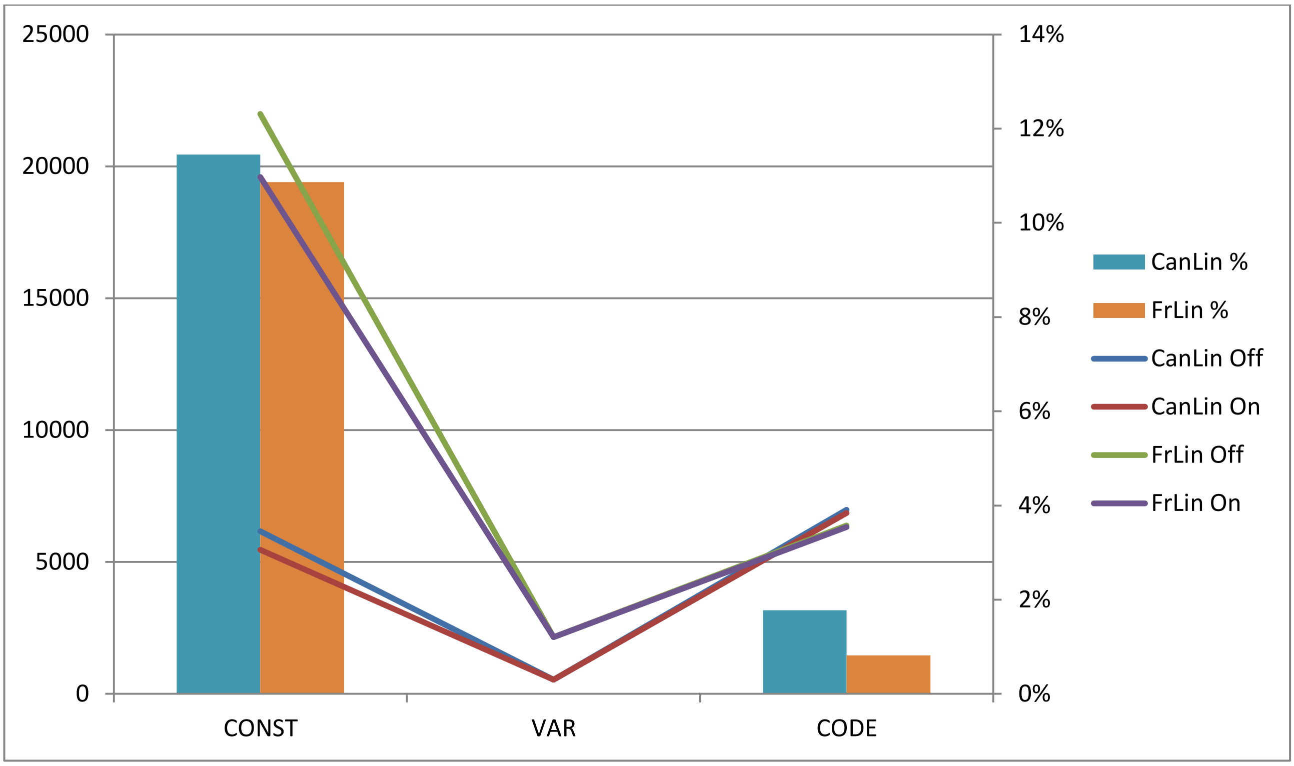

Figure 3-6 Data deduplication without operations

3.2.4.2 Unary and Binary Operations

Data can be reduced by using unary operations or operations on constants or operations on other data elements. The operations are located in the data access layer. Due to this, the code itself remains as implemented. This reduction has influence to the runtime of the embedded software.

