

# MICROSAR OS SafeContext

## Safety Manual

Renesas RH850 with compiler Green Hills

Authors	Senol Cendere, Yohan Humbert, Michael Kock
Version	1.10
Status	Released

## Document Information

### History

Author	Date	Version	Remarks
Senol Cendere	2014-02-17	1.00	Creation for RH850
Senol Cendere	2014-02-26	1.01	Updated the Requirement IDs
Senol Cendere	2014-05-09	1.02	Adaption for RH850 P1M
Senol Cendere	2014-08-18	1.03	Reworked after Safety Manual Review
Senol Cendere	2014-09-22	1.04	Added reference for Renesas Electronics RH850/P1M Safety Application Note Removed CPU derivative specification Removed compiler options (both are specified in safety case)
Yohan Humbert	2014-12-03	1.05	Added level support
Michael Kock	2015-08-18	1.06	Updated chapter Configuration Block
Senol Cendere	2016-01-05	1.07	Updated hardware specific part
Senol Cendere	2016-01-29	1.08	Rework after review
Senol Cendere	2016-02-09	1.09	Rework after review
Michael Kock	2016-06-17	1.10	Updated chapter Review Generated Code and Rework

## Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR Operating System Specification This document is available in PDF-format on the Internet at the AUTOSAR homepage: <a href="http://www.autosar.org">http://www.autosar.org</a>	3.x 4.x
[2]	OSEK	OSEK/VDX Operating System Specification This document is available in PDF-format on the Internet at the OSEK/VDX homepage: <a href="http://www.osek-vdx.org">http://www.osek-vdx.org</a>	2.2.3
[3]	Vector Informatik GmbH	MICROSAR OS SafeContext Technical Reference TechnicalReference_Os.pdf	9.01
[4]	Vector Informatik GmbH	MICROSAR Safe Silence Verifier Technical Reference TechnicalReference_MSSV.pdf	1.04
[5]	Vector Informatik GmbH	MICROSAR OS RH850 User Manual TechnicalReference_MICROSAROS_RH850.pdf	1.10
[6]	Vector Informatik GmbH	Vector MICROSAR OS SafeContext Concept	1.04
[7]	ISO	International Organization for Standardization, Draft International Standard ISO/DIS 26262 Road Vehicles - Functional Safety (all parts), 2009	2009
[8]	Renesas Electronics	V850E3v5 Architecture Specifications	(5 <sup>th</sup> edition)
[9]	Renesas Electronics	RH850G3K User's Manual: Software r01us0125ej0100_rh850g3k.pdf	Rev. 1.00 Aug, 2014
[10]	Renesas Electronics	RH850G3M User's Manual: Software r01us0123ej0100_rh850g3m.pdf	Rev. 1.00 Aug, 2014
[11]	Green Hills Software	MULTI: Building Applications for Embedded V850 and RH850 build_v800.pdf	PubID: build_v800-496213 Date: October 4, 2013

## Contents

<b>1</b>	<b>Purpose .....</b>	<b>8</b>
1.1	Safety Element out of Context (SEooC) .....	8
1.2	Standards and Legal Requirements .....	8
<b>2</b>	<b>Concept .....</b>	<b>9</b>
2.1	SafeContext Is One Part of a Whole .....	9
2.2	Safety Goal .....	9
2.3	Safety Requirements .....	9
2.4	SafeContext Functionality .....	10
2.4.1	ASIL Functionality .....	10
2.4.2	Detailed List .....	12
2.4.2.1	Provided Functionality .....	12
2.4.2.1.1	osGetConfigBlockVersion .....	14
2.4.2.2	Not provided Functionality .....	14
2.5	Safe State .....	15
<b>3</b>	<b>Overview of Requirements to the OS User .....</b>	<b>16</b>
<b>4</b>	<b>SafeContext Assumptions .....</b>	<b>18</b>
4.1	Context Definition .....	20
<b>5</b>	<b>OS Source Checksum .....</b>	<b>21</b>
<b>6</b>	<b>Patching the Configuration Block .....</b>	<b>23</b>
6.1	Using ElfConverter .....	24
6.2	Using ConfigBlockCRCPatch .....	24
<b>7</b>	<b>SafeContext Guidelines .....</b>	<b>25</b>
7.1	Configuration .....	25
7.2	Linking Example for Memory Mapping .....	27
<b>8</b>	<b>Configuration Block Review .....</b>	<b>28</b>
8.1	How to Read Back the Configuration .....	28
8.1.1	Using ElfConverter .....	29
8.1.2	Using HexConverter .....	29
8.1.3	Using ConfigViewer .....	30
8.2	Configuration Block Head .....	31
8.3	General Information .....	32
8.4	Task Start Address .....	34

8.5	Task Pre-emptive Configuration.....	35
8.6	Task Trusted Configuration.....	36
8.7	Task Stack Addresses .....	37
8.8	Task to Application Mapping .....	38
8.9	Category 2 ISR Trusted Configuration .....	39
8.10	Category 2 ISR to Application Mapping .....	40
8.11	Application Trusted Configuration.....	41
8.12	Trusted Functions Configuration.....	42
8.13	Non-Trusted Functions Configuration .....	43
8.14	Category 2 ISR Start Addresses.....	44
8.15	Category 2 ISR Nesting Configuration.....	45
8.16	Process to Core Mapping .....	46
8.17	Alarms to Core Mapping.....	47
8.18	Resources to Core Mapping.....	48
8.19	Counters to Core Mapping .....	49
8.20	Schedule Tables to Core Mapping .....	50
8.21	Application to Core Mapping.....	51
8.22	Trusted Functions to Core Mapping.....	52
8.23	Non-Trusted Functions to Core Mapping .....	53
8.24	Core Control Block Address .....	54
8.25	Peripheral Regions Configuration.....	55
8.26	Spinlock Lock Method .....	56
8.27	Spinlock Config Type.....	56
8.28	Optimized Spinlock Variable Addresses.....	56
8.29	Category 2 ISR Stack Address .....	57
8.30	Category 2 ISR Interrupt Channel Index.....	57
8.31	Category 2 ISR Priority Level .....	58
8.32	Category 2 ISR to Core Mapping.....	60
8.33	Application MPU Configuration.....	61
8.34	MPU Configuration .....	62
8.35	Application MPU ASID Configuration.....	63
<b>9</b>	<b>Generated OS Code .....</b>	<b>64</b>
9.1	Using MICROSAR Safe Silence Verifier (MSSV).....	64
9.2	Manual Reviews .....	66
9.2.1	Review generated file tcb.h .....	66
9.2.2	Review of tcb.c.....	67
9.2.3	Review of tcbpost.h .....	69
9.2.4	Review of trustfct.c & trustfct.h .....	71
9.2.4.1	File trustfct.c.....	71
9.2.4.2	File trustfct.h.....	73

<b>10</b>	<b>Review User Software.....</b>	<b>75</b>
<b>11</b>	<b>Hardware Specific Part.....</b>	<b>79</b>
11.1	Interrupt Vector Table .....	82
11.1.1	Header Include Section .....	82
11.1.2	Core Exception Vector Table .....	83
11.1.3	EIINT Vector Table.....	84
11.1.4	CAT2 ISR Wrappers .....	85
11.1.5	End of file Intvect_c<CoreID>.c .....	85
11.2	Linker Memory Sections .....	86
11.3	Linker Include Files .....	88
11.3.1	Review File osdata.dld .....	88
11.3.2	Review File ossdata.dld.....	89
11.3.3	Review File osstacks.dld .....	90
11.3.4	Review File osrom.dld .....	91
11.3.5	Review File ostdata.dld .....	91
11.4	Stack Size Configuration .....	92
11.5	Stack Monitoring.....	93
11.6	Usage of MPU Regions .....	93
11.7	Usage of Peripheral Interrupt API .....	93
<b>12</b>	<b>Glossary and Abbreviations.....</b>	<b>94</b>
12.1	Glossary.....	94
12.2	Abbreviations .....	95
<b>13</b>	<b>Contact .....</b>	<b>96</b>

## Illustrations

Figure 2-1	Stored and active contexts .....	11
Figure 3-1	Strategy for safety configuration.....	17
Figure 7-1	Linking Example.....	27

## Tables

Table 2-1	MICROSAR OS SafeContext Functionality .....	14
Table 2-2	Functionality – Not provided .....	15
Table 4-1	General SafeContext Assumptions.....	19
Table 6-1	ElfConverter Parameters.....	24
Table 8-1	ElfConverter Parameters.....	29
Table 8-2	HexConverter parameters .....	29
Table 8-3	ConfigViewer Parameters .....	30
Table 12-1	Glossary.....	94
Table 12-2	Abbreviations .....	95

# 1 Purpose

## 1.1 Safety Element out of Context (SEooC)

MICROSAR OS SafeContext is a Safety Element out of Context (SEooC). It is developed based on assumptions on the intended functionality, use and context, including external interfaces. To have a complete safety case, the validity of these assumptions has to be checked in the context of the actual item after integration of the SEooC.

The application conditions for SEooC provide the assumptions made on the requirements (including safety requirements) that are placed on the SEooC by higher levels of design and also on the design external to the SEooC and the assumed safety requirements and assumptions related to the design of the SEooC.

Information given by this document helps to check whether the SEooC fulfills the item requirements, or whether a change to the SEooC is necessary in accordance with the requirements of ISO 26262.

## 1.2 Standards and Legal Requirements

Standards followed by the development of MICROSAR OS SafeContext:

- > ISO 26262<sup>1</sup>
- > OSEK OS<sup>2</sup>
- > AUTOSAR OS<sup>3</sup>

---

<sup>1</sup> International Standard ISO 26262 Road Vehicles - Functional Safety (all parts), 2011

<sup>2</sup> OSEK/VDX Operating System, v2.2.3

<sup>3</sup> AUTOSAR Specification of Operating System



## 2 Concept

This chapter provides a description of the assumed safety requirements and the main concept.

### 2.1 SafeContext Is One Part of a Whole

SafeContext is part of Vector SafeExecution. SafeExecution consists of SafeContext for prevention from corrupted data and SafeWatchdog for supervision of timing behavior. This document covers SafeContext only.

### 2.2 Safety Goal

The safety goal is to ensure context integrity for all safety critical parts. Whenever a safety critical code is executed, it is guaranteed that the code is executed with the correct context. After pre-emption or interruption, execution is resumed with the correct context. The integrity of the memory is ensured by usage of hardware (e.g. MPU) and software measures.

### 2.3 Safety Requirements

To achieve this safety goal, the following assumed safety requirements are provided by SafeContext:

**ASA\_OS\_1:** Non-trusted software must be prevented from overwriting data of safety relevant software.

The OS assures this mainly by programming of an MPU.

**ASA\_OS\_2:** A *runtime context* (Task, Hook, (Non-)Trusted-Function or ISR) must not be destroyed by a switch (to another runtime context).

The OS assures this mainly by correct storage and restauration of the register context.

**ASA\_OS\_3:** A runtime context shall be set up according to compiler and processor specifications.

The OS assures this mainly by correct implementation of the register setup at context switches.

**ASA\_OS\_4:** Services to prevent data inconsistencies by racing conditions shall be provided.

The OS provides the following functions for this purpose:

- DisableAllInterrupts/EnableAllInterrupts,
- SuspendOsInterrupts/ResumeOSInterrupts
- SuspendAllInterrupts/ResumeAllInterrupts

**ASA\_OS\_5:** The OS never writes to unintended memory locations.

The OS assures this mainly by correct implementation of its functionality.

## 2.4 SafeContext Functionality

MICROSAR OS SafeContext implements the AUTOSAR OS and OSEK OS standards of a real-time operating system with some restrictions.

### 2.4.1 ASIL Functionality

Derived from our safety requirements, SafeContext provides the following functionality safely:

1. Context management
2. MPU management
3. Interrupt API
4. no unintended overwriting of memory

The *context* is defined as:

- > The set of registers, which is used by the compiler
- > The stack pointer
- > CPU mode (including interrupt state and privilege mode)
- > Memory access rights

Explanations:

- > A context switch occurs in several situations. These are: Task start/preemption/stop, ISR entry/exit, call of OS services, call of (Non-)Trusted Functions and Hook routines.

Conclusions:

- > If a user program is executed, it will always be executed with the correct context
- > After interruptions it is guaranteed that execution is resumed with the correct context
- > Freedom from interference with respect to memory will be achieved by using memory protection hardware (e.g. MPU) for non-trusted code.
- > Data inconsistency due to race conditions can be prevented by using the interrupt API.



#### Note

All other OS functionality, for example the sequence of task executions (scheduling) including the Task pre-emption is provided on QM level.

The operating system provides safe switching of memory access rights during context switches to ensure that non-trusted code does not modify data of other OS-Applications (if not explicitly allowed). In addition the OS interrupts Tasks or ISRs to execute higher priority ISRs. By switching to another context the correct context is set up. By switching back to an interrupted Task or ISR, the correct and unchanged context is restored. To avoid change of a saved context of an interrupted or waiting task, memory protecting hardware is used.

All points in the OS where context switches are performed or are necessary to perform are identified and developed according the safety standard.

At each point in time only one context is active. All other contexts are saved and protected by hardware against accidental alterations.

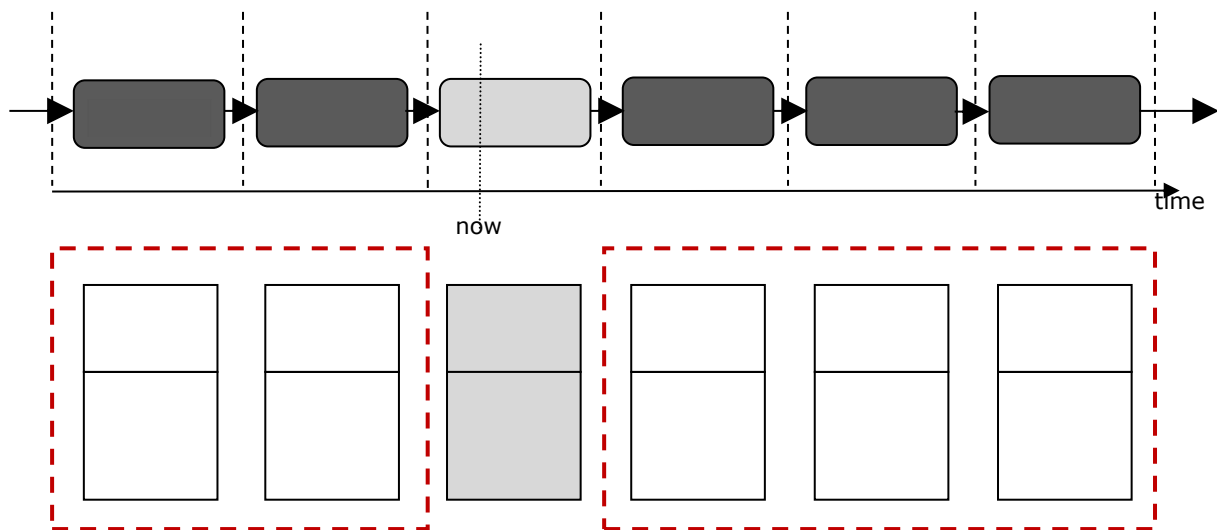


Figure 2-1 Stored and active contexts

## 2.4.2 Detailed List

### 2.4.2.1 Provided Functionality

The following OS services are provided by MICROSAR OS SafeContext.

Class	Description
Startup API	> > >
Shutdown API	>
Application API	> > > > > > > > > (if ASR4)
Interrupt API	> > > > > >
Stack usage API	> > >
Trusted Function API	>
Schedule Table API	> > > > >
Event API	> > >

Class	Description
	>
Alarm API	> > > >
Resource API	> >
Task API	> > > > >
Counter API	> > (ASR4)/ (ASR3) >
SafeContext Extensions	> > > > > > > > > > > >
OS System Hooks	> > > >
Error Handling	> > >

Class	Description
Timer handling	The handling of timer hardware is realized as QM software.
Scheduling	The correct sequence of processing application programs is realized with QM-Software (priority handling, including Resources).
ORTI	ORTI is realized as QM software.

Table 2-1 MICROSAR OS SafeContext Functionality

#### 2.4.2.1.1 osGetConfigBlockVersion

Description of the osGetConfigBlockVersion-API. [SPMF92:0045]

Prototype	
Return code	
	The version number which was specified by the user in the configuration attribute
Functional Description	
Return user configuration version.	
Particularities and Limitations	
> -	
Call context	
> any	

#### 2.4.2.2 Not provided Functionality

The features listed in the following table are not supported in SafeContext per default.

Class	Description
OS service API	<div> <div>&gt;</div> <div>&gt;</div> <div>&gt;</div> <div>&gt;</div> <div>&gt;</div> <div>&gt;</div> </div>
COM	OSEK COM inter task communication with messages is not supported
Interrupt resources	Resources are only available at task level.
Protection Reaction	The only allowed protection reaction in the is . Other reactions will be interpreted as . [SPMF92:0020]

Class	Description
Killing	Terminating Tasks or Applications is not supported.
OS Hooks	<p>&gt; (only for debugging!)</p> <p>&gt; (only for debugging!)</p> <p>&gt;</p> <p>&gt;</p>
OS Application specific Hooks	<p>&gt;</p> <p>&gt;</p> <p>&gt;</p>
Address Parameter Check	In case API functions with output-parameters are called with illegal address, they do not return with the error code E_OS_ILLEGAL_ADDRESS as required by the AUTOSAR specification. Instead the out parameter is written with the access rights of the caller, which may lead to a memory protection violation in case the given pointer is invalid.
Stack optimization	Single stack model and stack sharing are not supported.
Internal Resources	Internal Resources are not supported.
Configuration Aspects	<p>The following hooks must always be enabled:</p> <p>&gt;</p> <p>&gt;</p> <p>&gt;</p> <p>&gt;</p> <p>Only or is supported. Memory protection must be active.</p> <p>must be enabled.</p> <p>must be configured to .</p> <p>is not supported.</p> <p>is not supported.</p>

Table 2-2 Functionality – Not provided

## 2.5 Safe State

The safe state in SafeContext is shutdown (endless loop with interrupts disabled). The safe state is entered whenever the OS detects a violation of its safety goal or even an attempt. Before the safe state is entered, the is called. The may contain user code which is necessary to reach the defined safe state of the system. This might lead to a reset in combination with a watchdog.