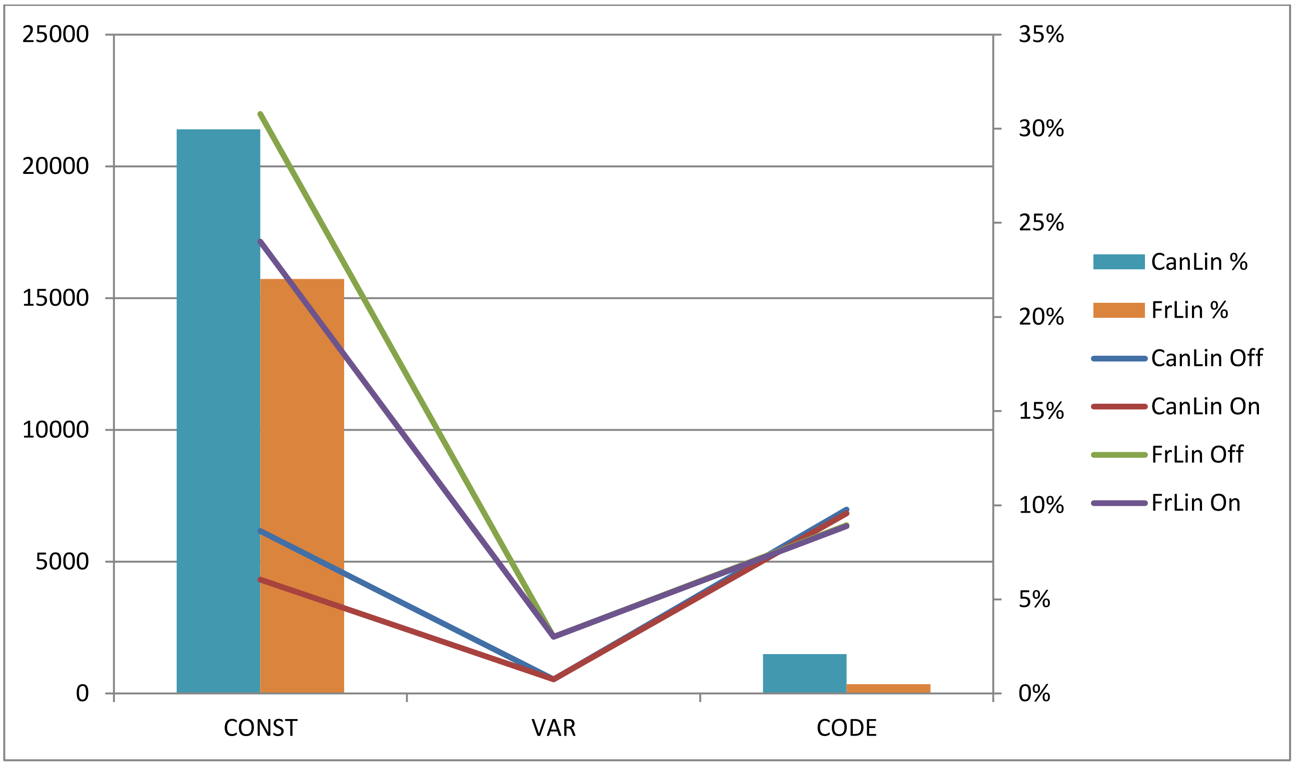

Figure 3-7 Data deduplication with operations

3.2.5 Data Streaming

Data can be packed into multiple streams of basic data types and identical parts can be overlapped with and without data offsets. The data access layer redirects to the dependent data index. There is no influence to the runtime of the embedded software, but the data compression rate is quite high in large configurations and complex modules containing lots of data.

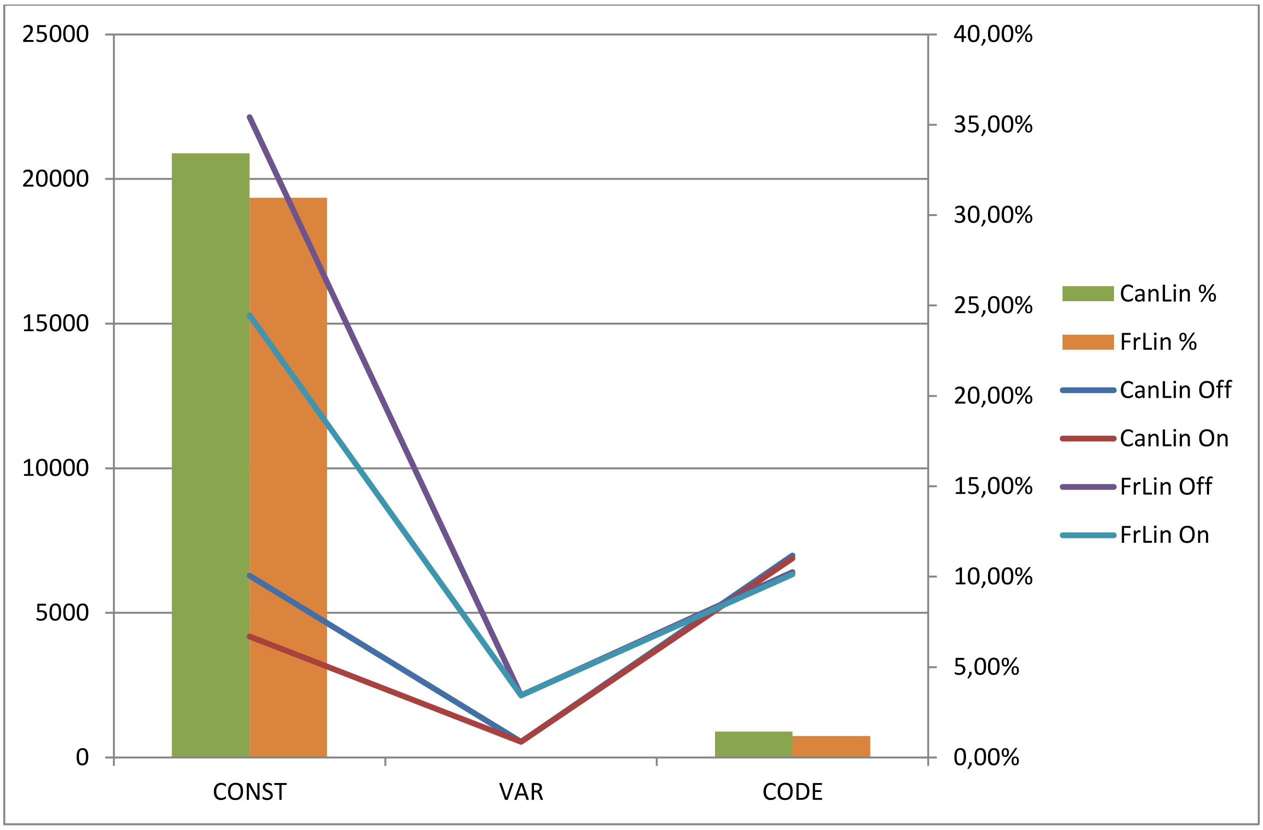

Figure 3-8 Data Streaming

3.3 CONFIG-CLASS Independent Optimizations

3.3.1 Sort Struct Elements

C structs are always sorted depending on the size of an element data type. Sorting structure elements reduces the number of padding bytes added by the compiler to align the data.

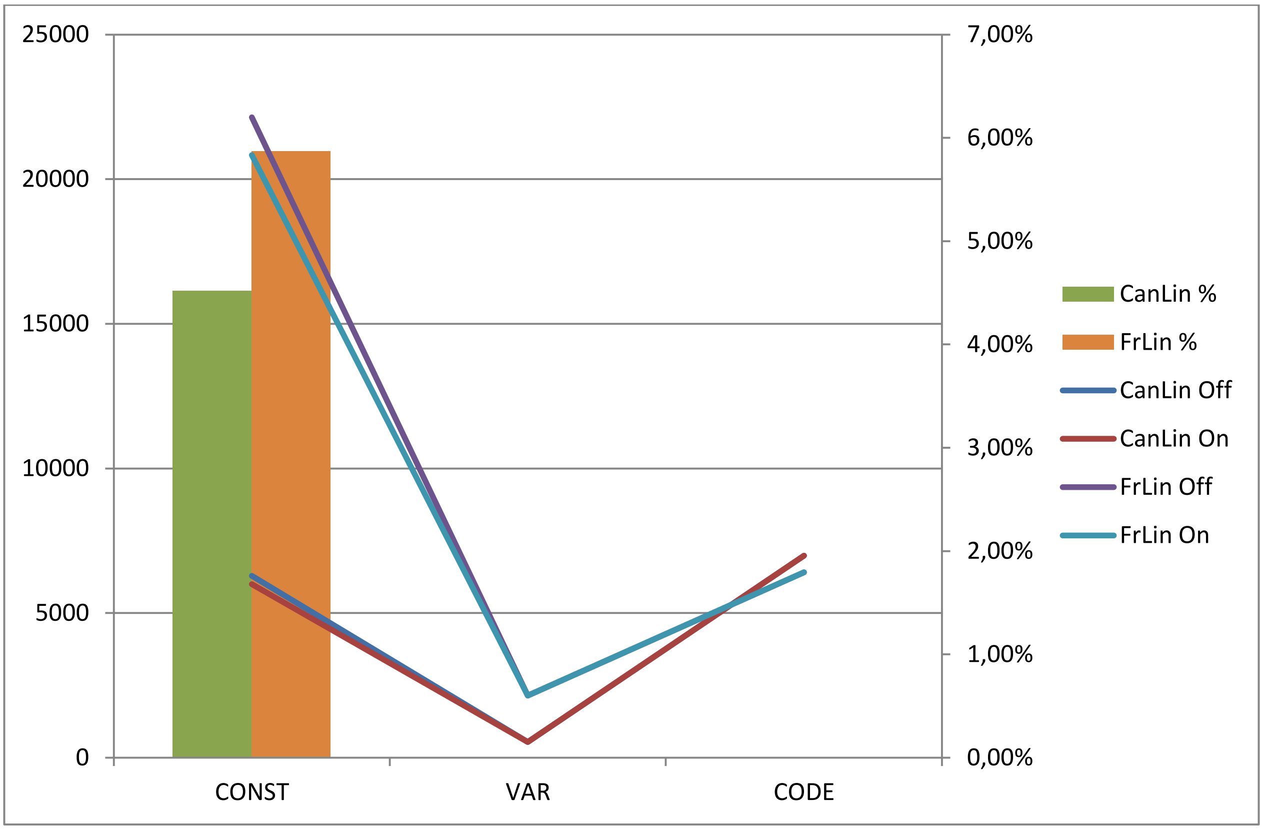

Figure 3-9 Sorting struct elements

3.4 SELECTABLE Optimizations

If the configuration variant is SELECTABLE based the following optimizations are automatically performed.

3.4.1 Merge of VAR and CONST Based Data

All VAR based generated data is merged between different predefined variants.

Example

A predefined variant LEFT_ECU needs a VAR based array of the type uint8 with 10 elements and predefined variant RIGHT_ECU needs a VAR based array of the type uint8 with 6 elements in the same context. The result is a variant independent generated VAR based array of the type uint8 with 10 elements.

Due to this, if the BSW configuration data is identical in different predefined variants, the module configuration is completely merged.

3.5 Freedom from Interference

The generated data elements are wrapped by the generated data access. Writing out of bounds in VAR arrays is a typically trap in software programming. To avoid overriding other variables, there are two safety strategies implemented. The strategy can be configured globally.

> **Index checking:** the data access checks the used index against the generator known size and values are not manipulated if the index value is out of bounds problem.

> **Index saturation:** VAR arrays are blown up to the next 2 n size and the used index is saturated by a mask value. Due to this, there is no out of bounds problem, but values in other indexes can be manipulated.

4 Integration

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

4.1 Dynamic Files

The dynamic files are generated by the configuration tool CFG5 for ComStackLib based BSW software.

File Name	Description
<MSN>_Cfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS PRE-COMPILE data.
<MSN>_Cfg.c	This file is generated dependent on the used code generator for compatibility reasons and contains if generated: > local constant macros > local function macros > local data types and structures > local data prototypes > local data > global data of CONFIG-CLASS PRE-COMPILE data.
<MSN>_Lcfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS LINK data.
<MSN>_Lcfg.c	This file contains: > local constant macros > local function macros > local data types and structures > local data prototypes > local data > global data of CONFIG-CLASS LINK and PRE-COMPILE data if the <MSN>_Cfg.c is

File Name	Description
	not generated.
<MSN>_PBcfg.h	This file contains: > global constant macros > global function macros > global data types and structures > global data prototypes > global function prototypes of CONFIG-CLASS POST-BUILD data.
<MSN>_PBcfg.c	This file contains: > local constant macros > local function macros > local data types and structures > local data prototypes > local data > global data of CONFIG-CLASS POST-BUILD data.
<MSN>_XMI21.xml	This file is a XMI file to visualize data relations e.g. in Enterprise Architect. The file is used for development purposes at Vector and informational for the customer.

Table 4-1 Generated files

4.2 IMPLEMENTATION-CONFIG-VARIANT dependent Data

The CONFIG-CLASS of generated data depends on the configured IMPLEMENTATION-CONFIG-VARIANT and the IMPLEMENTATION-CONFIG-CLASSES described in the <MSN>_bswmd.arxml.

Expert Knowledge
If the generated data is in a C struct and the struct contains pre-compile and postbuild changeable data, the data nature is postbuild.