### 3 Overview of Requirements to the OS User

For integration of the SafeContext into a particular context, the user has the following requirements to be fulfilled. They can be seen as steps to integrate the SEooC in the ECU without harming the assumed safety goal.

The top level requirements are listed in the following table. They are considered in more detail later. If all sub-requirements are checked, you can check the according top level requirement too.

Description of requirements to the OS user	Fulfilled
Check that all assumptions made by SafeContext are valid (see chapter "SafeContext Assumptions")	
Check code integrity of the used OS sources (see chapter "OS Source Checksum")	
Add CRC into the configuration block after linkage (see chapter "Patching the Configuration Block")	
Follow SafeContext guidelines (see chapter "SafeContext Guidelines")	
Review the safety relevant configuration data (see chapter "Configuration Block Review")	
Qualify the generated OS code (see chapter "Generated OS Code")	
Review your software (see chapter "Review User Software")	
Check specific requirements to the user (see chapter "Hardware Specific Part")	



#### Caution

All requirements listed in this document must be checked and fulfilled by the user!



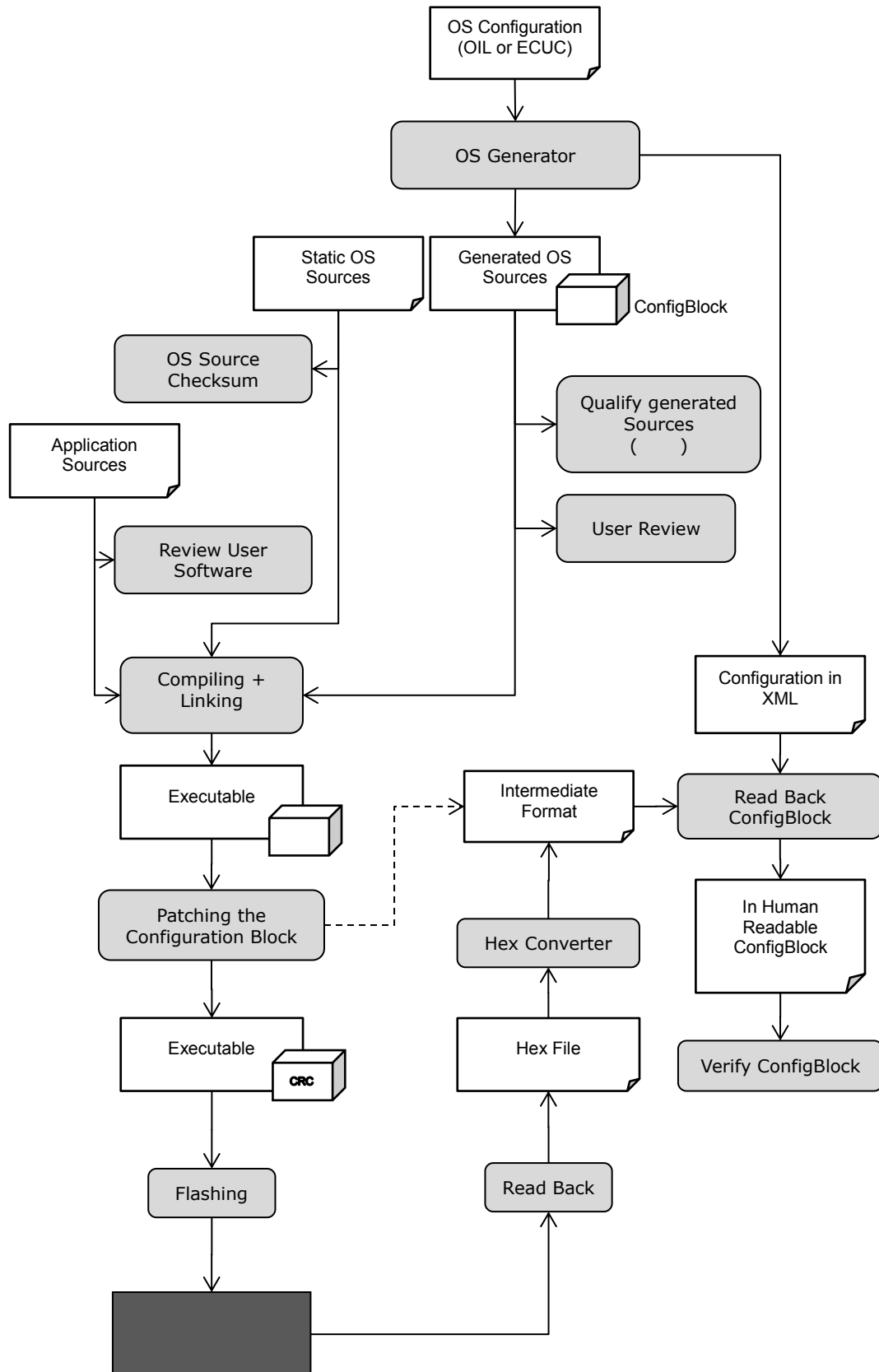


Figure 3-1 Strategy for safety configuration

## 4 SafeContext Assumptions

All assumptions must be checked to be true. Assumptions concerning the focus of SafeContext are given by the safety goals and related safety requirements described in the safety case. Assumptions about the environment are described in this chapter.

Description of requirements to the OS user	Fulfilled
<p><i>Know the SafeContext concept</i></p> <p>The system safety concept must not rely on OS functionality which is not part of the SafeContext safety requirements. Only the safety requirements (see chapter “Safety Requirements”) are assured by SafeContext.</p> <p>[SPMF92:0075]</p>	
<p><i>Timing and Scheduling</i></p> <p>If timing and scheduling is safety critical in your application, you have to supervise this by an external tool like e.g. SafeWatchdog.</p> <p>[SPMF92:0050]</p>	
<p><i>Know your memory configuration</i></p> <p>Setup of memory sections must be planned by the system designer. Whether or not the planned setup is configured correctly must be verified by reading the configuration back from the ECU and reviewing it against system design and hardware manuals.</p>	
<p><i>Know the OS specifications</i></p> <p>The user shall read the OS specifications for OSEK OS and AUTOSAR OS.</p>	
<p><i>Know how to use the OS</i></p> <p>The user shall read the OS manuals:</p> <ul style="list-style-type: none"> <li>&gt; General Technical Reference Manual</li> <li>&gt; Specific Technical Reference Manual</li> </ul> <p>Versions are listed in the delivered safety case.</p>	
<p><i>Use only the delivered OS generator</i></p> <p>The user shall only use the delivered OS generation tool for generating the user specific configuration.</p>	
<p><i>Correctness of processor</i></p> <p>The processor provides its functionality with sufficient safety, so that the OS needs not take care about potential hardware failure.</p> <p>This might be assured by usage of a lockstep processor.</p>	
<p><i>Correctness of memory</i></p> <p>The memory works with sufficient safety, so that the OS needs not to take care about potential hardware failure.</p>	
<p><i>Correctness of MPU</i></p> <p>The MPU provides its functionality with sufficient safety, so that the OS needs not take care about potential hardware failure.</p> <p>The OS provides an API ( ) which can be used by the user to check the MPU.</p>	
<p><i>Correctness of hardware manuals</i></p>	

Description of requirements to the OS user	Fulfilled
<p>The Hardware manuals and the compiler manuals are sufficiently reliable, so that the OS needs not take care about potential deviations between hardware functionality and its description in the manuals.</p> <p>Versions of the used hardware manuals are listed in the delivered safety case.</p> <p>[SPMF92:0017]</p>	
<p><i>Correctness of compiler tool chain</i></p> <p>SafeContext assumes that the compiler, assembler and linker generate code with the required safety level.</p>	
<p><i>Correctness of compiler version and options</i></p> <p>The used compiler version and options are identical to them which are used during development.</p> <p>Used compiler version and options are listed in the delivered safety case.</p>	
<p><i>Code integrity</i></p> <p>The source code and generated configuration of MICROSAR OS SafeContext is compiled, linked and downloaded to the ECU correctly and not modified afterwards.</p> <p>[SPMF92:0043]</p>	
<p><i>Context definition</i></p> <p>The user shall not rely on registers, which are not part of the context of the OS.</p> <p>The context definition is listed in chapter 4.1</p>	
<p><i>Hardware handled by the OS shall not be manipulated by user code</i></p> <p>User code shall not handle hardware which is handled by the OS. This may include:</p> <ul style="list-style-type: none"> <li>&gt; Interrupt Controller [SPMF92:0083]</li> <li>&gt; MPU [SPMF92:0085]</li> <li>&gt; Timer</li> </ul>	
<p><i>Don't manipulate short addressing base registers</i></p> <p>Do not manipulate registers which are used by the compiler for relative addressing of code or data. [SPMF92:0084]</p>	

Table 4-1 General SafeContext Assumptions

## 4.1 Context Definition

The context which is used by MICROSAR OS RH850 consists of the following registers:

FPU used	Registers	Size in Bytes	FPU not used	Registers	Size in Bytes
	R1	4		R1	4
	R2	4		R2	4
	R4	28 * 4 = 112		R4	28 * 4 = 112
	R5			R5	
	...			...	
	R30			R30	
	R31			R31	
total size = 152	EIPC	4	total size = 144		
	EIPSW	4			
	CTPC	4		EIPC	4
	CTPSW	4		EIPSW	4
	MPLA0	4		CTPC	4
	MPUA0	4		CTPSW	4
	FPSR	4		MPLA0	4
	FPEPC	4		MPUA0	4



### Caution

FPU setting in OS configuration must match the used compiler FPU option!

## 5 OS Source Checksum

The OS is delivered as source code. To assure that source code files are not altered after the testing and release a checksum is calculated. The user shall calculate the checksum to verify the correctness of the source code he is using. [SPMF92:0042]

A checksum calculation program (CCodeSafe.exe) is provided to the user. It is called with the following argument:

This tool calculates the CRC32 checksum over all files specified in file .

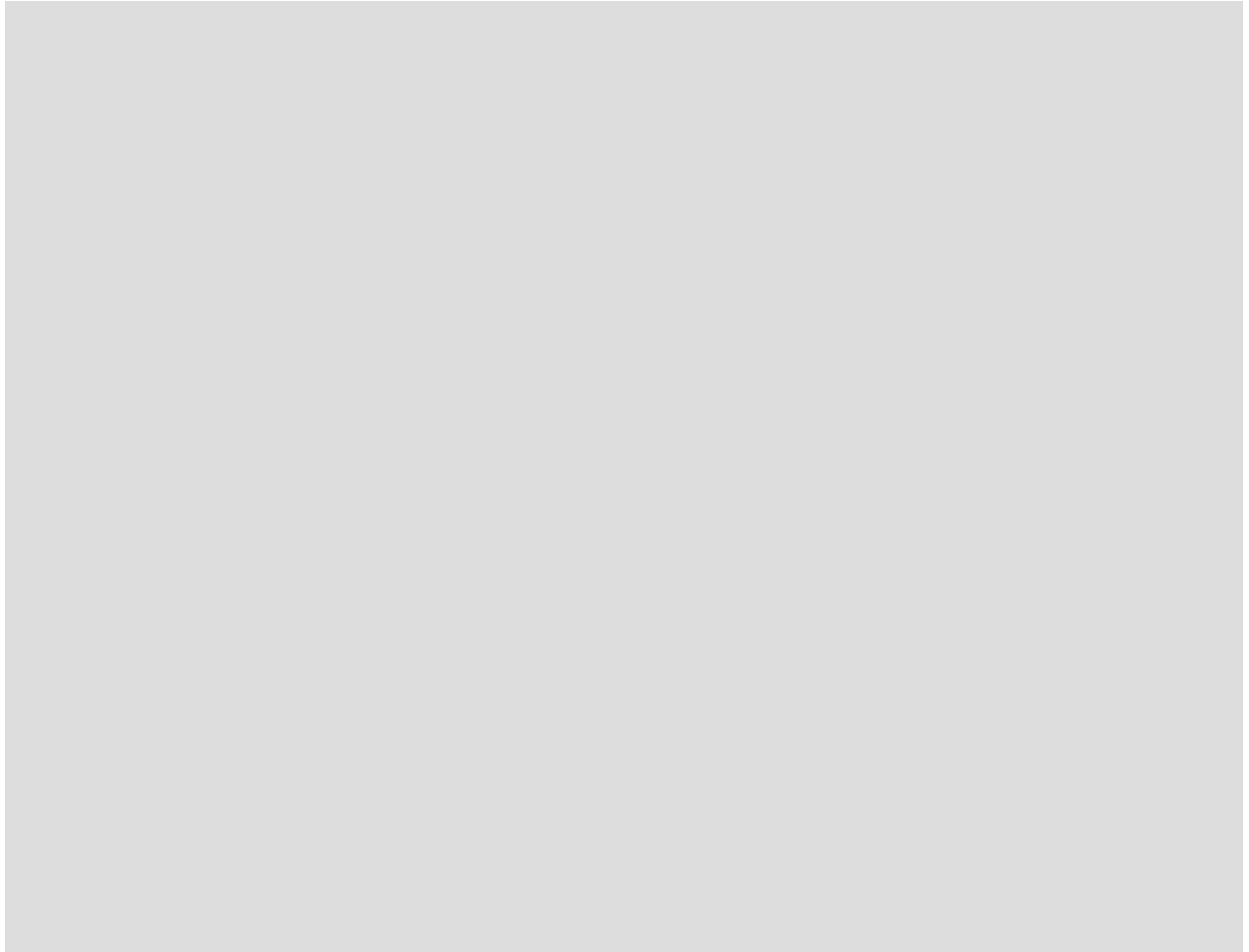
Description of requirements to the OS user	Fulfilled
Use the delivered source files from Vector! Do not use changed copies in a productive system! Also consider header include order of the compiler.	
The file                      shall contain the OS sources listed in the Safety Case.	
The calculated checksum returned by tool CCodeSafe.exe must be identical to the checksum given in the Safety Case.	



### Example

An example for a

file:



## 6 Patching the Configuration Block

Configuration information which is relevant to reach the safety goals is stored in a data structure called the configuration block. The integrity of this information is checked in using a CRC16 checksum.

The CRC must be calculated after compiling and linking the application. There are two programs provided to calculate and apply the CRC patch into the binary file:

	For patching the CRC into an ELF file. Creating an intermediate file for reading back the configuration block is also possible with this tool.
	For patching the CRC into an Intel HEX or Motorola SREC file.