IMPLEMENTATION-CONFIG-VARIANT	Description
VARIANT-PRE-COMPILE [SELECTABLE]	> All generated data is of CONFIG-CLASS PRE-COMPILE and generated into <MSN>_Cfg.c or <MSN>_Lcfg.c (if <MSN>_Cfg.c does not exist). > CONFIG-CLASS LINK and POST-BUILD data does not exist.
VARIANT-LINK-TIME	> CONFIG-CLASS PRE-COMPILE data and is generated into <MSN>_Cfg.c or <MSN>_Lcfg.c(if <MSN>_Cfg.c does not exist). > CONFIG-CLASS LINK data and is generated into <MSN>_Lcfg.c. > CONFIG-CLASS POST-BUILD data changeable data does not exist.
VARIANT-POST-BUILD-LOADABLE [SELECTABLE]	> CONFIG-CLASS PRE-COMPILE data and is generated into <MSN>_Cfg.c or <MSN>_Lcfg.c(if <MSN>_Cfg.c does not exist). > CONFIG-CLASS LINK data and is generated into <MSN>_Lcfg.c. > CONFIG-CLASS POST-BUILD data and is generated into <MSN>_PBcfg.c.

Table 4-2 IMPLEMENTATION-CONFIG-VARIATIONS

4.3 Optimization Levels

This chapter describes optimization levels and their configuration. Use Table 4-3 Optimization Levels and Table 4-4 Optimization Decision Table to tailor your configuration.

Optimization	Description
Small (Default)	The data is reduced by operations and not packed into a data stream.
Fast	The data is not reduced by operations and not packed into a data stream.
Tiny	The data is not reduced by operations and packed into a data stream.
Teeny-weeny	The data is reduced by operations and packed into a data stream.

Table 4-3 Optimization Levels

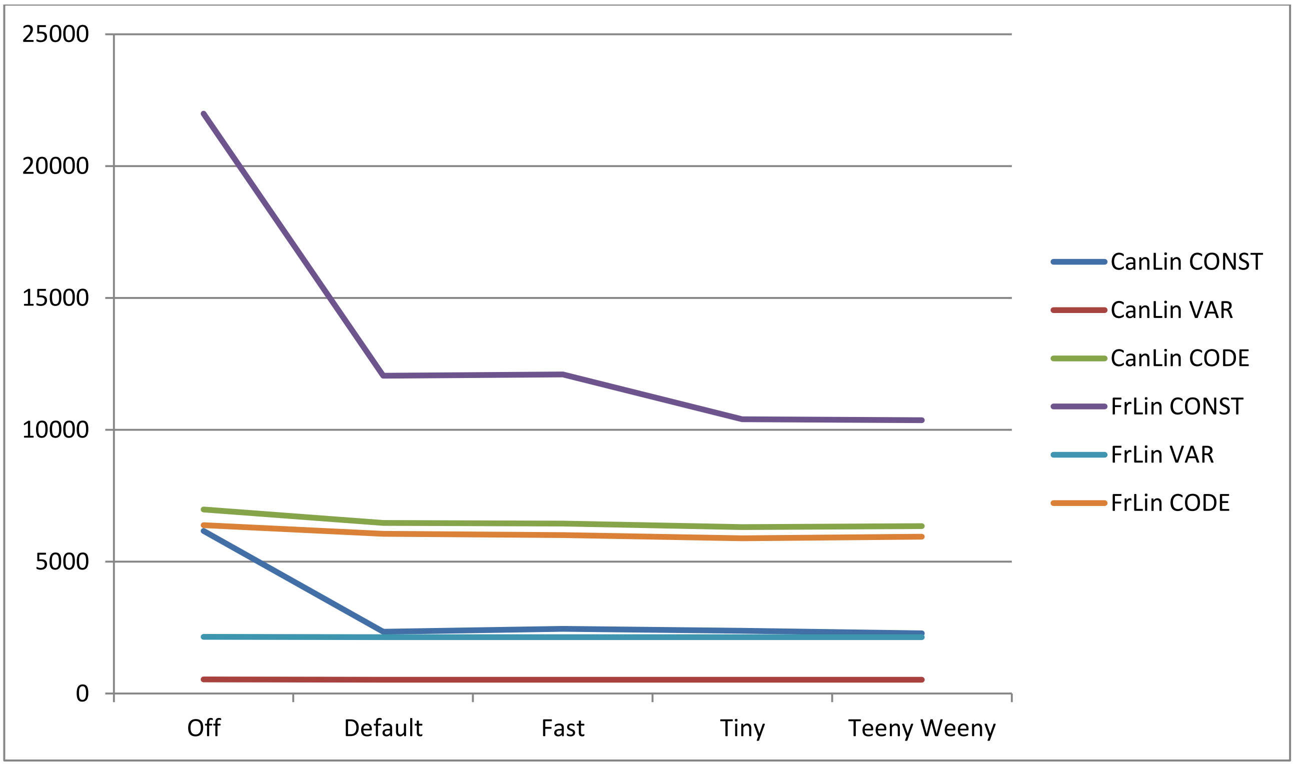

Figure 4-1 Resources in optimization variants

Parameter \ Optimization Level	Small (Default)	Fast	Tiny	Teeny-weeny
<MSN>MinimizeNumericalDataTypes	TRUE	TRUE	TRUE	TRUE
<MSN>ConstDataDeduplication	DEDUPLICATE_CONST_DATA_WITH_CAST	DEDUPLICATE_CONST_DATA_WITH_CAST	DEDUPLICATE_CONST_DATA_WITH_CAST	DEDUPLICATE_CONST_DATA_WITH_CAST
<MSN>OptimizeConstArrays2Define	TRUE	TRUE	TRUE	TRUE
<MSN>OptimizeConstVars2Define	TRUE	TRUE	TRUE	TRUE
<MSN>StructBoolDataUsage	BITMASKING	BOOLEAN	BOOLEAN	BITMASKING
<MSN>DeduplicateZero2NIndirectedData	TRUE	TRUE	TRUE	TRUE
<MSN>ReduceBoolDataByNegationThreshold	2	0	0	2
<MSN>ReduceNumericalDataByOffsetThreshold	2	0	0	2
<MSN>ReduceBoolDataByNumericalComparisonThreshold	2	0	0	2
<MSN>ReduceNumericalDataByArraySubtractionThreshold	2	0	0	2
<MSN>DeduplicateBoolDataByNumericalComparision	2	0	0	2
<MSN>UseSignedDataTypesInIndexArrays	FALSE	FALSE	FALSE	FALSE
<MSN>ReduceDataByStreaming	FALSE	FALSE	TRUE	TRUE

Table 4-4 Optimization Decision Table

4.4 MISRA, PRQA and Compiler Warnings

The MICROSAR code is in the most cases a piece of hand written static code and generated data and code for different compilers. This combination of hand written and generated code can produce MISRA deviations or compiler warnings. This chapter extends [1].