The following steps are necessary to patch the configuration block:

1. Compile and link the complete application project to build the executable
2. Run CRC patch tool on executable
3. Write modified executable into ECU flash memory

Description of requirements to the OS user	Fulfilled
Check that CRC is non-zero. If so, change user configuration version to avoid zero CRC. [SPMF92:0071]	

## 6.1 Using ElfConverter

The program ElfConverter.exe is called with the following parameters:

```
ElfConverter.exe <Input-File> <Output-File> --crc_patch
```

Parameter	Description
<Input-File>	Input file with ELF-format which was built by the compiler tool chain
<Output-File>	Intermediate output file which is used by the tool ConfigViewer.exe for reading back the configuration block.
--crc_patch	Patch CRC checksum into ELF-file
--help	Show help

Table 6-1 ElfConverter Parameters



### Example

Patching the CRC checksum into configuration block of ELF-file testappl.out:

## 6.2 Using ConfigBlockCRCPatch

The tool ConfigBlockCRCPatch.exe is called with the following parameters:



### Example

Read HEX-file project.hex and create file project2.hex with patched CRC checksum:

Read SREC-file project.mot and create file project2.mot with patched CRC checksum:

The address of the configuration block must be taken via symbol `_osConfigBlock` from the linker generated map file. [SPMF92:0016]



## 7 SafeContext Guidelines

### 7.1 Configuration

Description of requirements to the OS user	Fulfilled
<p><i>Non-ASIL user code shall be part of Non-Trusted Applications</i></p> <p>All non-ASIL user code must be executed by Non-Trusted Applications with no write access to safety relevant data (including stacks) and no read or write access to safety relevant peripherals. [SPMF92:02.0034]</p>	
<p><i>ASIL user code shall not violate the SafeContext safety goals</i></p> <p>All user code, which has access to safety relevant data (including stacks, and OS data) or peripherals, must be implemented on ASIL level. This code shall never violate the safety goals of SafeContext. [SPMF92:0011]</p> <p>Code which typically has access to safety relevant data (depending on user configuration):</p> <ul style="list-style-type: none"> <li>&gt; Trusted Functions [SPMF92:0080] [SPMF92:03.0008]</li> <li>&gt; Trusted Tasks</li> <li>&gt; Trusted ISRs</li> <li>&gt; System Hooks <ul style="list-style-type: none"> <li>&gt; StartupHook [SPMF92:0040] [SPMF92:03.0007]</li> <li>&gt; ErrorHook [SPMF92:0012] [SPMF92:03.0006]</li> <li>&gt; ProtectionHook [SPMF92:0009] [SPMF92:03.0005]</li> <li>&gt; ShutdownHook [SPMF92:0013] [SPMF92:03.0009]</li> </ul> </li> <li>&gt; Reset Handler / Startup Code [SPMF92:0005] [SPMF92:03.0001]</li> <li>&gt; Exception Handlers [SPMF92:0087] [SPMF92:03.0010]</li> <li>&gt; Category 1 ISRs [SPMF92:0054] [SPMF92:03.0004] [SPMF92:03.0010]</li> </ul>	
<p><i>Alignment of data sections</i></p> <p>All data sections shall be linked with MPU alignment granularity (e.g. 32 bytes). See the controller's reference manual to know what the MPU granularity is. [SPMF92:0065] [SPMF92:04.0005]</p>	
<p><i>Consider category 1 ISRs</i></p> <p>Category 1 ISRs are completely transparent to the OS. The OS does not perform stack switching for category 1 ISRs! Consider this during configuration of stack sizes. [SPMF92:0086]</p>	
<p><i>NMIs shall be category 1 ISRs</i></p> <p>Non-maskable Interrupts (NMI) shall be configured to be category 1 ISRs. [SPMF92:0053] [SPMF92:03.0002]</p>	
<p><i>Link global safety data considering stack growing direction</i></p> <p>Link global safety data (all OS data and at least ASIL relevant application data) so that it cannot be corrupted by stack overflows (see Figure "Linking example" below for an example). [SPMF92:0091]</p>	
<p><i>FPU setting in OS configuration must match compiler FPU option.</i> [SPMF92:04.0019]</p>	

Description of requirements to the OS user	Fulfilled
<p>Check that <code>osdRH850_FPU</code> is only defined in file <code>tcb.h</code> and in <code>osek.h</code></p> <p>If compiler option <code>-fnone</code> or <code>-fsoft</code> is used, then assure that in OS configuration the attribute <code>SupportFPU</code> is set to <code>FALSE</code> and that in file <code>tcb.h</code> the following line is generated:</p> <p>If compiler option <code>-fsingle</code> or <code>-fhard</code> is used, then assure that in OS configuration the attribute <code>SupportFPU</code> is set to <code>TRUE</code> and that in file <code>tcb.h</code> the following line is generated:</p>	

## 7.2 Linking Example for Memory Mapping

The memory mapping should look like following example:

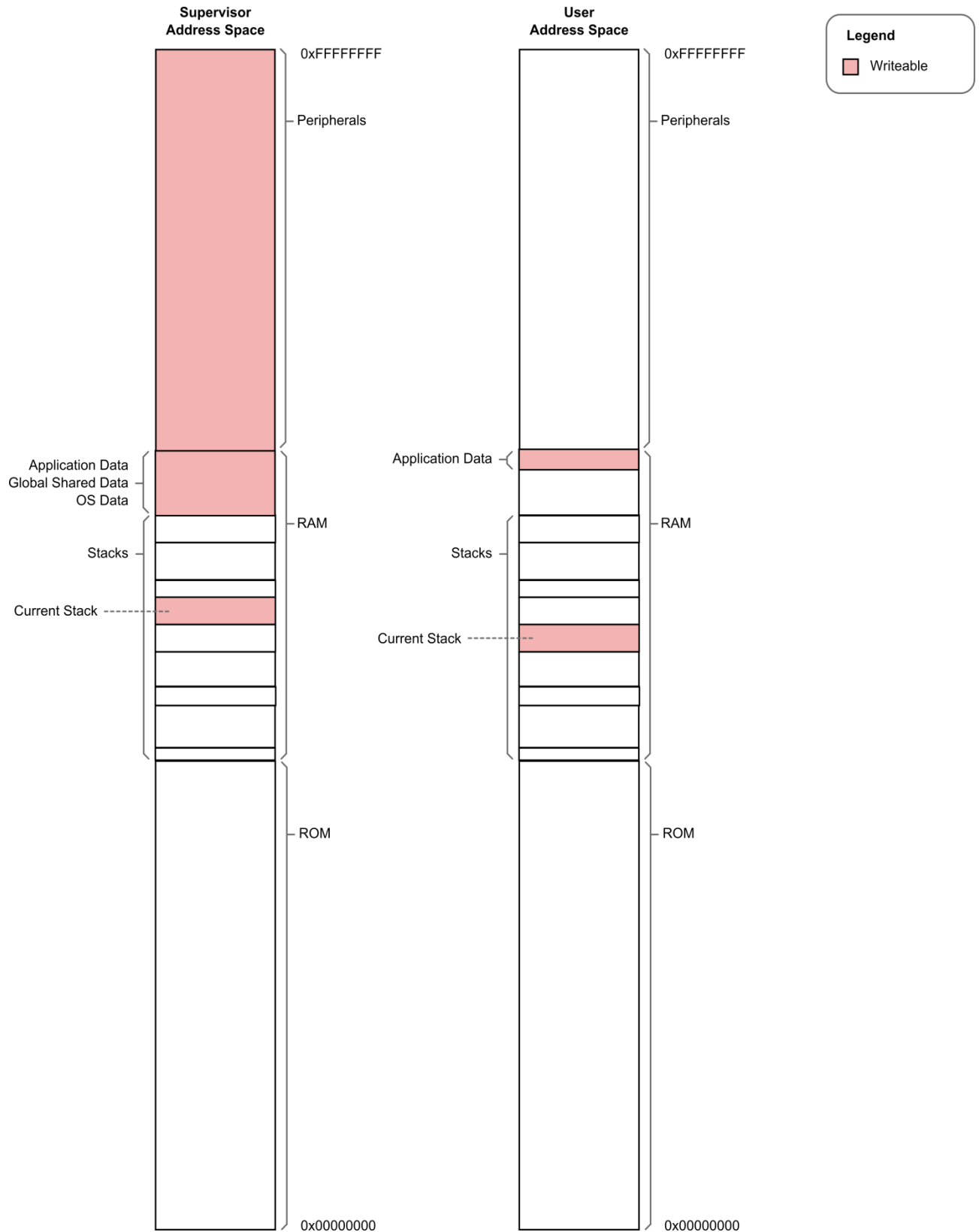


Figure 7-1 Linking Example

## 8 Configuration Block Review

The configuration of MICROSAR OS SafeContext is generated into C-code. The generator itself has not been developed in accordance to ASIL. Therefore, the generated configuration information needs to be reviewed. The safety relevant configuration is generated into a structure called configuration block (or ConfigBlock). This chapter describes how to review this ConfigBlock. As the ConfigBlock is simply a constant data structure in the flash memory of an ECU, humans will have difficulties to read it. Therefore, the configuration viewer is able to transform the ECU internal representation into a human readable format. The process of reading the ConfigBlock and transforming it into the human readable format is described in the following subchapter. [SPMF92:0038]

The setup of the memory protecting hardware depends on the correct configuration of the OS. All configuration parameters, which are necessary to ensure the safety goal, are stored in a contiguous memory block (configuration block). The configuration block can be located to a fix address and can be read back from the ECU, e.g. by XCP or a debug interface. [SPMF92:0034]

The configuration block is secured by a CRC16 checksum. The way how the configuration block is read back does not need to be safe. The configuration block is translated into a human readable format to allow a review against the intended configuration.

### 8.1 How to Read Back the Configuration

The configuration is read back in two steps:

1. Patch CRC checksum into executable file and create the intermediate file:
  - a) use ElfConverter.exe for CRC patch and intermediate file creation
  - b) or use ConfigBlockCRCPatch.exe for CRC patch and use HexConverter.exe for intermediate file creation
2. Extract the configuration block from intermediate file into human readable output by using the tool ConfigViewer.exe

The configuration block format is platform dependent. Also this information may be retrieved in different ways, e.g. as the HEX output of the linker or as an upload from the ECU via a protocol like XCP. As this may result in various file formats a conversion into an intermediate format is required.

### 8.1.1 Using ElfConverter

The tool ElfConverter.exe is able to patch CRC into ELF-file and create the intermediate file for reading back the configuration block. It is called with the following parameters:

```
ElfConverter.exe <Input-File> <Output-File> --crc_patch
```

Parameter	Description
<Input-File>	Input file with ELF-format which was built by the compiler tool chain
<Output-File>	Intermediate output file which is used by the tool ConfigViewer.exe for reading back the configuration block.
--crc_patch	Patch CRC checksum into ELF-file
--help	Show help

Table 8-1 ElfConverter Parameters

### 8.1.2 Using HexConverter

The tool HexConverter.exe creates the intermediate file for reading back the configuration block. It must be used after the CRC is patched via tool ConfigBlockCRCPatch.exe. Tool HexConverter.exe is called with the following parameters:

```
HexConverter.exe -i <Input-File> -o <Output-File> -b <Address> -s <Size>
```

All parameters are mandatory.

Parameter	Value	Description
-i	Input File Path and Name	File which was created by ConfigBlockCRCPatch.exe
-o	Output File Path and Name	Intermediate output file which is used by the tool ConfigViewer.exe for reading back the configuration block.
-b	Base Address of data structure _osConfigBlock (see linker map file)	Base address of the configuration block as defined in the linker map file. It is the address of the symbol 'osConfigBlock'. Value has to be given as hexadecimal (e.g. 0x008000).
-s	0xFFFF	This value must be at least the size of the configuration block in byte. Bigger values are also allowed. E.g. this value can be set to 0xFFFF

Table 8-2 HexConverter parameters

### 8.1.3 Using ConfigViewer

The tool ConfigViewer.exe reads the intermediate file and creates a human readable file. The intermediate input file must be generated via tool ElfConverter.exe or HexConverter.exe. ConfigViewer.exe is called with the following parameters:

ConfigViewer -i <Input-File> -o <Output-File> -x <XML-File>		
Parameter	Value	Description
-i	Input File Path & Name	Intermedia input file which was created by ElfConverter.exe or HexConverter.exe
-o	Output File Path & Name	Human readable configuration output file
-x	XML-ConfigFile	optional: XML-file which was generated by the OS generator with additional description of the OS configuration

Table 8-3 ConfigViewer Parameters

ConfigViewer.exe expects the intermediate file format to contain nothing else than the configuration block with patched CRC checksum.

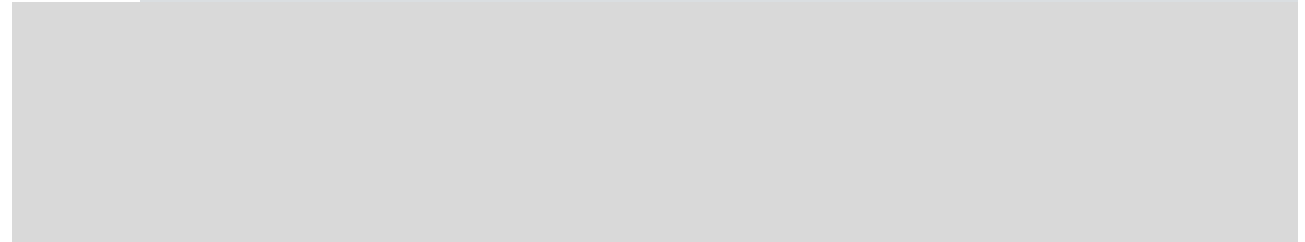
## 8.2 Configuration Block Head

The configuration block head contains information to identify the configuration block.



### Example

The configuration block head must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed start address, length and CRC is correct and matches the linker map file of symbol osConfigBlock.	
Check that the listed configuration block format is 03.00	
Check that the listed version of MICROSAR OS RH850 SafeContext matches the delivered OS version.	
If you are using the user configuration version, check that the listed one matches the configured value. [SPMF92:0045]	
The names printed by the ConfigViewer come from an unsafe source. For this reason check that object IDs and object Names have the same mapping in all configuration block sub-containers.	

### 8.3 General Information

The configuration viewer generates the general information as followed [SPMF92:0063].

**Example**

The container “General Information” must look like:

**Note**

To minimize variants in code, the OS generator introduces dummies for each OS object type. These dummies can be identified by the prefix \_\_\_\_\_ and are handled the same way like other OS objects. None of these objects are active at runtime.



Description of requirements to the OS user	Fulfilled
Check that the listed number of OS objects matches the OS configuration. (Consider that dummy OS objects are generated, to minimize variants in the OS code.)	
Check number of listed OS objects matches the elements in the following sub containers. [SPMF92:0074]	
Check the number of MPU regions which are provided by the used CPU derivative.	
Check the number of dynamic MPU regions which must be same as the number of MPU regions configured in the application with most dynamic MPU regions.	
Check the number of static MPU regions configured for the OS.	
Check the stack MPU values lower address matches the corresponding stack start address	
Compare the system stack start address in the configuration viewer output against the label <code>_osSystemStack_&lt;CoreID&gt;_StartAddr</code> in the linker map-file.	
Compare the system stack end address in the configuration viewer output against the label <code>_osSystemStack_&lt;CoreID&gt;_EndAddr</code> in the linker map-file.	
Check that both labels are 4 Byte aligned [SPMF92:04.0002]. This prevents that any data of other sections is accessible by the same MPU region.	
Check that only the array variable <code>osSystemStack&lt;CoreID&gt;</code> is located between the labels described above.	
Check the difference between the system stack start- and end-address against the designed (and therefore also configured) system stack size.	

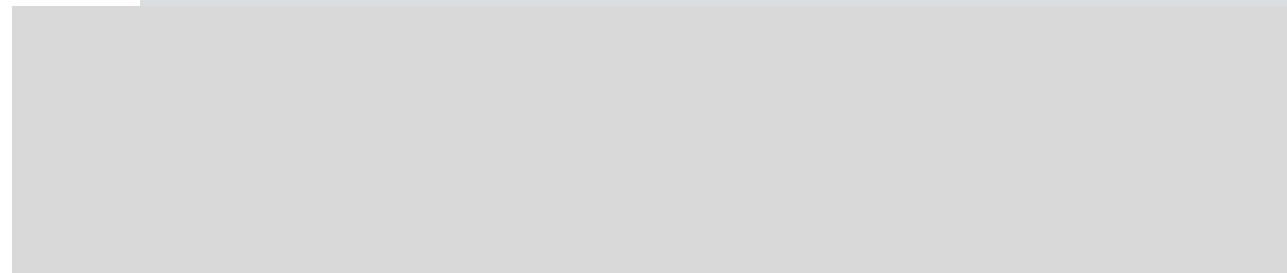
## 8.4 Task Start Address

Container “Task Start Address” contains the start addresses of all task functions.



### Example

The container “Task Start Address” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Task start addresses match to the function start address called <b>&lt;Task-Name&gt;func</b> (see linker map file). [SPMF92:02.0025]	

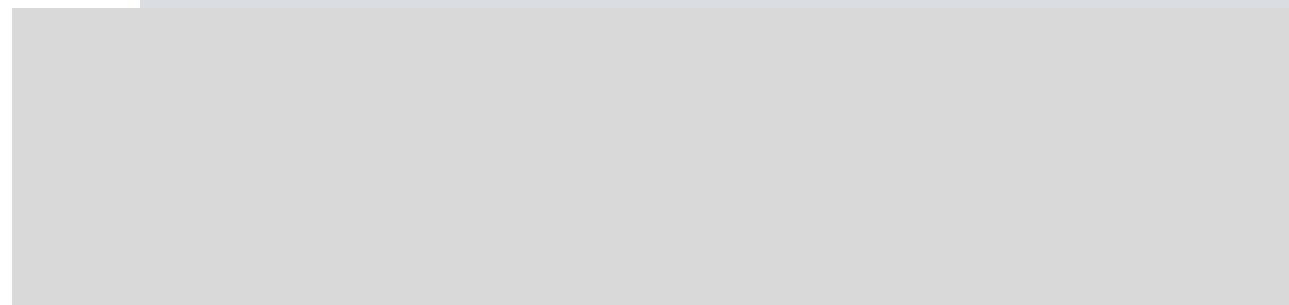
## 8.5 Task Pre-emptive Configuration

Container “Task Pre-emptive Attribute Setting” contains information about which task is full-pre-emptive or non-pre-emptive.



### Example

Container “Task Pre-emptive Attribute Setting” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed task pre-emptive attribute settings matches the OS configuration.	

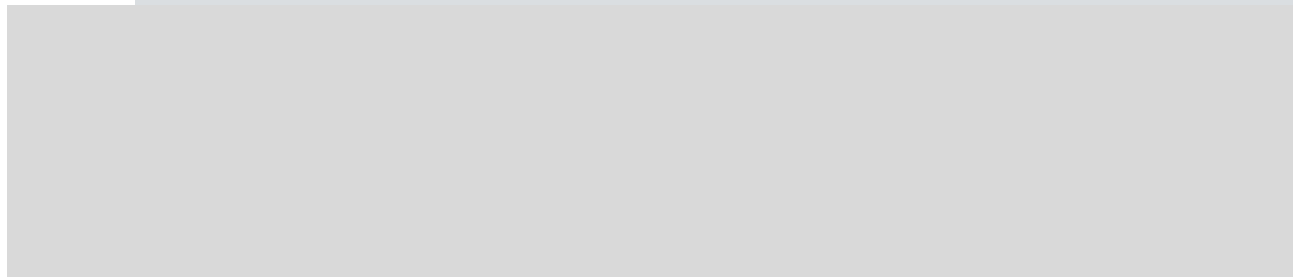
## 8.6 Task Trusted Configuration

Container "Task Trusted Attribute Setting" contains information about which task is trusted or non-trusted.



### Example

Container "Task Trusted Attribute Setting" must look like:



Description of requirements to the OS user	Fulfilled
Check that the trusted attribute setting of each task fits to the trusted attribute setting of the owner application [SPMF92:0061].	