4.4.1 General

Note
The ComStackLib switch <MSN>OptimizeConstArrays2Define may produce compiler warnings. If you don't trust your compiler or your project settings do not allow the usage of compiler warnings, configure <MSN>OptimizeConstArrays2Define to false.

Deviation ID	MD_CSL_3199
Violated rule	PRQA Redundancy 3199 (The value of '%s' is never used following this assignment.)
Reason	The parameter /MICROSAR/EcuC/EcucGeneral/DummyStatement is configured to TRUE to avoid the compiler warning about unused function parameters. If the function is an interface to other modules and the prototype is specified by a standard, the prototype cannot be changed. If the function is not defined by a standard, the parameter could be removed in the implementation. The disadvantage is that the code itself is stuffed with preprocessor statements and the number variations of the software are exploding. Due to this, the code will not be changed.
Potential risks	The function contains unused code.
Prevention of risks	Configure the parameter /MICROSAR/EcuC/EcucGeneral/DummyStatement to FALSE and accept the compiler warning about unused function parameters. OR The code inspection is in charge to detect unused code.

Examples	`#define MSN_PROCESS_DATA STD_OFF`
	`#define MSN_USE_DUMMY_STATEMENT STD_ON`
	
	`void Msn_foo(uint8 a)`
	`{`
	`#if (MSN_PROCESS_DATA == STD_ON)`
	`  /* some code which uses the parameter a */`
	`#endif`
	`#if (MSN_USE_DUMMY_STATEMENT == STD_ON)`
	`# if (MSN_PROCESS_DATA == STD_OFF)`
	`  a=a;`
	`# endif`
	`#endif`
	`}`

Table 4-5 MD_CSL_3199

Deviation ID	MD_CSL_750_759
Violated rule	Rule 18.4 (Unions shall not be used.)
Reason	Generated data uses array and symbol based data access. The embedded code itself uses only one access type. Due to this critical runtime effects do not occur.
Potential risks	The A2L data may not match to the real data.
Prevention of risks	Each delivery is integrated and tested on the real target system.
Examples	see below

```
/* symbolic data access for A2L */
typedef struct sMsn_FooDataStructType
{
  boolean indexA;
  boolean indexB;
} Msn_FooDataStructType;

/* union data type to have array and symbolic data
access */
typedef union uMsn_FooDataType
{
  boolean raw[2]; /**< this element is used for array
based data access from the embedded code */
  Msn_FooDataStructType str; /**< this element is
used for symbolic based data access from A2L */
} Msn_FooDataUType;

/* this variable array uses the union data type */
Msn_FooDataUType msn_FooData;
```

Table 4-6 MD_CSL_750_759

Deviation ID	MD_CSL_0779
Violated rule	Rule 5.1 (Identifiers (internal and external) shall not rely on the significance of more than 31 characters.)
Reason	Generated symbols may exceed the 31 character limitation, because the code generator concatenates strings based on fixed rules.
Potential risks	The linker or compiler may mismatch symbols.
Prevention of risks	Modern compilers for AUTOSAR platforms do not have this limitation any more.

Examples	`#if (MSN_DEFRXSIGGRPINFOENDIDXOFDEFRXPDUINFO == STD_ON)`
	`{`
	`  Msn_DefRxSigGrpInfoEndIdxOfDefRxPduInfoType idxRxSigGrpInfo =`
	`  Msn_GetDefRxSigGrpInfoStartIdxOfDefRxPduInfo(idxRxPduInfo);`
	`  /* some code */`
	
	`#endif`

Table 4-7 MD_CSL_0779

<table>
<tr><th>Deviation ID</th><th>MD_CSL_2018</th></tr>
<tr><td>Violated rule</td><td>Rule 14.1 (This switch default label is unreachable.)</td></tr>
<tr><td>Reason</td><td>The parameter <MSN>OptimizeConstArrays2Define is configured to TRUE.</td></tr>
<tr><td>Potential risks</td><td>The default case of the switch statement contains possibly dead code.</td></tr>
<tr><td>Prevention of risks</td><td>The code inspection is in charge to detect useless conditions with possibly dead code.</td></tr>
<tr><td>Examples</td><td><pre>
#define MSN_PROCESS_DATA   STD_ON
#define MSN_CASE_SMALL   5
#define MSN_CASE_MEDIUM   8
#define MSN_CASE_LARGE   12
#define MSN_CASE_SMALL_USED   FALSE
#define MSN_CASE_MEDIUM_USED   TRUE
#define MSN_CASE_LARGE   FALSE

/* this array is reduced to a constant define
const uint8 msn_FooData [2] =
{
  MSN_CASE_MEDIUM,
  MSN_CASE_MEDIUM
};
*/
#define Msn_GetFooData (Index)   MSN_CASE_MEDIUM

void Msn_foo(uint8 a)
{
#if (MSN_PROCESS_DATA == STD_ON)
  switch(Msn_GetFooData(a))
  {
#if (MSN_CASE_SMALL_USED == STD_ON)
    case MSN_CASE_SMALL:
      /* some MSN_CASE_SMALL code */
      break;
#endif
#if (MSN_CASE_MEDIUM_USED == STD_ON)
    case MSN_CASE_MEDIUM:
      /* some MSN_CASE_MEDIUM code */
      break;
#endif
#if (MSN_CASE_LARGE_USED == STD_ON)
    case MSN_CASE_LARGE:
      /* some MSN_CASE_LARGE code */
      break;
#endif
    default:
      /* some default handling like calling Det */
  }
#endif
}
</pre></td></tr>
</table>

Table 4-8 MD_CSL_2018

Deviation ID	MD_CSL_3355_3356_3358_3359
Violated rule	Rule 13.7 (The result of this logical operation or control expression is always 'false' or 'true')
Reason	The parameter <MSN>OptimizeConstArrays2Define is configured to TRUE.
Potential risks	The function contains useless conditions with possibly dead code.
Prevention of risks	The code inspection is in charge to detect useless conditions with possibly dead code.