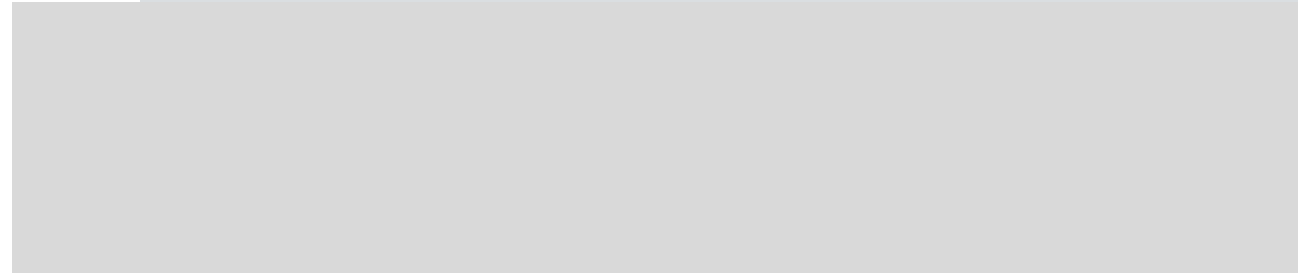
## 8.7 Task Stack Addresses

Container “Task Stack Lower Boundary Address” contains start address of all task stacks.



### Example

Container “Task Stack Lower Boundary Address” must look like:

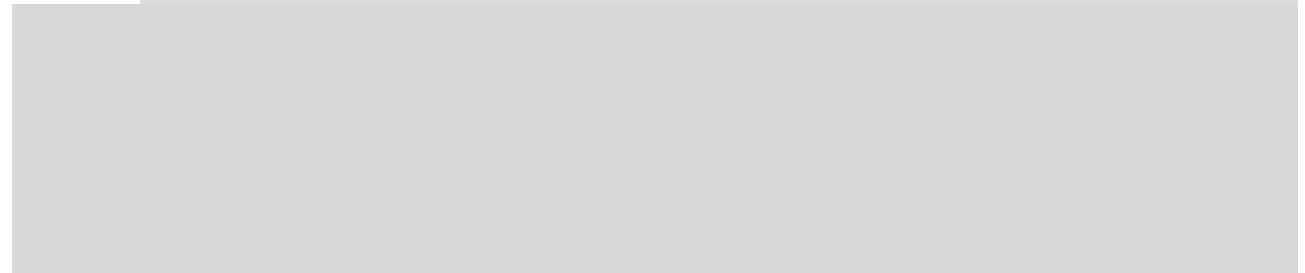


Container “Task Stack Upper Boundary Address” contains end address of all tasks stacks.



### Example

Container “Task Stack Upper Boundary Address” must look like:



Description of requirements to the OS user	Fulfilled
Check that the difference between stack start and stack end address fits to the configured size of each task stack.	
Check that the stacks do not overlap [SPMF92:0056]. Start address of stack on higher address must be $\geq$ end address of stack on lower address.	
Compare address value of each stack with the address given in the linker map file.	
Check that all task stack sizes are a multiple of 4 Bytes [SPMF92:04.0008]. This prevents that any data of another section is accessible in the same MPU region.	

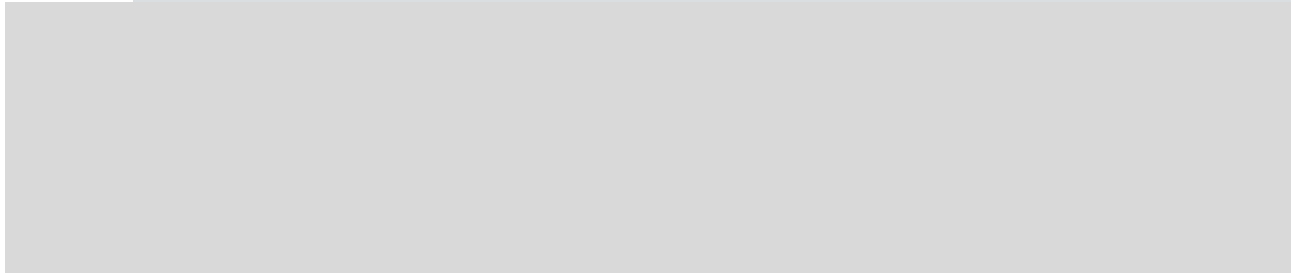
## 8.8 Task to Application Mapping

Container “Task to Application Mapping” contains the assignment of tasks to applications.



### Example

Container “Task to Application Mapping” must look like:



Description of requirements to the OS user	Fulfilled
The configuration of each application contains a list of the tasks which belong to this application. Check for each of the named applications that it owns exactly those tasks listed in the configuration viewer output. [SPMF92:0062]	

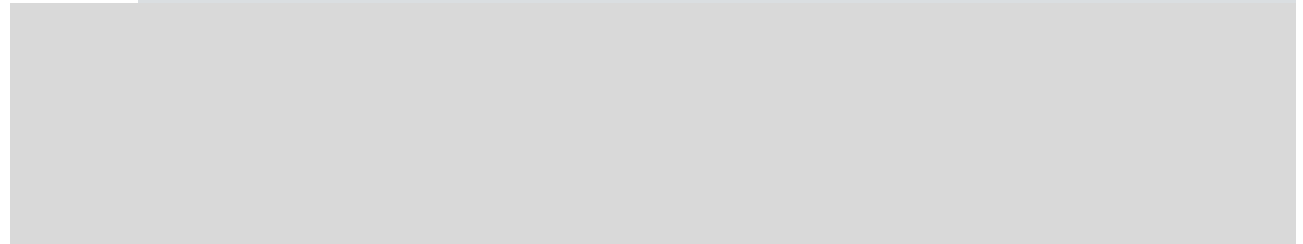
## 8.9 Category 2 ISR Trusted Configuration

Container “Category 2 ISR Trusted Attribute Setting” contains information about which category 2 ISR is trusted or non-trusted.



### Example

Container “Category 2 ISR Trusted Attribute Setting” must look like:



Description of requirements to the OS user	Fulfilled
Check that the trusted attribute setting of each category 2 ISR fits to the trusted attribute setting of the owner application.	



### Note

The trusted attribute setting of the system timer ISR `osOstmInterrupt_c<CoreID>` cannot be configured because it is always a trusted ISR.

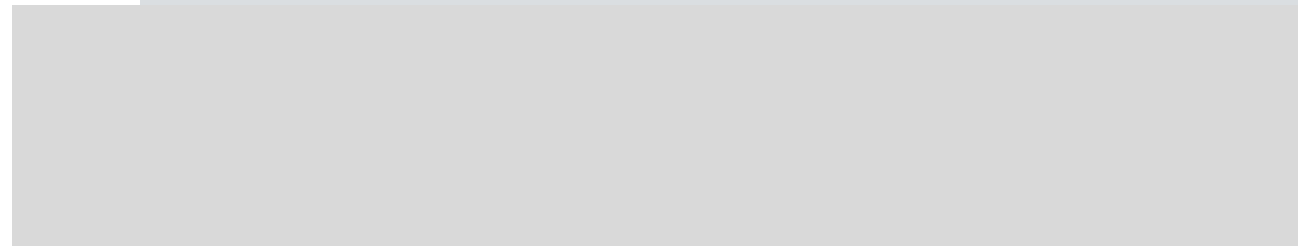
## 8.10 Category 2 ISR to Application Mapping

Container “Category 2 ISR to Application Mapping” contains the assignment of category 2 ISR functions to applications.



### Example

Container “Category 2 ISR to Application Mapping ” must look like:



Description of requirements to the OS user	Fulfilled
The configuration of each application contains a list of ISRs which belong to this application. Check for each of the named applications that it owns exactly those ISRs which are listed in the configuration viewer output [SPMF92:02.0028].	
Check that category 1 ISRs are not listed in the configuration block [SPMF92:0055].	



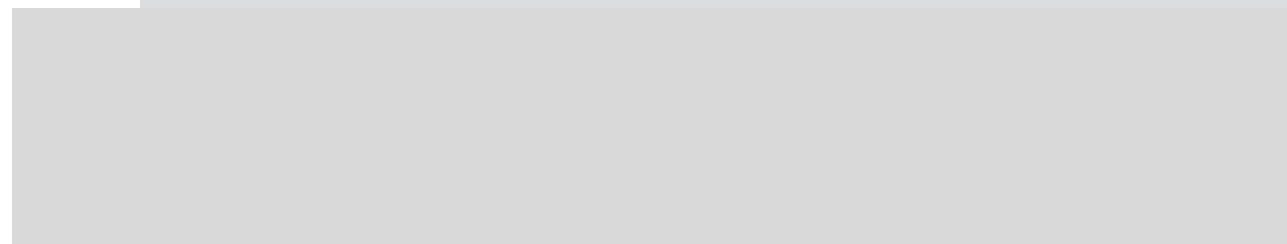
## 8.11 Application Trusted Configuration

Container “Application Trusted Attribute Setting” contains information about which application is trusted or non-trusted.



### Example

Container “Application Trusted Attribute Setting” must look like:



Description of requirements to the OS user	Fulfilled
Check that the trusted attribute of the listed applications matches to the OS configuration. [SPMF92:02.0027]	

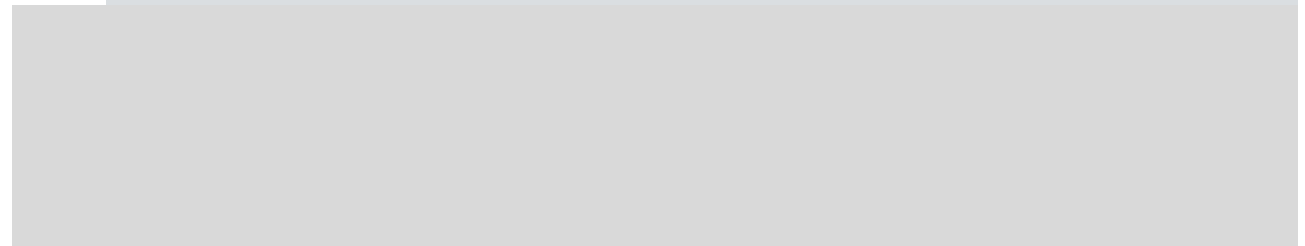
## 8.12 Trusted Functions Configuration

Container “Trusted Functions Start address and Application Mapping” contains the configuration of trusted functions.



### Example

Container “Trusted Functions Start Address and Application Mapping” must look like:



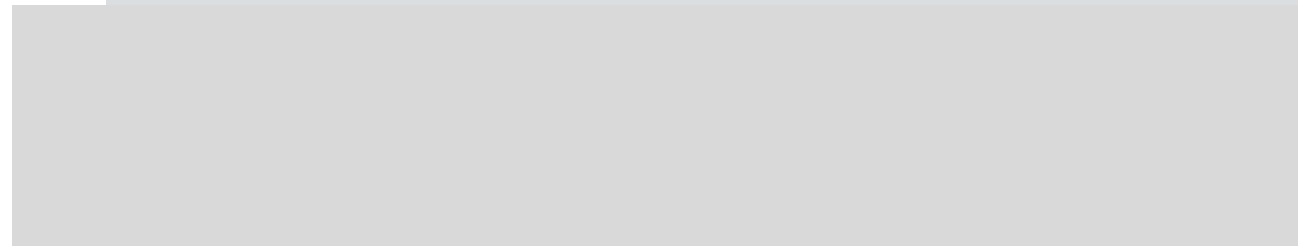
Description of requirements to the OS user	Fulfilled
Check that the listed Trusted Function IDs match to the corresponding defines of function names generated in tcb.h. [SPMF92:0048]	
Check that the listed Trusted Function start addresses matches the Trusted Function start addresses called <i>TRUSTED_&lt;TrustedFunctionName&gt;</i> (see linker map file). [SPMF92:0057]	
The configuration of each application contains a list of the trusted functions which belong to this application. Check for each of the named applications that it owns exactly those trusted functions listed in the configuration viewer output.	

### 8.13 Non-Trusted Functions Configuration

Container “Non-Trusted Functions Start Address and Application Mapping” contains the configuration of non-trusted functions.

**Example**

Container “Non-Trusted Functions Start Address and Application Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Non-Trusted Function start addresses matches the address of symbols <b>NONTRUSTED_&lt;NonTrustedFunctionName&gt;</b> (see linker map file). [SPMF92:02.0024]	
Check that the listed Non-Trusted Function to application assignment matches the OS configuration. [SPMF92:02.0029]	

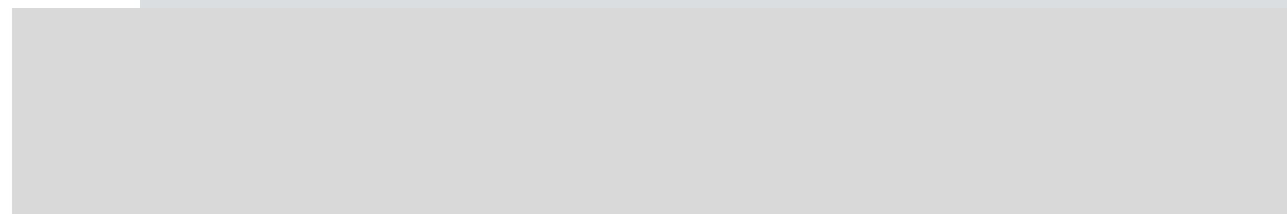
## 8.14 Category 2 ISR Start Addresses

Container “Category 2 ISR Function Start Address” contains the start address of all category 2 ISR functions.



### Example

Container “Category 2 ISR Function Start Address” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed category 2 ISR start addresses match to the function start address called <code>_&lt;ISR-Name&gt;func</code> (see linker map file). [SPMF92:02.0026]	



### Note

In case the ISR has a configured  , search for the symbol   instead, which is located at the address, shown in the configuration block.

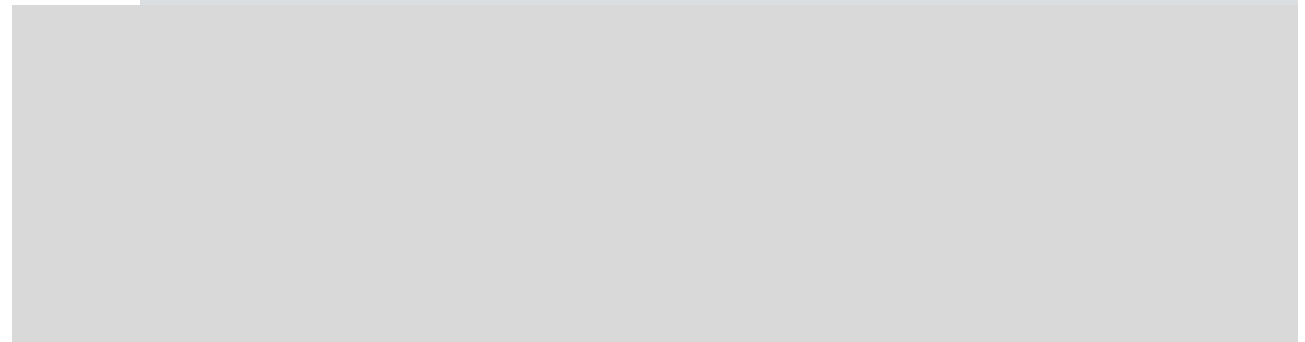
## 8.15 Category 2 ISR Nesting Configuration

Container “Category 2 ISR Nested Attribute Setting” contains information about which category 2 ISR is nestable or non-nestable.



### Example

Container “Category 2 ISR Nested Attribute Setting” must look like:

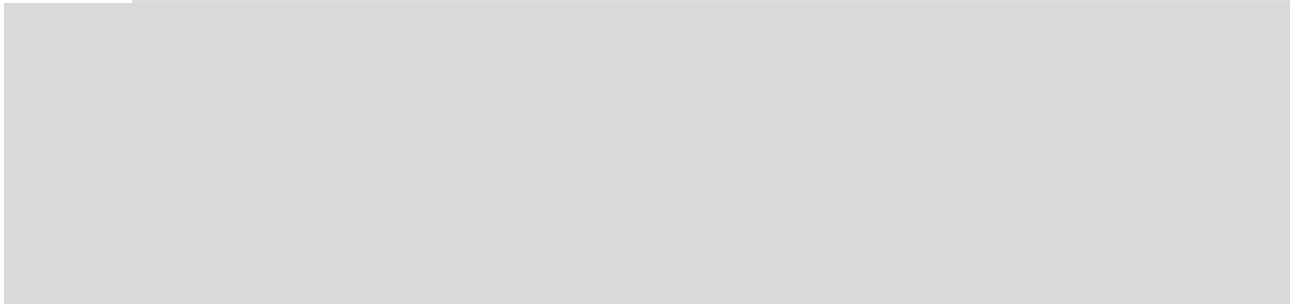


Description of requirements to the OS user	Fulfilled
Check that the listed category 2 ISR nested attributes match the OS configuration. [SPMF92:02.0031]	

8.16 Process to Core Mapping

Container “Process to Core Mapping” contains mapping information of all processes to the available cores. The term “process” means the superset of Tasks and ISRs.

**Example**  
Container “Process to Core Mapping” must look like:



Description of requirements to the OS user					Fulfilled
Check that the listed Tan	S to	]	yl}	<input type="checkbox"/>	ont rm

## 8.17 Alarms to Core Mapping

Container “Alarms to Core Mapping” contains the mapping of Alarms to processor cores.



### Example

Container “Alarms to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Alarms to core assignment matches the OS configuration.	

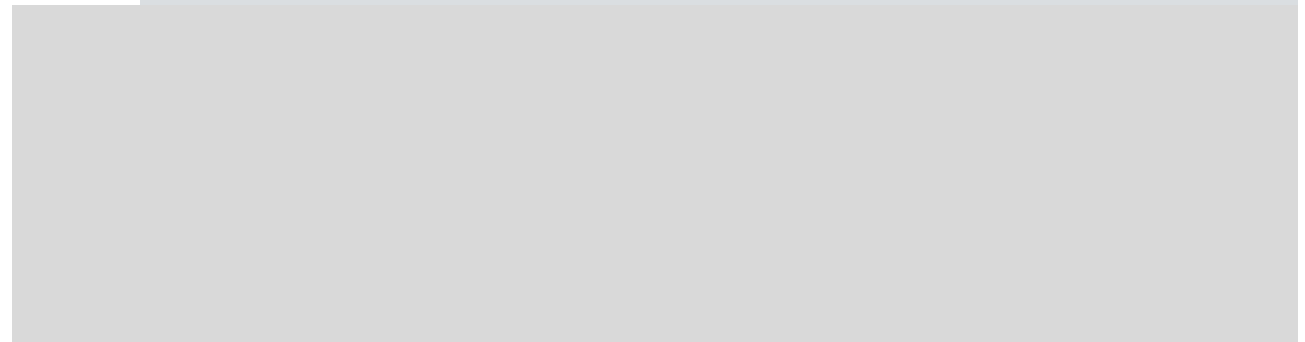
## 8.18 Resources to Core Mapping

Container “Resources to Core Mapping” contains the mapping of Resources to processor cores.



### Example

Container “Resources to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Resources to core assignment matches the Resource usage. All Tasks and ISRs, which use a Resource, shall belong to OS Applications, which belong to the listed core.	



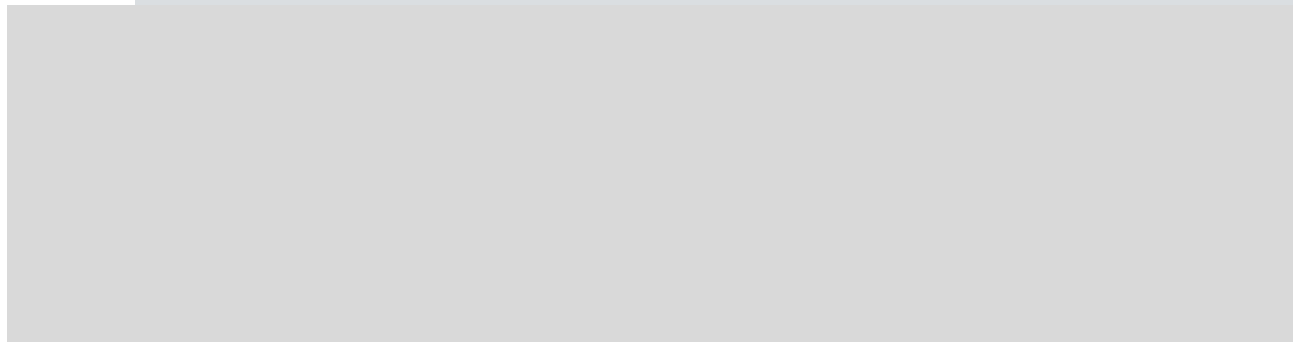
## 8.19 Counters to Core Mapping

Container “Counters to Core Mapping” contains the mapping of Counters to processor cores.



### Example

Container “Counters to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Counter to core assignment matches to the OS configuration.	

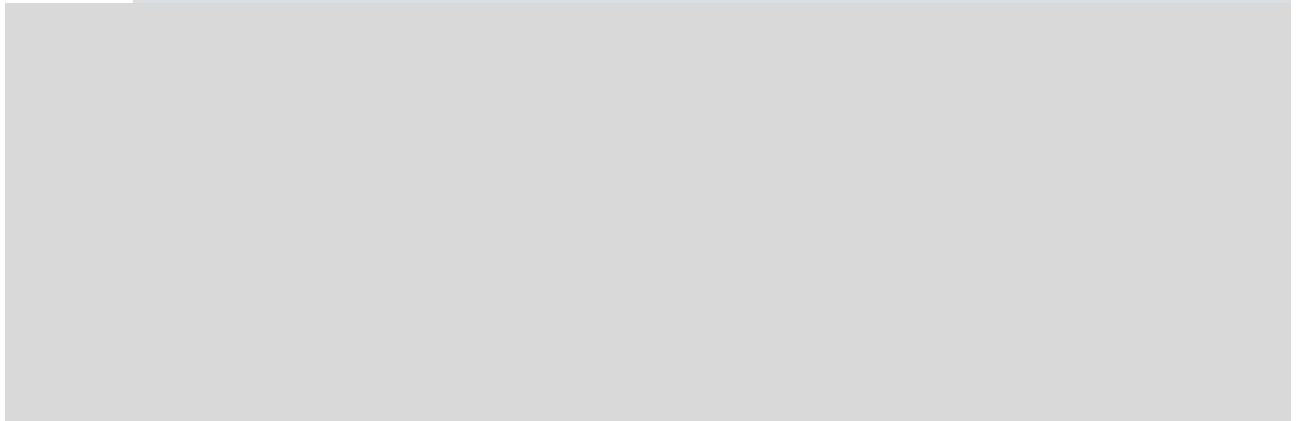
## 8.20 Schedule Tables to Core Mapping

Container “Schedule Tables to Core Mapping” contains the mapping of Schedule Tables to processor cores.



### Example

Container “Schedule Tables to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Schedule Tables to core assignment matches the OS configuration.	

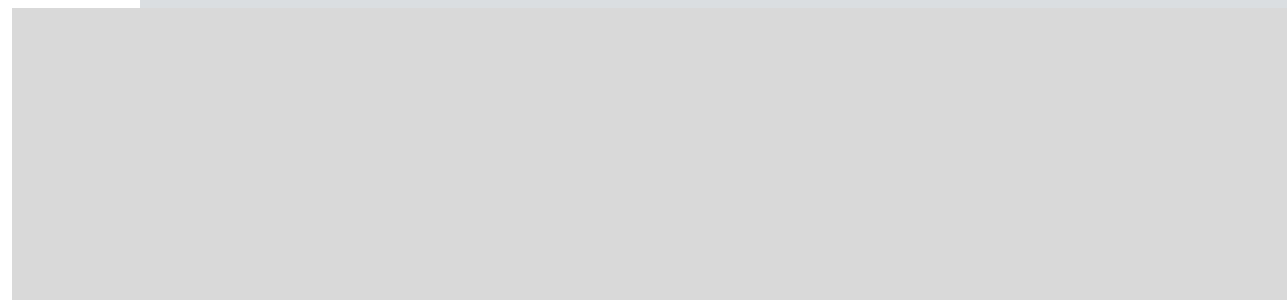
## 8.21 Application to Core Mapping

Container “Applications to Core Mapping” contains the mapping of applications to processor cores.



### Example

Container “Applications to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Applications to core assignment matches the OS configuration.	

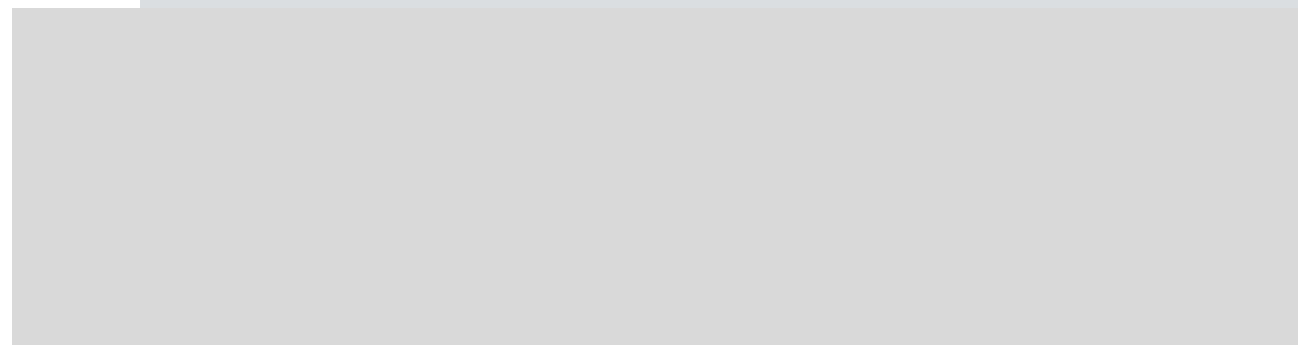
## 8.22 Trusted Functions to Core Mapping

Container “Trusted Functions to Core Mapping” contains the mapping of Trusted Functions to processor cores.



### Example

Container “Trusted Functions to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Trusted Functions to core assignment matches the OS configuration.	

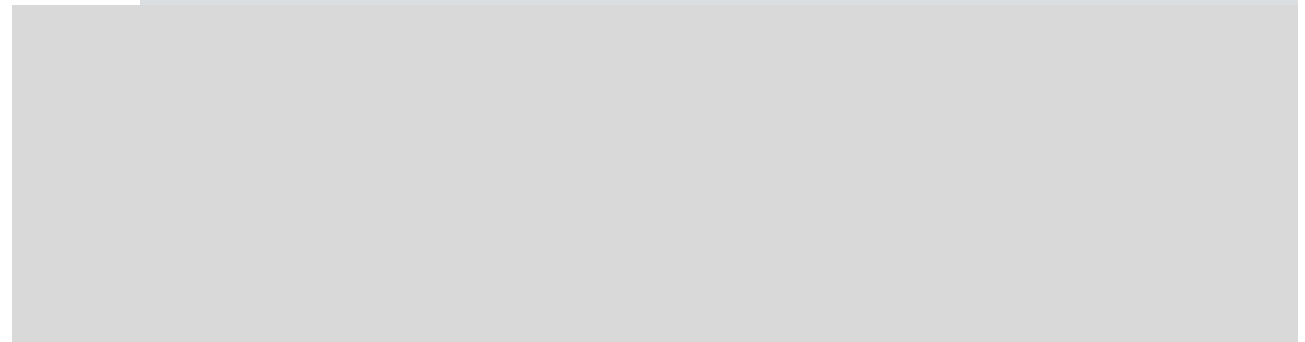
## 8.23 Non-Trusted Functions to Core Mapping

Container “Non-trusted Functions to Core Mapping” contains the mapping of Non-Trusted Functions to processor cores.



### Example

Container “Non-Trusted Functions to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Non-Trusted Functions to core assignment matches the OS configuration.	

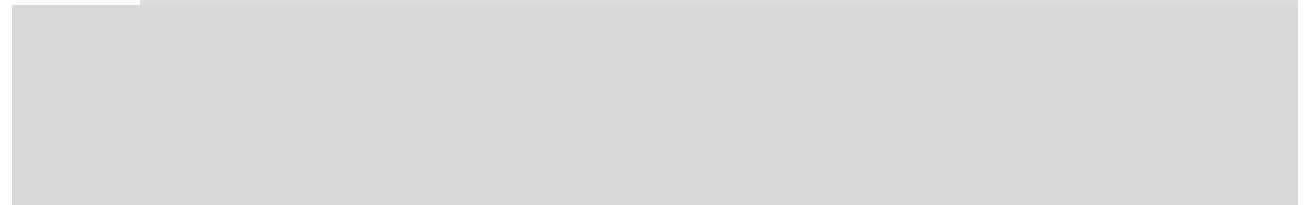
## 8.24 Core Control Block Address

Container “Core Control Block Start Address” contains the start address of the core specific control block data structure.



### Example

Container “Core Control Block Start Address” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed Control Block Start Address matches to the start address of RAM data structure called <code>osCtrlVarsCore0</code> (see linker map file).	

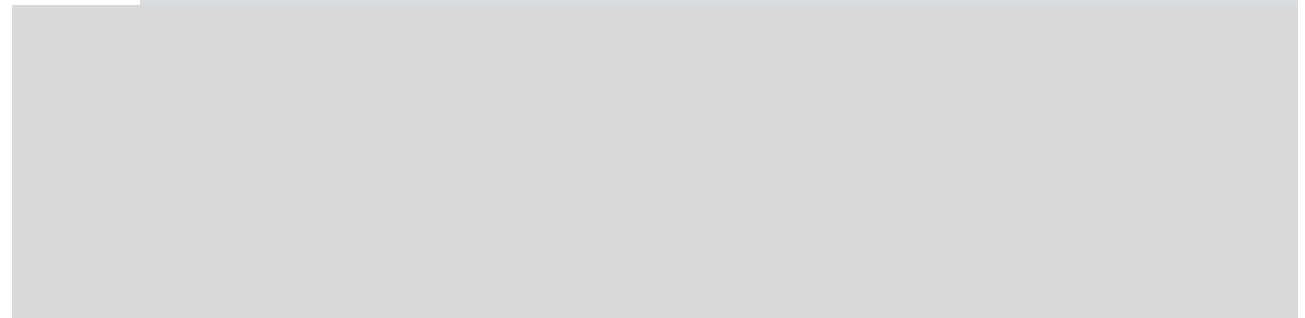
## 8.25 Peripheral Regions Configuration

Container “Peripheral Regions Configuration” contains information about peripheral regions configuration. [SPMF92:02.0030]



### Example

Container “Peripheral Regions Configuration” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed peripheral region start and end addresses match the OS configuration.	
Check that the listed accessing application names match the OS configuration.	

## 8.26 Spinlock Lock Method

Container “Spinlock Lock Method” contains information about how spinlocks prevent interruption by ISRs and pre-emption by tasks with higher priority.

Possible settings are:

This container is always available but only relevant in multi core systems.



### Example

Container “Spinlock Lock Method” must look like:

24. Spinlock Lock Method: N/A

## 8.27 Spinlock Config Type

Container “Spinlock Config Type” contains information whether the spinlock type is standard AUTOSAR or optimized.

This container is always available but only relevant in multi core systems.



### Example

Container “Spinlock Config Type” must look like:

25. Spinlock Config Type: N/A

## 8.28 Optimized Spinlock Variable Addresses

Container “Optimized Spinlock Variable Addresses” contains information about the variable addresses of optimized spinlocks. Standard AUTOSAR spinlocks have no such address, so there will be a NULL pointer for them.

This sub-container is always available but only relevant in multi core systems.



**Example**

Container “Optimized Spinlock Variable Addresses” must look like:

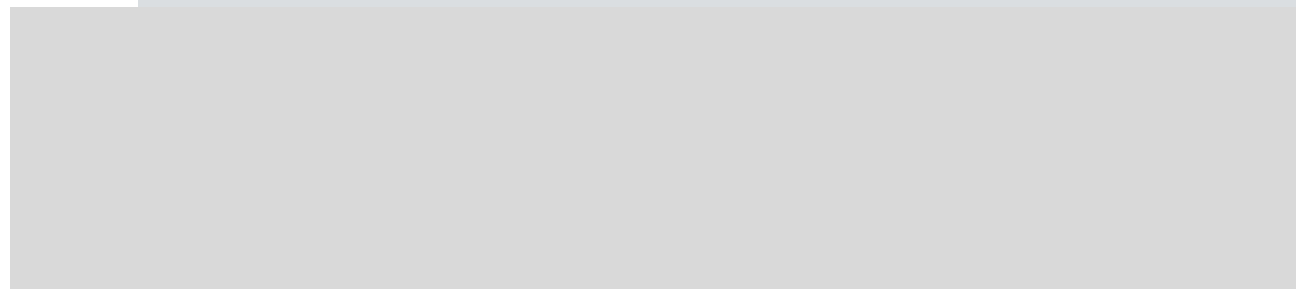
26. Optimized Spinlock Variable Address: N/A

## 8.29 Category 2 ISR Stack Address

Container “Category 2 ISR Stack Address Area” contains the stack start and stack end address of all category 2 ISR functions.

**Example**

Container “Category 2 ISR Stack Address Area” must look like:



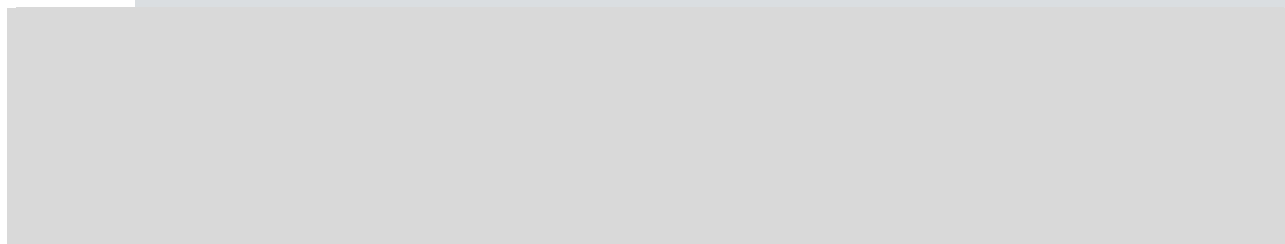
Description of requirements to the OS user	Fulfilled
Check that for each listed category 2 ISR the start and end address matches to the stack of the corresponding interrupt priority level. See linker map file for symbols <code>_osIntStackLevel&lt;PriorityLevel&gt;_c&lt;CoreID&gt;</code>	

## 8.30 Category 2 ISR Interrupt Channel Index

Container “Category 2 ISR Interrupt Channel Index” contains the interrupt channel index of all category 2 ISR functions.