Examples	`#define MSN_PROCESS_DATA          STD_ON`
	
	`/* this array is reduced to a define`
	`const boolean msn_FooData [2] =`
	`{`
	`  TRUE,`
	`  TRUE`
	`};`
	`*/`
	`#define Msn_IsFooData (Index)      TRUE`
	
	`void Msn_foo(uint8 a)`
	`{`
	`#if (MSN_PROCESS_DATA == STD_ON)`
	`  if(Msn_IsFooData(a))`
	`  {`
	`    /* some code */`
	`  }`
	`#endif`
	`}`

Table 4-9 MD_CSL_3355_3356_3358_3359

<table>
<tr><th>Deviation ID</th><th>MD_CSL_3453</th></tr>
<tr><td>Violated rule</td><td>Rule 19.7 (A function should be used in preference to a function-like macro.)</td></tr>
<tr><td>Reason</td><td>ComStackLib based modules use macros to access generated RAM and ROM data. The implementation of data access functions would cause much code and runtime.</td></tr>
<tr><td>Potential risks</td><td>Resulting code is difficult to understand or may not work as expected.</td></tr>
<tr><td>Prevention of risks</td><td>Code inspection and test of the different variants in the component test.</td></tr>
<tr><td>Examples</td><td><pre>
#define MSN_PROCESS_DATA   STD_ON

/* this array is accessed by a generated data access
macro */
const boolean msn_FooData [2] =
{
  TRUE,
  TRUE
};

#define Msn_IsFooData (Index)   msn_FooData[Index]

void Msn_foo(uint8 a)
{
#if (MSN_PROCESS_DATA == STD_ON)
  if(Msn_IsFooData(a))
  {
    /* some code */
  }
#endif
}
</pre></td></tr>
</table>

Table 4-10 MD_CSL_3453

Deviation ID	MD_CSL_310
Violated rule	Rule 11.4 (A cast should not be performed between a pointer to object type and a different pointer to object type.)
Reason	The parameter <MSN>OptimizeConstArrays2Define is configured to TRUE AND the module configuration variant is PRE-COMPILE or POST-BUILD-LOADABLE SELECTABLE. The values behind a symbol are reduced to a constant define, but a non NULL_PTR is needed to identify the usage of the values in the source code. Due to this the module root symbol is used.
Potential risks	The compiler and MISRA warns about the cast of different pointer types.
Prevention of risks	The code uses the generated macros to access data values and does not touch the pointers.

Examples	`#define Msn_GetFoo(Index)      1U`
	`#define Msn_HasFoo ()          (Msn_ConfigDataPtr->FooPtrOfPCConfig != NULL_PTR)`
	`#define Msn_Foo                ((Msn_FooPtrType)(&(Msn_PCConfig)))`
	
	`CONST(Msn_PCConfigsType, MSN_CONST) Msn_PCConfig = {`
	`  { /* Index: 0 Keys: [Config_LeftFront] */`
	`      Msn_Foo  /**< the pointer to Msn_Foo */  /* PRQA S 0310 */  /* MD_CSL_310 */`
	`    , 5U                    /**< the number of elements in Msn_Foo */`
	`  },`
	`  { /* Index: 1 Keys: [Config_RightFront] */`
	`      NULL_PTR              /**< the pointer to Msn_Foo */`
	`    , 0U                    /**< the number of elements in Msn_Foo */`
	`  }`
	`};`

Table 4-11 MD_CSL_0310

4.4.2 Bitfields

The data type of bit fields is configurable in the EcuC module and important if <MSN>StructBoolDataUsage is configured to BITFIELD. According to Table 4-12 /MICROSAR/EcuC/EcucGeneral/BitFieldDataType the usage of UNSIGNED_INT is the best choice, but for some compilers the usage of UNSIGNED_CHAR is for some reasons required and you want to live with the MISRA violations.

BitFieldDataType Literal	Description
INT	• does typically not produce a compiler warning • violates MISRA Rule 6.4 Bit fields shall only be defined to be of type unsigned int or signed int. MISRA Rule 6.5 Bit fields of type signed int shall be at least 2 bits long. MISRA Rule 10.1 The value of an expression of integer type shall not be implicitly converted to a different underlying type if: a) it is not a conversion to a wider integer type of the same signedness, or b) the expression is complex, or c) the expression is not constant and is a function argument, or d) the expression is not constant and is a return expression (if TRUE is assigned to the value as initializer)
UNSIGNED_INT	• does typically not produce a compiler warning • violates no MISRA Rule
UNSIGNED_CHAR	• does typically produce a compiler warning like warning C4214: nonstandard extension used : bit field types other than int • violates MISRA Rule 6.4 Bit fields shall only be defined to be of type unsigned int or signed int.
UNSIGNED_SHORT	• does typically produce a compiler warning like warning C4214: nonstandard extension used : bit field types other than int • violates MISRA Rule 6.4 Bit fields shall only be defined to be of type unsigned int or signed int.

Table 4-12 /MICROSAR/EcuC/EcucGeneral/BitFieldDataType

4.4.3 <MSN>_Has Macros in the SELECTABLE Use Case

The usage of <MSN>_Has* macros produces in the SELECTABLE use case compiler warnings like “The result of this logical operation is always 'false' or ‘true’”. This compiler warning is up to now acceptable because the compiler detects automatically the case where the “if” condition is not needed and removes automatically the runtime consuming if condition. A typical use case is described in the following example code.