**Example**

Container “Category 2 ISR Interrupt Channel Index” must look like:



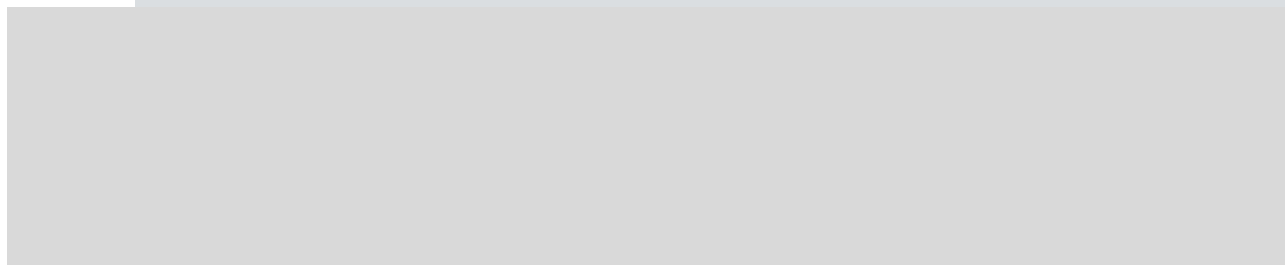
Description of requirements to the OS user	Fulfilled
Check that for each listed category 2 ISR the interrupt channel index matches to the OS configuration.	

### 8.31 Category 2 ISR Priority Level

Container “Category 2 ISR Interrupt Priority Level” contains the priority level of all category 2 ISR functions.

**Example**

Container “Category 2 ISR Interrupt Priority Level” must look like:



Description of requirements to the OS user	Fulfilled
Check that for each listed category 2 ISR the interrupt priority level matches to the OS configuration.	

Description of requirements to the OS user	Fulfilled
Note: The interrupt priority of system ISR osSystemCat2ISR is always 128.	

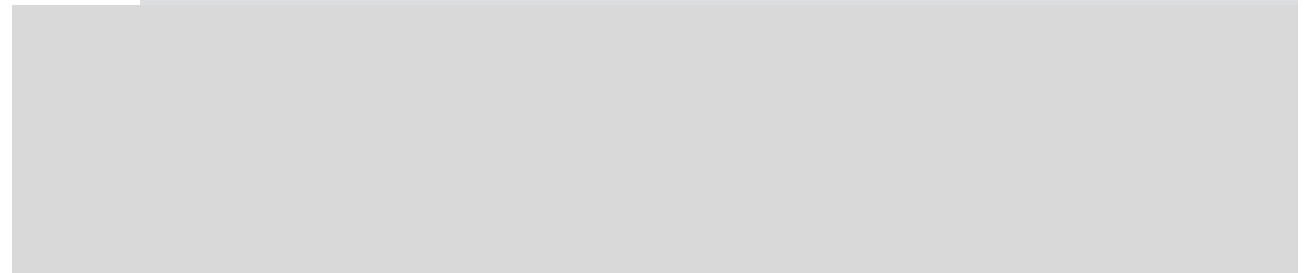
### 8.32 Category 2 ISR to Core Mapping

Container “Category 2 ISR to Core Mapping” contains the mapping of category 2 ISRs to processor cores. [SPMF92:02.0030]



#### Example

Container “Category 2 ISR to Core Mapping” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed category 2 ISR to core assignment matches the OS configuration.	

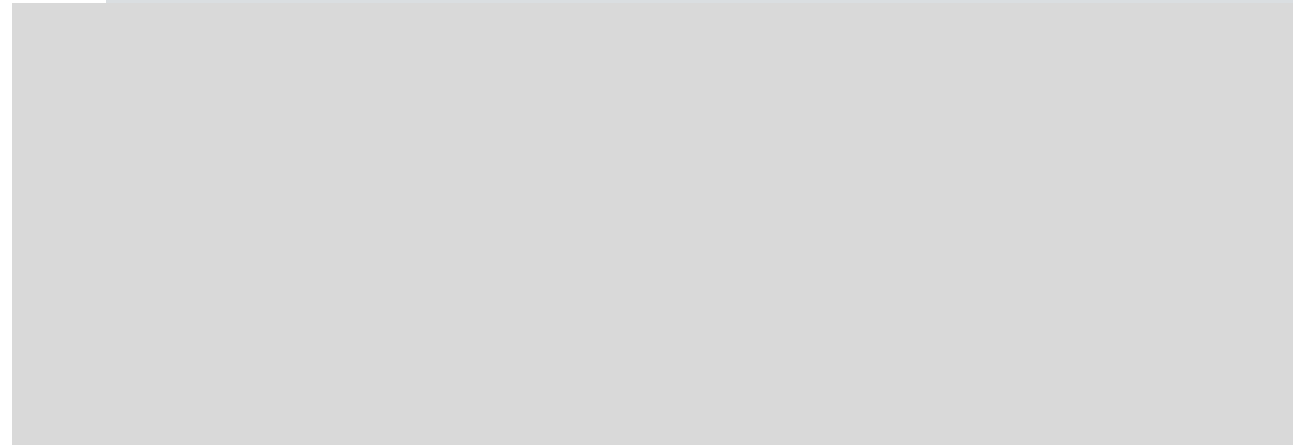
### 8.33 Application MPU Configuration

Container “Application MPU Configuration” contains the dynamic MPU region settings of all applications. [SPMF92:02.0030]



#### Example

Container “Application MPU configuration” must look like:



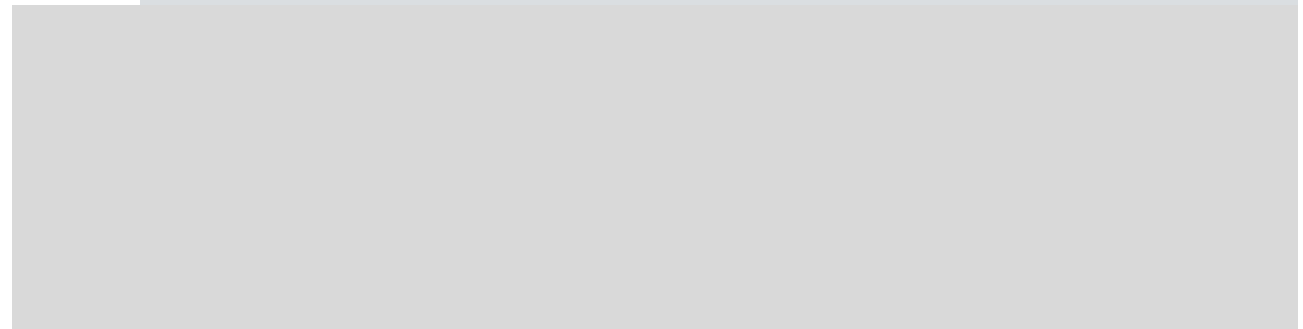
Description of requirements to the OS user	Fulfilled
Trusted applications must always have Start-Addr = 0x00000010 and End-Addr = 0x00000000.	
Unused MPU regions in non-trusted applications must have Start-Addr = 0x00000010 and End-Addr = 0x00000000	
Check that values of Start-Addr and End-Addr in row of non-trusted applications matches the OS configuration.	
Check that MPU regions of trusted applications have Start-Addr value smaller than the End-Addr value.	
Check that unused MPU regions of non-trusted applications have Start-Addr value smaller than the End-Addr value.	
Check that all trusted and non-trusted applications are listed.	
Check that all Start-Addr values are aligned to 4 Byte boundary [SPMF92:04.0009]	
Check that all End-Addr values point to the last valid byte in the specified area.	
Overlapping of memory regions is not allowed [SPMF92:04.0010].	
The next region after the end address must be aligned at to a 4 Byte boundary.	
Compare Start-Addr of non-trusted applications that the address value fits to address of the application specific linker symbol used in configuration settings [SPMF92:0052].	
Compare End-Addr of non-trusted applications that the address value fits to address of the application specific linker symbol used in configuration settings [SPMF92:0052].	
If a linker section for application data memory region is empty then the end address is below the start address. The start or end address may then overlap with other linker sections. This can be ignored because it does not harm the MPU functionality.	

### 8.34 MPU Configuration

Container “MPU Configuration” contains the MPU regions settings [SPMF92:02.0030]. The OS uses the settings to initialize the MPU during StartOS. Static regions stay unchanged after StartOS. Stack region and dynamic regions are reprogrammed when context is switched.

**Example**

Container “MPU configuration” must look like:



The MPU configuration in ConfigBlock must be reviewed by the user: [SPMF92:04.0005]

Description of requirements to the OS user	Fulfilled
Check that the total number of Region matches the number of MPU regions which are provided by the used CPU derivative.	
Dynamic regions must have the following entries: Start-Addr=0x00000010, End-Addr=0x00000000 and Attributes=0x000000c3.	
Unused regions must have the following entries: Start-Addr=0x00000010, End-Addr=0x00000000 and Attributes=0x00000000	
Region 0 Start-Addr must match to the system stack start address.	
Region 0 End-Addr must match to the system stack end address – 4 Bytes.	
Region 0 entry Attributes must match 0x000000c3.	
Check that the number of static regions matches the OS configuration.	
Check each static region that Start-Addr and End-Addr match the OS configuration.	
Check each static region that Start-Addr is lower than End-Addr.	
Check each static region that Attributes access rights matches the OS configuration.	
Check each static region that Attributes access rights contain the correct ASID.	

**Note**

Default ASID value is 0x3ff if no ASID is configured.

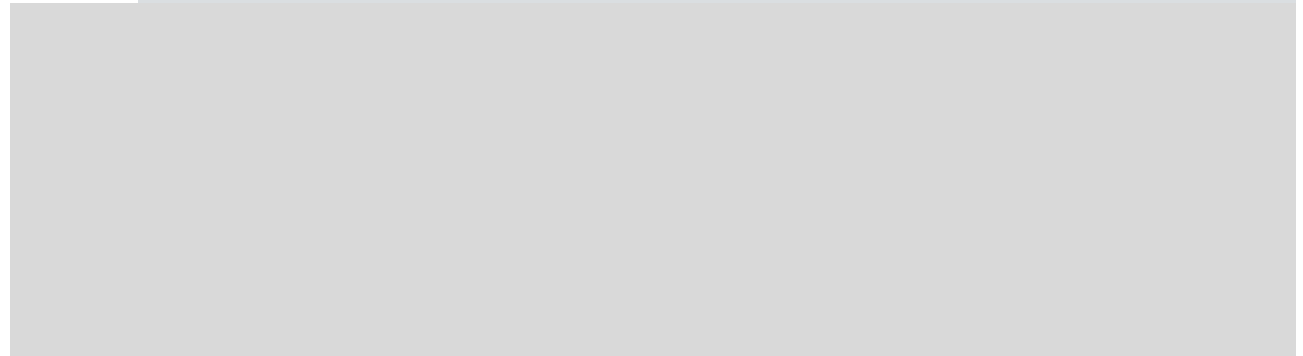
RH850 F1L does not support ASID due to hardware restrictions.

### 8.35 Application MPU ASID Configuration

Container “Application MPU ASID Configuration” contains the MPU ASID settings of all applications.

**Example**

Container “Application MPU ASID configuration” must look like:



Description of requirements to the OS user	Fulfilled
Check that the listed ASID configuration matches the OS configuration.	
All applications with ASID value 0x03ff must not have an ASID identifier configured.	

**Note**

Default ASID value is 0x3ff if no ASID is configured.

RH850 F1L does not support ASID due to hardware restrictions.

## 9 Generated OS Code

MICROSAR OS is a massively configurable software component. As a result, the analysis of the OS modules cannot be completely performed until the user's configuration data is available. The user shall use MICROSAR Safe Silence Verifier (MSSV) to qualify the generated part of the OS, which depends on user's configuration. MSSV is a Vector tool, which performs checks of potential dangerous code constructs. [SPMF92:0049] For more information about MSSV see the Technical Reference Manual of MSSV [4].

Description of requirements to the OS user	Fulfilled
The user must not modify a generated module configuration code file manually unless explicitly required by the technical reference manual or explicitly direction formulated by Vector.	
All generated files of a software project shall be generated based on the same configuration. Generated files of several configurations must not be mixed up unless explicitly allowed by Vector.	
The user shall apply steps for qualifying the generated sources on the final configuration which is used for the production. If the configuration changes, source qualification steps shall be reapplied.	

### 9.1 Using MICROSAR Safe Silence Verifier (MSSV)

The following chapter tells how you shall apply MSSV.exe on the OS sources.

MSSV.exe is called with the following parameters:

```
MSSV.exe -i <SourcePath> -i <GenPath> -i <HeaderPath> -D osdNOASM
```

Parameters	Description
-i <SourcePath>	This parameter is multiple used: At least path of OS source files (e.g. osek.c), path of generated OS files (e.g. tcb.c) and path of additional header files (e.g. Std_Types.h) must be specified.
-r <ReportFile>	Optional: Path and name which shall be used to save the MSSV report. If not used then the results will be saved in file report.html
-D osdNOASM	Disable OS assembler code parts. Mandatory for OS. Parameter -D can be multiple used.
--help	Show help

Description of requirements to the OS user	Fulfilled
The MICROSAR Safe Silence Verifier shall only be executed on Windows XP SP3+ (32Bit) or Windows 7 (32Bit or 64Bit).	
The user must not modify the MICROSAR Safe Silence Verifier report.	

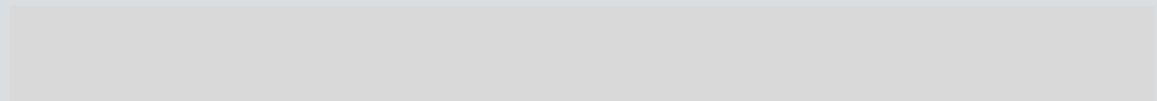


Description of requirements to the OS user	Fulfilled
The user shall verify that the used OS sources are checked by verifying the names and paths of the modules within the report.	
The user shall verify that the evaluated report matches to the execution of the MICROSAR Safe Silence Verifier by verifying the name, creation date and time, path and folder of the report.	
Check that MSSV returns with no errors, no warnings and the final verdict of the report is "pass". If MSSV did not pass contact OS support. [SPMF92:0088]	

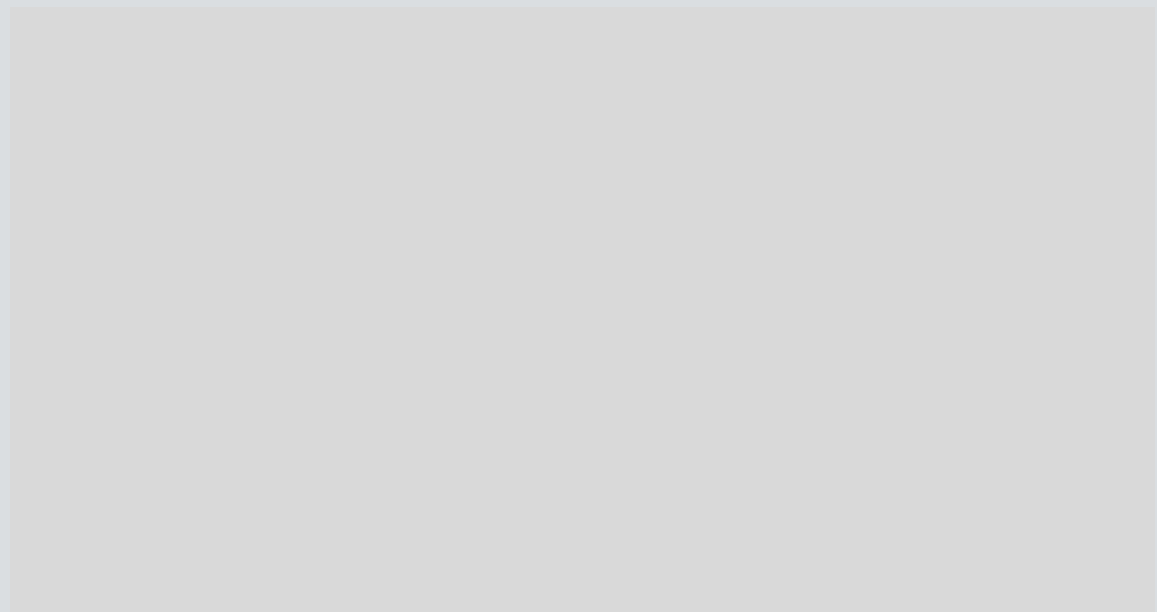


### Example

An example for qualifying generated OS sources:



The output of MSSV.exe must look like:



## 9.2 Manual Reviews

Some generated code parts currently cannot be checked automatically. Therefore the user has to check them manually.

### 9.2.1 Review generated file tcb.h

Description of requirements to the OS user	Fulfilled
<p>1. You shall review the system level definition to be the maximum of all category 2 ISR priorities and check mask definition to be the corresponding value, which has to be stored in PMR register to mask (i.e. disable) all category 2 interrupts. [SPMF92:02.0019] [SPMF92:04.0017] [SPMF92:02.0032] [SPMF92:02.0033] [SPMF92:0059] [SPMF92:0060]</p> <div></div>	
<p>2. Check that osdExceptionDetails is defined = 1</p> <div></div>	

## 9.2.2 Review of tcb.c

The function `osSysActivateTask` is used for handling Expiry Points. It calls OS internal functions to perform the configured actions at the configured Expiry Points. This function is generated, therefore the user has to review that safety requirements are fulfilled.



### Example

The function




may look like:






Description of requirements to the OS user	Fulfilled
Check that <code>osSysActivateTask</code> is only called by <code>osSysActivateTask</code> in <code>osSysActivateTask</code> and nowhere else.	
Check for all <code>osSysActivateTask()</code> calls pass valid Task names	

Description of requirements to the OS user	Fulfilled
User has to review that the parameter of each call of <code>osSysActivateTask</code> in <code>tcb.c</code> is a valid task identifier (define in <code>osSys.h</code> ). [SPMF92:05.0009]	
<i>Check for all <code>osSysSetEvent()</code> calls pass valid Task and Event names</i> The user has to review that the first parameter of all calls of <code>osSysSetEvent()</code> in <code>tcb.c</code> is a valid task identifier (define in <code>osSys.h</code> ) and that the value of the <code>EventName</code> define is smaller than <code>OS_MAX_EVENT_NAME</code> . [SPMF92:05.0006]	
Check that the array <code>osSysAlarmIndex</code> contains <code>OS_MAX_ALARM</code> elements and that each element contains the index of the counter related to that alarm.	
Check that the array <code>osSysTaskID</code> contains valid task ID values (Task ID of either the activated task or the Task ID of the task which receives an event).	
Check that <code>osSysPriority</code> contains priorities smaller than <code>OS_MAX_PRIORITY</code> .	
Check the size of all <code>osSysAlarmIndex</code> arrays The size must be the value of <code>OS_MAX_ALARM</code> . [SPMF92:05.0012]	
Check that the array <code>osSysAlarmIndex</code> has the size of <code>OS_MAX_ALARM</code> . [SPMF92:05.0002]	
Check that all elements of one entry in <code>osSysAlarmIndex</code> are related to the same counter <code>osSysAlarmIndex[CounterName]</code> , [SPMF92:05.0002]	
Check for the correct ordering of the entries in <code>osSysAlarmIndex</code> . Correct ordering means that the first entry must be related to the counter which is defined to zero (in <code>osSys.h</code> ). The second entry must be related to the counter which is defined to one etc. [SPMF92:05.0002]	
Check for the correct size of the Heap arrays. The size of such an array <code>osSysAlarmIndex</code> must be <code>OS_MAX_ALARM</code> . Number of alarms related to counter <code>&lt;Counter Name&gt;</code> + Number of Scheduletables related to <code>&lt;Counter Name&gt;</code> + 1 [SPMF92:05.0005]	

### 9.2.3 Review of tcbpost.h

Description of requirements to the user	Fulfilled
<p>Check that the Alarm names are mapped onto adjoining numbers. Starting with zero to            (                      is defined in                      ) [SPMF92:05.0013]</p> <div>  <div>Example</div> </div>	
<p>Check that the Counter names are mapped onto adjoining numbers. Starting with zero to            (                      is defined in                      ) [SPMF92:05.0003]</p> <div>  <div>Example</div> </div>	
<p>Check that the task names are mapped onto adjoining numbers starting from zero to            (                      is defined in                      ) [SPMF92:05.0010]</p> <div>  <div>Example</div> </div>	

<p>Check that the category 2 ISR names are mapped onto adjoining numbers starting from zero to</p> <p>(                                  is defined in                                  ) [SPMF92:05.0015]</p> <div data-bbox="215 367 1267 687">  <div>Example</div> </div>	
<p>Check that the Schedule Table names are mapped onto adjoining numbers. Starting with zero to</p> <p>(                                  is defined in                                  )</p> <div data-bbox="215 893 1267 1090">  <div>Example</div> </div>	
<p>Check that the values of                                  are defined to</p> <div data-bbox="215 1261 1267 1608">  <div>Example</div> </div>	

### 9.2.4 Review of trustfct.c & trustfct.h

The following review checks are only relevant for trusted functions that belong to an OS-application which has the parameter set to .

The OS provides the possibility to use generated stub functions for the call of trusted functions. These stubs are generated into the following files:

>

>

Both files are generated on QM level but executed in supervisor-mode. Therefore the generated code shall be reviewed.

If stubs are generated for a trusted function, there are two parts: the Caller stub and the Callee-stub. The Caller-stub is a function which packs the parameters which shall be passed to the trusted function into a C-struct and calls the API function . The Callee-stub unpacks the parameters from the struct and passes them to the users trusted function.

#### 9.2.4.1 File trustfct.c

Review the file according the following review criteria.

Description of requirements to the OS user	Fulfilled
The name of the trusted function caller function body shall be <div></div>	
If second parameter of API function is then the following local variable shall exist: <div></div>	
Check that the API function only uses a reference to this local variable.	
The first parameter shall be: <div></div>	
If the trusted function has no arguments, then the second parameter shall be: <div></div>	
If the trusted function has arguments, then the second parameter shall be: <div></div>	
Callee-stubs shall have the following function prototype: <div></div>	
Callee-stubs shall not modify any ASIL data.	

Description of requirements to the OS user	Fulfilled
Callee-stubs shall not call any function other than the specific trusted function.	
Callee-stubs shall not consume more stack space than available.	
Trusted functions which have arguments shall be called with following parameter notation syntax: <div></div>	



### Example

The following trusted function configuration:

Leads to the following code in :



### 9.2.4.2 File trustfct.h

Review the file according the following rules:

Description of requirements to the OS user	Fulfilled
<p>Each trusted function shall have an index definition with following notation syntax:</p> <p>&gt; If configuration attribute is “ ” “:”</p> <p>_____</p> <p>&gt; If configuration attribute is “ ” “:”</p> <p>_____</p>	
<p>The mapping of trusted function names to indexes shall be consistent to the mapping of trusted function start addresses to indexes as printed in the configuration block output. [SPMF92:0048]</p>	
<p>Each trusted function stub which uses function arguments shall have its specific C-struct type definition</p>	
<p>This C-struct shall contain all arguments which are used by the specific trusted function.</p>	
<p>Check that the following union type definition is generated and check that each trusted function which uses function arguments is listed in this union type definition.</p> <p>_____</p>	

**Example**

The following trusted function configuration:

Leads to the following code in :



Description of requirements to the OS user	Fulfilled
<p><i>Usage of osCheckMPUAccess API</i></p> <p>If you are using the <span style="float: right;">API</span>, Check that the API function is only called with addresses which, reading and writing does not have any side effects (e.g. potentially not true for peripheral registers). [SPMF92:02.0017].</p>	
<p><i>If you have write access to stacks, stack overflows cannot be detected by hardware</i></p> <p>The OS cannot safely detect stack overflows in software which has write access to all stacks. If write access to all stacks is really needed (e.g. for RAM checking), the user has to ensure that the software does not produce a stack overflow! [SPMF92:0078]</p>	
<p><i>APIs in exception handlers</i></p> <p>Exception handlers must not call any OS API function beside:</p> <ul style="list-style-type: none"> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> </ul> <p>[SPMF92:0067]</p>	
<p><i>Category 1 ISRs shall be transparent</i></p> <p>All ISRs of category 1 must be implemented such that they are transparent with respect to the processor state for the code they interrupt. This includes core registers, MPU settings and the current interrupt priority.</p>	
<p><i>APIs in category 1 ISRs</i></p> <p>Category 1 ISRs shall not call any OS API function beside:</p> <ul style="list-style-type: none"> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> <li>&gt;</li> </ul> <p>[SPMF92:0067] [SPMF92:02.0018]</p>	
<p><i>Check out-parameters in Trusted Functions</i></p> <p>Trusted functions which get a pointer shall check the pointer address to be in an expected range before they write to the pointer address. This shall prevent overwriting of safety relevant data when writing to the pointer address. [SPMF92:0001]</p>	
<p><i>Check caller in Trusted Functions</i></p> <p>Trusted functions shall validate whether they are called by an authorized caller only. This may be done by using the API function <span style="float: right;">.</span> [SPMF92:0047]</p>	
<p><i>No (Non-)Trusted Functions in Hooks</i></p> <p>Hook routines shall not call any trusted function or non-trusted function.</p>	

Description of requirements to the OS user	Fulfilled
<p><i>No APIs in NMIs</i></p> <p>Non-maskable interrupts shall not use any OS APIs. [SPMF92:0019], [SPMF92:0067]</p>	
<p><i>Using Interrupt API before calling StartOS</i></p> <p>If the user needs to use the interrupt API before he calls <code>osStartOS</code>, he shall call <code>osDisableGlobalKM</code> and <code>osDisableGlobalUM</code>.</p> <p>After calling these functions interrupt API works only for the straight forward case. OS error handling and MPU won't be initialized, so the OS won't be able to handle any user errors or detect stack overflows.</p>	
<p><i>Functions <code>osDisable/EnableLevelKM</code> and <code>osDisable/EnableGlobalKM</code></i></p> <p>Functions <code>osDisableLevelKM</code>, <code>osEnableLevelKM</code>, <code>osDisableGlobalKM</code> and <code>osEnableGlobalKM</code> shall only be called by trusted applications, i.e. when CPU is in privileged mode. [SPMF92:0094]</p>	
<p><i>Functions <code>osEnableLevelKM</code>, <code>osEnableLevelUM</code>, <code>osEnableLevelAM</code></i></p> <p>Assure that these functions are only called if the interrupt level was equal to the task level at the matching disable function (especially: not called in ISR!) Mind that the level functions may use the global flag, if disabling on level is not supported for a specific hardware. [SPMF92:0095]</p>	
<p><i>Functions <code>osEnableGlobalKM</code>, <code>osEnableGlobalUM</code>, <code>osEnableGlobalAM</code></i></p> <p>Assure that these functions are only called if interrupts were enabled by means of the global flag at the matching disable function (especially: not called in ISR with <code>EnableNesting=FALSE</code>) Mind that the global interrupt API functions may use the interrupt level in case no global disabling is supported on a specific hardware or if timing protection is active. [SPMF92:0096]</p>	
<p><i>Functions <code>osEnableLevelKM</code>, <code>osEnableLevelUM</code>, <code>osEnableLevelAM</code> and functions <code>osEnableGlobalKM</code>, <code>osEnableGlobalUM</code>, <code>osEnableGlobalAM</code></i></p> <p>Assure that these functions are not called in a nested way, even if they seem to use different locking mechanisms. [SPMF92:0097]</p>	
<p><i>Header file <code>osek.h</code></i></p> <p>This header file is included into the OS almost everywhere. Make sure that you do not manipulate the OS in an unforeseen way. [SPMF92:0098]</p>	
<p><i>Timing Hooks</i></p> <p>Assure that hook functions do not violate safety goals especially do not manipulate safety relevant data, the interrupt state and do not cause stack overflow (prevention of stack overflow may be unnecessary if the hardware allows to keep the MPU active in privileged mode) (interrupt disabling by means of the global flag and restauration its's state before returning to the caller shall be explicitly allowed if no OS API functions are used for that). [SPMF92:0099]</p>	
<p><i>Timing Hooks Parameters</i></p> <p>Assure that parameters of the hook routines are not used to reference a memory location to write to without a check for validity. (Mind that the invalid values are usually big numbers). [SPMF92:0100]</p>	

Description of requirements to the OS user	Fulfilled
<i>Timing Hooks</i> The OS does not guarantee the call of the hook routines on ASIL level. There is no safety requirement which assures that the hooks are correctly called (at correct time, in correct context etc.). [SPMF92:0101]	
<i>Timing Hooks</i> Calling OS API functions in OS_VTH* hooks is not allowed. [SPMF92:0102]	
<i>Timing Hooks</i> Usage of OS variables in OS_VTH* hooks is dangerous as they may not be consistent. [SPMF92:0103]	
<i>Timing Hooks</i> Mind that the CAT1 ISRs may interrupt the hook routines which may cause concurrent calls. [SPMF92:0104]	

## 11 Hardware Specific Part

For RH850 SafeContext the following safety relevant requirements must be checked by the user:

- All assembly code (outside the SafeContext) shall be reviewed, not to change the content of registers of R4 and R5 after StartOS is called [SPMF92:04.0001]. Check in compiler list files that only the startup module and the OS modules do modify registers R4 and R5.
- The user has to review that each ISR is called at least once (coverage of application). The tests shall cover the activation of all ISRs and verify that the correct ISR was started. This measure shall prevent the activation of wrong ISRs because of a mix up in the interrupt vector table [SPMF92:0008].
- The user has to review the configuration by means of the ConfigBlock in accordance to the review rules which are defined in chapter 12.2 [SPMF92:0014],[SPMF92:05.0008].
- The user has to review that all libraries fit to the used compiler options. All used libraries need to be checked for using the correct compiler options (e.g. SDA usage need to be identical to the specified options for the OS) [SPMF92:0010].
- The PreTaskHook and the PostTaskHook must not be used in safety code which is released for serial production. Pre/PostTaskHook shall only be used for debugging or test purposes. Absence of Pre/PostTaskHook must be reviewed in generated file tcb.h: [SPMF92:02.0022],[SPMF92:02.0023],[SPMF92:05.0011]  
The user must check that the following defines are set in the generated file tcb.h:  

```
#define osdPreTaskHook 0
#define osdPostTaskHook 0
```
- The complete config block content must be reviewed by the means of the BackReader [SPMF92:04.0002], [SPMF92:04.0006].
- The address value of the application specific linker symbols for MPU region start and end address must be checked that between them only the corresponding application data sections are mapped [SPMF92:04.0004], [SPMF92:04.0010], [SPMF92:04.0011].
- The address value of the linker symbols `_osGlobalShared_StartAddr` and `_osGlobalShared_EndAddr` must be checked that between them only the global shared data sections are mapped [SPMF92:04.0013].
- The CPU must run in supervisor mode when StartOS is called [SPMF92:04.0007].
- The application shall check the config block version by using OS API function `osGetConfigBlockVersion` [SPMF92:0045], [SPMF92:04.0003]
- The user has to review that all task stacks, all ISR stacks and the system stack have 4 Byte alignments [SPMF92:04.0008].
- The user has to review the generated linker include files `osdata.dld`, `osrom.dld`, `osdata.dld`, `osstacks.dld` and `osdata.dld` if they are used for serial production [SPMF92:04.0014]. See chapter 11.3.
- The user has to review that coverage is disabled [SPMF92:04.0015]. `osdEnableCoverage` shall not be defined in header and source files and it shall not be defined via compiler option `-DosdEnableCoverage`.

- The user has to consider DMA controller usage if the used RH850 derivative incorporates a DMA controller (DMAC). The DMA controller has direct access to the data bus. Therefore DMA access to memory is not controlled by MPU protection. This must be considered especially for safety OS systems if any DMA access is wanted.
- The user has to review that `osInitialize` and `osInitINTC` are called before `StartOS` is called if OS interrupt API functions or OS peripheral interrupt API functions are used before `StartOS` [SPMF92:0070].
- The user has to review that all generated files belong together and that all generated files are compiled and linked [SPMF92:0064]. The user shall not change any generated header file (\*.h) or source file (\*.c).
- The user has to check that the application does not modify interrupt controller registers `EBASE`, `INTBP`, `INTCFG`, `SCBP` and `SCCFG` after `StartOS` is called [SPMF92:0069].
- The application shall not modify any register in interrupt controller unit `INTC` by own functions or routines after `StartOS` is called. The application shall only use the OS API functions for changing registers in unit `INTC`. [SPMF92:0083]
- The user has to check validity and type of the reference parameter when calling the following OS API functions [SPMF92:05.0001]:
  - `GetTaskID`
  - `GetTaskState`
  - `GetApplicationState`
  - `GetEvent`
  - `GetAlarm`
  - `GetScheduleTableStatus`
  - `GetCounterValue`
  - `GetElapsedValue`
  - `GetElapsedCounterValue`
- The user has to assure that symbol `_osStartupStack_StartAddr` is provided in linker file and points to the startup stack [SPMF92:05.0004].
- User has to review that `osdTimerInterruptSourceNumberCore0` in generated file `tcb.h` is equal to channel index of timer unit `OSTM0` [SPMF92:05.0007].
- User has to review in generated file `tcb.c` the content of arrays `"oskGetCurrentTimeOps"` and `"oskCounterId2GetCurrentTimeOpIdx"` [SPMF92:05.0016].
- User has to review in generated file `tcb.c` the arrays `"oskInsertMinHeapOps"` and `"oskCounterId2InsertMinHeapOpIdx"` [SPMF92:05.0017].
- The user has to check that the application does not modify CPU core register `PMR` after `StartOS` is called [SPMF92:04.0018].
- If High Resolution Timer is used as `SystemTimer` then the user has to review that the following defines exists in `tcbpost.h` and that `HiResSystemTimer` is defined smaller than `osdNumberOfCounters` [SPMF92:05.0018]:

```
#define HiResSystemTimer          <x>
#define osdTimerOSTM_HIRESID HiResSystemTimer
```



- If Standard Timer is used as SystemTimer then the user has to review that the following defines exists in tcbpost.h and that SystemTimer is defined smaller than osdNumberOfCounters [SPMF92:05.0019]:

```
#define SystemTimer    <x>
#define osdTimerOSTM  SystemTimer
```

- The user has to review in the generated file tcb.c that the size of the array oskAlarmHeapSize is equal to osdNumberOfCounters. Furthermore, the user has to review that each entry must be equal to 1 + the number of alarms that are related to the counter <CounterName> + the number of schedule tables that are related to the counter <CounterName>. The counter osSystemSWCounter is the only exception to this rule. This counter must always exist and the related array element must be one.

## 11.1 Interrupt Vector Table

Basically the interrupt vector tables must be provided by the application. An example vector table is generated into file `intvect_c<CoreID>.c`. This file is generated by QM software and must not be used directly as ASIL code. It must be reviewed carefully for compliance to the description below because the code which is called by the interrupt vector table runs automatically in supervisor mode and therefore this code must be developed according to ASIL level [SPMF92:0004], [SPMF92:02.0020], [SPMF92:02.0021], [SPMF92:04.0012].

The file `intvect_c<CoreID>.c` consists of the following parts:

- Header Include Section
- Core Exception Vector Table
- EIINT Vector Table
- CAT2 ISR Wrappers

The following subchapters describe these parts and do intentionally not describe any comments.