Example

The generated CONST or VAR data element accesses by `Msn_GetFooData()` is needed in all predefined variants. Due to this, the generated `Msn_HasFooData()` macro is always true and the compiler warning occurs.

```
#define MSN_USE_INIT_POINTER          STD_ON

#define Msn_HasFooData()              TRUE

void Msn_foo(uint8 a)
{
#if (MSN_USE_INIT_POINTER == STD_ON)
  if(Msn_HasFooData())
#endif
  {
    /* some code and process Msn_GetFooData() */
  }
}
```

5 Configuration

ComStackLib based BSW generators can be configured according with CFG5. For a detailed description see 5.2.

5.1 Configuration Variants

The configuration classes of ComStackLib based BSW generators depend on the supported configuration variants. For their definitions please see the BSW specific <MSN>_bswmd.arxml file.

5.2 Configuration with a GCE

Note
The configuration parameters, their multiplicity and default values depend on the BSW module. For their definitions please see the BSW specific <MSN>_bswmd.arxml file.

Container Name	<MSN>General
Path	\MICROSAR\<MSN>\<MSN>General
Multiplicity	1..1
Description	The general configuration container of the ComStackLib based BSW configuration

Table 5-1 Container

Attribute Name	Value Type	Description
<MSN>OutOfBoundsWriteProtectionStrategy	ENUM	This parameter is used to configure a strategy to protect the code to write out of bounds. NONE: no protection strategy is generated in the data access. INDEX_SATURATION: arrays are blown up and the data access index is saturated by an appropriate mask. The advantage is the speed of the data access, but own data elements at other indexes of the same variable can be overridden. INDEX_CHECKING: the data access index is validated by a runtime check. The advantage is that values are never written to incorrect indexes of the data access.
<MSN>OutOfBoundsWriteSanitizer	BOOL	This parameter activates/deactivates the generation of runtime checks which call a DET error notification function to find easily out of bounds write problems.

Attribute Name	Value Type	Description
		The Feature must not be used in production code! FALSE: no checks are generated in the data access. TRUE: the data access is enriched with DET checks to validate indexes.
<MSN>OutOfBoundsReadSanitizer	BOOL	This parameter activates/deactivates the generation of runtime checks which call a DET error notification function to find easily out of bounds read problems. The Feature must not be used in production code! FALSE: no checks are generated in the data access. TRUE: the data access is enriched with DET checks to validate indexes.
<MSN>MinimizeNumericalDataTypes	ENUM	This parameter is used to minimize the datatypes of CONFIG-CLASS PRE-COMPILE numerical data based on the used values. NONE: The datatype is not minimized. MINIMIZE_NUMERICAL_DATA_TYPES_WITHOUT_CAST: The datatype of numerical data is minimized. Unsigned data types are not changed to signed datatypes. Code: the code size is reduced. RAM: the RAM size is reduced. ROM: the ROM size is reduced. Runtime: no change expected. MINIMIZE_NUMERICAL_DATA_TYPES_WITH_CAST: The datatype of numerical data is minimized. Unsigned data types can be optimized to signed datatypes. Code: the code size is reduced. RAM: the RAM size is reduced. ROM: the ROM size is reduced. Runtime: no change expected.
<MSN>ConstDataDeduplication	ENUM	This parameter is used to deduplicate CONFIG-CLASS PRE-COMPILE ROM data. NONE: The generated data is not deduplicated. DEDUPLICATE_CONST_DATA_WITHOUT_CAST: The data is deduplicated without using casts. Code: no change expected. RAM: no change expected. ROM: the ROM size can be minimized. Runtime: no change expected. DEDUPLICATE_CONST_DATA_WITH_CAST: The data is deduplicated

Attribute Name	Value Type	Description
		using casts. Code: no change expected. RAM: no change expected. ROM: the ROM size can be minimized more than in DEDUPLICATE_CONST_DATA_WITHOUT_CAST. Runtime: no change expected.
<MSN>OptimizeConstArrays2Define	BOOL	This parameter activates/deactivates the capability to generate CONFIG-CLASS PRE-COMPILE ROM arrays as constant define. TRUE: ROM arrays are generated as constant define if all values are identical. Code: the code size is smaller. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased. FALSE: ROM arrays are generated as data even if all values are identical.
<MSN>OptimizeConstVars2Define	BOOL	This parameter activates/deactivates the capability to generate CONFIG-CLASS PRE-COMPILE ROM constants as constant define. TRUE: ROM constants are generated as constant define. Code: the code size is smaller. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased. FALSE: ROM constants are always generated as data.
<MSN>StructBoolDataUsage	ENUM	This parameter is used to tailor the usage of boolean data in structures in all CONFIG-CLASSES. The difference between BITFIELD and BITMASKING depends on your compiler options and memory mapping. BOOLEAN: The datatype of boolean data is native boolean. Code: the code size is small. RAM: no change expected. ROM: the ROM size is large. Runtime: the runtime is fast. BITFIELD: The bitfield type is used and the compiler extracts the boolean data from structures. Code: the code size is larger than using BOOLEAN. RAM: no change expected. ROM: the ROM size is smaller than using BOOLEAN.

Attribute Name	Value Type	Description
		Runtime: the runtime is larger than using BOOLEAN. BITMASKING: Generated Masks are used to extract the boolean data from structures. Code: the code size is larger than using BOOLEAN. RAM: no change expected. ROM: the ROM size is smaller than using BOOLEAN. Runtime: the runtime is larger than using BOOLEAN.
<MSN>DeduplicateZero2NIndirectedData	BOOL	This parameter activates/deactivates the capability to compress 0:N relational ROM data in all CONFIG-CLASSES without increasing the runtime. This option can be used in lib builds and in postbuild configurations. TRUE: 0:N relational ROM data is compressed without decreasing the runtime. Code: no change expected. RAM: no change expected. ROM: the ROM size is minimized. Runtime: no change expected. FALSE: 0:N relational ROM data is not compressed.
<MSN>ReduceBoolDataByNegationThreshold	INT	This parameter activates/deactivates the capability to compress boolean CONFIG-CLASS PRE-COMPILE ROM data by using the negation operator. 0: The optimization is not performed. >0: This is the threshold to activate the data optimization. Code: the code size is increased due to the usage of the negation operator in the data access. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased due to the usage of the negation operator in the data access.
<MSN>ReduceNumericalDataByOffsetThreshold	INT	This parameter activates/deactivates the capability to compress numerical CONFIG-CLASS PRE-COMPILE ROM data by using a constant offset. 0: The optimization is not performed. >0: This is the threshold to activate the data optimization. Code: the code size is increased due to the usage of the constant offset operation in the data access. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased due to the usage of the constant

Attribute Name	Value Type	Description
		offset operation in the data access.
<MSN>ReduceBoolDataByNumericalComparisonThreshold	INT	This parameter activates/deactivates the capability to compress boolean CONFIG-CLASS PRE-COMPILE ROM data by using comparison with other ROM data. 0: The optimization is not performed. >0: This is the threshold to activate the data optimization. Code: the code size is increased due to the usage of the operation in the data access. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased due to the usage of the operation in the data access.
<MSN>ReduceBoolDataByNumericalRelationThreshold	INT	This parameter activates/deactivates the capability to compress boolean CONFIG-CLASS PRE-COMPILE ROM data by using relational comparison with other ROM data. 0: The optimization is not performed. >0: This is the threshold to activate the data optimization. Code: the code size is increased due to the usage of the operation in the data access. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased due to the usage of the operation in the data access.
<MSN>ReduceNumericalDataByArraySubtractionThreshold	INT	This parameter activates/deactivates the capability to compress numerical CONFIG-CLASS PRE-COMPILE ROM data by using a subtraction with other ROM data. 0: The optimization is not performed. >0: This is the threshold to activate the data optimization. Code: the code size is increased due to the usage of the operation in the data access. RAM: no change expected. ROM: the ROM size is minimized. Runtime: the runtime is increased due to the usage of the operation in the data access.
<MSN>DeduplicateBoolDataByNumericalComparision	ENUM	This parameter is used to tailor the CONFIG-CLASS PRE-COMPILE ROM data deduplication mechanisms. A comparison with 0 is very efficient, but a numerical comparison with a value not 0 can be used to increase the ROM data compression rate. NONE: ROM data deduplications are switched off. Code: the code size is small. RAM: no change expected.

Attribute Name	Value Type	Description
		ROM: the ROM size is large. Runtime: the runtime is fast. DEDUPLICATE_DATA_WITH_ZERO: ROM data deduplications can be applied with the value 0. Code: the code size is larger than using NONE RAM: no change expected. ROM: the ROM size is smaller than using NONE. Runtime: the runtime is larger than using NONE. DEDUPLICATE_DATA_WITH_ANY_VALUE: ROM data deduplications can be applied with any numerical value. Code: the code size is larger than using NONE RAM: no change expected. ROM: the ROM size is smaller than using DEDUPLICATE_DATA_WITH_ZERO. Runtime: the runtime is larger than using NONE.
<MSN>ReduceDataByStreaming	BOOL	This parameter activates/deactivates the capability to pack generated CONFIG-CLASS PRE-COMPILE ROM data into a data type dependent stream. TRUE: generated const data is packed into a data type dependent stream. Code: no change expected. RAM: no change expected. ROM: configuration dependent smaller than with FALSE. Runtime: no change expected. FALSE: generated const data is not packed into a data type dependent stream.
<MSN>ShortSymbols	BOOL	This parameter activates/deactivates the capability to generate shortened symbol names. FALSE: symbol names are generated in a human readable style based on the MIP, tags and variant names. TRUE: symbol names are generated based on the MIP and a CRC32.
<MSN>InterfacesForDeactivatedData	BOOL	This parameter activates/deactivates the capability to generate bsw data interfaces for deactivated data elements. This is an advantage for the BSW developer to reduce the time to market with a development environment using auto completition and to investigate potential interfaces. FALSE: data interfaces are not generated if the data elementis deactivated. TRUE: data interfaces are generated as e.g. emty macros.

Attribute Name	Value Type	Description
<MSN>ReferringKeysInComments	BOOL	This parameter activates/deactivates the capability to generate referring keys in comments. This is an advantage for the developer to investigate indirections, but this feature reduces the overall readability of the generated data. FALSE: referring keys are not generated in comments. TRUE: referring keys are generated in comments.

Table 5-2 Attributes of ComStackLib based BSW generators

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BSWMD	The BSWMD is a formal notation of all information belonging to a certain BSW artifact (BSW module or BSW cluster) in addition to the implementation of that artifact.
CFG5	Generation tool for MICROSAR components.
Electronic Control Unit	Also known as ECU. Small embedded computer system consisting of at least one CPU and corresponding periphery which is placed in one housing.
Post-build	This type of configuration is possible after building the software module or the ECU software. The software may either receive parameters of its configuration during the download of the complete ECU software resulting from the linkage of the code, or it may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU software modules. In order to make the post-build time reconfiguration possible, the reconfigurable parameters shall be stored at a known memory location of ECU storage area.
Use case	A model of the usage by the user of a system in order to realize a single functional feature of the system.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CPU	Central Processing Unit
DET	Development Error Tracer
ECU	Electronic Control Unit
GCE	Generic Configuration Editor
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MIP	Module Implementation Prefix
MISRA	Motor Industry Software Reliability Association
RAM	Random Access Memory
ROM	Read-Only Memory
SWS	Software Specification
XMI	The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML).

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com