### 11.1.1 Header Include Section

Generated file `intvect_c<CoreID>.c` starts exactly with the following lines:

```
#if defined USE_QUOTE_INCLUDES
#include "vrm.h"
#else
#include <vrm.h>
#endif

#define osdVrmGenMajRelNum 1
#define osdVrmGenMinRelNum 6
#if defined USE_QUOTE_INCLUDES
#include "vrm.h"
#else
#include <vrm.h>
#endif

#if defined USE_QUOTE_INCLUDES
#include "Os.h"
#else
#include <Os.h>
#endif

#if defined USE_QUOTE_INCLUDES
#include "osekext.h"
#else
#include <osekext.h>
#endif

#ifdef osdNOASM
```

### 11.1.2 Core Exception Vector Table

The core exception vector table section looks exactly like the following lines:

```
#pragma asm

.section ".osExceptionVectorTable_c0", "ax"
.align 512

.globl _osExceptionVectorTable_c0
_osExceptionVectorTable_c0:

    .offset 0x0000
    .globl _osCoreException_c0_0x0000
_osCoreException_c0_0x0000:
    jr <interrupt_handler_0>

    .offset 0x0010
    .globl _osCoreException_c0_0x0010
_osCoreException_c0_0x0010:
    jr <interrupt_handler_1>

    .offset 0x0020
    .globl _osCoreException_c0_0x0020
_osCoreException_c0_0x0020:
    jr <interrupt_handler_2>

    ...

    .offset 0x01F0
    .globl _osCoreException_c0_0x01F0
_osCoreException_c0_0x01F0:
    jr <interrupt_handler_23>

    .globl _osExceptionVectorTableEnd_c0
_osExceptionVectorTableEnd_c0:

#pragma endasm
```

The sequence of core exception vector table entries starts at vector address 0x0000, increases in steps of 0x0010 and ends with vector address 0x01F0.

<interrupt\_handler\_x> is the name of the handler which is called when an exception occurs.

If no specific handler is configured then osUnhandledCoreException is called:

example for unused interrupt source

```
    .offset 0x0010
    .globl _osCoreException_c0_0x0010
_osCoreException_c0_0x0010:
    jr _osUnhandledCoreException
```

### 11.1.3 EIINT Vector Table

The EIINT vector table section starts exactly with the following lines:

```
#pragma asm

.section ".osEIINTVectorTable_c0", "ax"
.align 512

.globl _osEIINTVectorTable_c0
_osEIINTVectorTable_c0:

.word  _<EIINT_handler_0>
.word  _<EIINT_handler_1>
.word  _<EIINT_handler_2>
...
.word  _<EIINT_handler_x>
```

For each unused interrupt source the following table entry must be generated:

```
.word  _osUnhandledEIINTException
```

For each interrupt source used as category 1 ISR the following table entry must be generated:

```
.word  _<Cat1_EIINT_handler>
```

<Cat1\_EIINT\_handler> is the name of the application specific EIINT handler which is called when an interrupt on the corresponding source occurs.

For each interrupt source used as category 2 ISR the following table entry must be generated:

```
.word  _<Cat2_ISR_Name>_CAT2
```

<Cat2\_EIINT\_handler> is the name of the OS ISR wrapper which is called when an interrupt on the corresponding source occurs. The CAT2 ISR wrapper section is described in the next chapter.

The EIINT vector table section ends exactly with the following lines:

```
.globl _osEIINTVectorTableEnd_c0
_osEIINTVectorTableEnd_c0:

#pragma endasm
```

### 11.1.4 CAT2 ISR Wrappers

The category 2 ISR wrapper section looks exactly like the following lines:

```
#pragma asm

    .section ".os_text", "ax"

    osCAT2ISRC0 (...)

    osCAT2ISRC0 (...)

    osCAT2ISRC0 (...)

    ...

    osCAT2ISRC0 (...)

#pragma endasm
```

Each category 2 ISR handler must be generated exactly like the following line [SPMF92:0044]:

```
osCAT2ISRC0 (<Cat2_ISR_Name>, <Cat2_ISR_Priority>)
```

Macro osCAT2ISRC0() is defined in osext.h.

It defines the CAT2 ISR prologue for each interrupt source which is used as category 2 ISR.

<Cat2\_ISR\_Name> is the unique name of the ISR function. This must be the same name as used in the EIINT vector table at the corresponding channel index position with postfix:

```
.word  __<Cat2_ISR_Name>_CAT2
```

<Cat2\_ISR\_Priority> is the value of the interrupt priority level which is configured for the corresponding ISR. The valid range of <Cat2\_ISR\_Priority> is 0 ... 7

### 11.1.5 End of file Intvect\_c<CoreID>.c

Generated file intvect\_c<CoreID>.c ends exactly with the following line:

```
#endif /* ifndef osdNOASM */
```

## 11.2 Linker Memory Sections

The OS uses specific memory section names for the linker. Check that these section names are used in the linker file and check that they are used in the assigned area.

The OS uses the linker memory section names described in the following table:

Section Name and Type	Description and Link Requirements
.osExceptionVectorTable_c<CoreID> section type .text	Contains the core exception vector table. It is generated into file intvect_c<CoreID>.c
.osEIINTVectorTable_c<CoreID> section type .text	Contains the EIINT exception vector table. It is generated into file intvect_c<CoreID>.c
.os_text section type .text	Contains the interrupt handler for category 2 ISRs. It is generated into file intvect_c<CoreID>.c Must be linked to program code section.
.os_text section type .text	Contains all OS program code, except those which must be placed in special sections (e.g. vector table). Must be linked to program code section.
.os_rodata section type .rodata	Contains the OS constant data, except those which must be placed in special sections (e.g. configuration block osConfigBlock). Must be linked to constant data section.
.os_rodata section type .rodata	Contains all OS constants which are placed in ROSDA area, except those which must be placed in special sections (e.g. configuration block osConfigBlock). Must be linked to the constant data in ROSDA section
.osConfigBlock_rodata section type .rodata	Contains the configuration block if SDA optimization is disabled. Must be linked to constant data section.
.osConfigBlock_rodata section type .rodata	Contains the configuration block if SDA optimization is enabled Must be linked to constant data in ROSDA section.
.os_bss section type .bss	Contains the uninitialized OS variables Optional initialized to zero by system startup code. Must be linked to the data section.
.os_data section type .data	Contains the initialized OS variables which must be copied from ROM to RAM by system startup code. Must be linked to the data section. This section must be empty!
.os_sbss section type .sbss	Contains uninitialized OS variables which are placed in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to the SDA section.
.os_sdata section type .sdata	Contains initialized OS variables which are placed in SDA area if SDA optimization is enabled. Must be linked to the SDA section. This section must be empty!

Section Name and Type	Description and Link Requirements
.osTaskStack<TaskIndex> section type .bss	Contains the uninitialized task specific stack. Must be linked to the data section.
.osSystemStack_c<CoreID> section type .bss	Contains the uninitialized OS system stack. Must be linked to the data section.
.osIntStackLevel<Priority> section type .bss	Contains the uninitialized ISR specific stack. Must be linked to the data section.
.osAppl_<ApplName>_bss section type .bss	Contains uninitialized application data. Optional initialized to zero by system startup code. Must be linked to the data section.
.osAppl_<ApplName>_sbss section type .sbss	Contains uninitialized application data in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to SDA section.
.osAppl_<ApplName>_data section type .data	Contains initialized application data. Must be linked to data section.
.osAppl_<ApplName>_sdata section type .sdata	Contains initialized application data in SDA area if SDA optimization is enabled. Must be linked to SDA section.
.osGlobalShared_bss section type .bss	Contains uninitialized global shared data. Optional initialized to zero by system startup code. Must be linked to data section.
.osGlobalShared_sbss section type .sbss	Contains uninitialized shared data in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to SDA section.
.osGlobalShared_data section type .data	Contains initialized shared global data. Must be linked to data section.
.osGlobalShared_sdata section type .sdata	Contains initialized shared data in SDA area if SDA optimization is enabled. Must be linked to SDA section.

### 11.3 Linker Include Files

The generated linker include files shall be reviewed if they are used for serial production.

#### 11.3.1 Review File osdata.dld

File osdata.dld contains mapping of section types .bss and .data for trusted applications.

For each trusted application the generated lines must look like:

```
/* trusted application <ApplName> */  
.osAppl_<ApplName>_bss    align(4) :>.  
.osAppl_<ApplName>_data  align(4) :>.
```

The linker include file ends with the section mapping for OS data which must look like:

```
.os_bss    align(4) :>.  
.os_data   align(4) :>.
```



### 11.3.2 Review File ossdata.dld

The generated example file ossdata.dld contains the mapping of section types .sbss and .sdata for trusted and non-trusted applications.

For non-trusted applications it also contains the mapping of section types .bss and .data. This is necessary due to optimization for MPU region settings.

For each trusted application the generated lines must look like:

```
/* trusted application <AppName> */
.osAppl_<AppName>_sbss      align(4) :>.
.osAppl_<AppName>_sdata     align(4) :>.
```

For each non-trusted application the generated lines must look like:

```
/* non-trusted application <AppName> */
.osAppl_<AppName>_bss       align(4) :>.
.osAppl_<AppName>_data      align(4) :>.
.osAppl_<AppName>_sbss      align(4) :>.
.osAppl_<AppName>_sdata     align(4) :>.
_<Appl_Section_StartAddr> = addr(.osAppl_<AppName>_bss);
_<Appl_Section_EndAddr>   = endaddr(.osAppl_<AppName>_sdata)-1;
```

After the mapping of the application sections the mapping for OS SDA sections must be generated like the following lines:

```
.os_sbss      align(4) :>.
.os_sdata     align(4) :>.
```

After the mapping of the OS SDA sections the mapping for global shared sections must be generated like the following lines:

```
.osGlobalShared_sbss      align(4) :>.
.osGlobalShared_sdata     align(4) :>.
.osGlobalShared_bss       align(4) :>.
.osGlobalShared_data      align(4) :>.
_osGlobalShared_StartAddr = addr(.osGlobalShared_sbss);
_osGlobalShared_EndAddr   = endaddr(.osGlobalShared_data)-1;
```

### 11.3.3 Review File osstacks.dld

File osstacks.dld contains mapping of all stack sections.

Each stack section mapping for used interrupt priority levels must look like:

```
.osIntStackLevel<PrioLevel> align(4) :>.  
_osIntStackLevel<PrioLevel>_StartAddr = addr(.osIntStackLevel<PrioLevel>);  
_osIntStackLevel<PrioLevel>_EndAddr = endaddr(.osIntStackLevel<PrioLevel>);
```

<PrioLevel> is the interrupt priority level 0 ... 15

The mapping for the system stack section must look like:

```
.osSystemStack align(4) :>.  
_osSystemStack_StartAddr = addr(.osSystemStack);  
_osSystemStack_EndAddr = endaddr(.osSystemStack);
```

The system stack section is followed by the OS task stack sections which must look like:

```
.osTaskStackosSystemApplicationCore00 align(4) :>.  
_osTaskStackosSystemApplicationCore00_StartAddr =  
    addr(.osTaskStackosSystemApplicationCore00);  
_osTaskStackosSystemApplicationCore00_EndAddr =  
    endaddr(.osTaskStackosSystemApplicationCore00);  
  
.osTaskStackosSystemApplicationCore01 align(4) :>.  
_osTaskStackosSystemApplicationCore01_StartAddr =  
    addr(.osTaskStackosSystemApplicationCore01);  
_osTaskStackosSystemApplicationCore01_EndAddr =  
    endaddr(.osTaskStackosSystemApplicationCore01);
```

The OS task stack sections are followed by the application task stack sections.

Each application task stack section mapping must look like:

```
.osTaskStack<applname><index> align(4) :>.  
_osTaskStack<applname><index>_StartAddr = addr(.osTaskStack<applname><index>);  
_osTaskStack<applname><index>_EndAddr = endaddr(.osTaskStack<applname><index>);
```

<applname> is the name of the owner application

<index> is the index number of the task stack: 0 ... number of tasks per application

### 11.3.4 Review File osrom.dld

File osrom.dld contains the sections used for initialized variables.

For each application which has data to be initialized during startup code the following lines must be generated:

```
.ROM_osAppl_<ApplName>_data    ROM(.osAppl_<ApplName>_data)    :>.
.ROM_osAppl_<ApplName>_sdata    ROM(.osAppl_<ApplName>_sdata) :>.
.ROM_osAppl_<ApplName>_tdata    ROM(.osAppl_<ApplName>_tdata) :>.
```

File osrom.dld ends with the global shared initialized data sections which must look like:

```
.ROM_GlobalShared_data    ROM(.osGlobalShared_data)    :>.
.ROM_GlobalShared_sdata    ROM(.osGlobalShared_sdata) :>.
.ROM_GlobalShared_tdata    ROM(.osGlobalShared_tdata) :>.
```

### 11.3.5 Review File ostdata.dld

File ostada.dld contains the mapping for application data in TDA section.

For each application which has data in TDA section the following line must be generated:

```
.osAppl_<ApplName>_tdata    align(4) MAX_SIZE(0x0100) :>.
```

File ostdata.dld ends with the mapping for global shared data in TDA section which must look like:

```
.osGlobalShared_tdata    align(4) MAX_SIZE(0x0100) :>.
```

## 11.4 Stack Size Configuration

The size of task stacks, ISR stacks and the system stack is configured by the user. The application code must not use more stack than configured. Before trusted or non-trusted application code (tasks, ISRs, trusted and non-trusted functions) is executed the OS always reprograms MPU region 0 in order to protect the stack memory areas.

The following table provides an overview of the stacks and which code parts need to be considered for the analysis of the required stack sizes.

Stack	Usage
System Stack	StartupHook ErrorHook ProtectionHook [SPMF92:0082] ShutdownHook [SPMF92:0081]
Task Stacks	the corresponding task function and its call tree ISRs of category 1 (when interrupting a task) ErrorHook Storing a context (144 Byte)
ISR Stacks	the corresponding category 2 ISR function and its call tree ISRs of category 1 (when interrupting an ISR) ErrorHook Storing a context (144 Byte)

If no static analysis for the stack requirement is made, the stack usage may be measured by means of the API functions `osGetStackUsage`, `osGetISRStackUsage` and `osGetSystemStackUsage`, when `StackUsageMeasurement` is configured. Measurement has to consider the maximum stack usage of the code under measure. It has to be ensured, that all directly and indirectly called functions are executed and use the maximum possible stack.

Stack Usage measurement is implemented by filling the stack with a pattern on startup and counting the number of continuous patterns which have not been overwritten with another value. This may lead to a too small measured value in case the function under measure uses this pattern as value on its stack.

As the hardware allows to enable interrupts even in non-trusted code, any non-trusted ISR may enable nesting. Therefore, the user shall expect that interrupt nesting can always occur when defining the system stack size [SPMF92:0089].

The stack usage must be measured after the maximum call depth has been reached [SPMF92:0090].

## 11.5 Stack Monitoring

The stack memory area is write protected via MPU region 0. Trusted and non-trusted applications and the OS cannot write to stack areas which belong to other applications.

This hardware based stack monitoring does not detect all stack errors [SPMF92:0076]. Stack overflow cannot be detected if the task or ISR stack is mapped immediately after the corresponding application data or global shared section. Stack underflow cannot be detected if the task or ISR stack is mapped immediately before the corresponding application data or global shared section.

On derivatives with MPU inactive in privileged mode all OS API functions are executed on the stack of caller without memory protection. [SPMF92:04.0020{!MPUPRIVMODE}]

## 11.6 Usage of MPU Regions

MPU region 0 is always used for stack area protection. It is always reprogrammed when the context is switched. Therefore MPU region 0 cannot be configured by the user.

Each MPU region 1 ... 11 can be configured for static or dynamic usage:

- If a MPU region is configured for static usage, then it is initialized in StartOS and not changed any more. For static MPU regions the user must specify the access attributes.
- If a MPU region is configured for dynamic usage, then it is initialized in StartOS and not always reprogrammed when the context is switched. The access attributes for dynamic MPU regions are configured by the OS.

All stack sections must be mapped to a consecutive memory area. A static MPU region must be configured for this memory area so that trusted and non-trusted application have only read access to it. That means in supervisor and in user mode only reading is possible. Write access to dedicated stack is achieved at runtime via reprogrammed MPU region 0 when a task or ISR is started.

A static MPU region must be configured for the applications data area so that in supervisor mode read and write is possible and in user mode only reading is possible. Write access to the dedicated application data area is achieved via dynamic MPU region which must be configured for each non-trusted application.

A static MPU region must be configured for the code and const area (i.e. ROM/FLASH) so that in supervisor mode (trusted applications) read, write and execute is possible and in user mode (non-trusted applications) only read and execute is possible in that area.

## 11.7 Usage of Peripheral Interrupt API

The OS provides functions which allow write access to EI level interrupt control registers EICn and to EI level interrupt mask registers IMRn in user mode. Non-trusted applications can enable or disable peripheral interrupt sources by means of this functions. Call of the OS peripheral interrupt API functions must be checked in every application that only valid interrupt sources are modified [SPMF92:04.0016].

## 12 Glossary and Abbreviations

### 12.1 Glossary

Term	Description
OS-Application	An OS-Application is a set of tasks, ISRs and (Non-)Trusted Functions with common peripheral/memory access rights and executing in the same privilege mode.
Trusted software	Trusted application code is code which is executed with supervisor privileges. The user has to ensure that this code does not interfere with other components. Examples are start-up code, global Hook functions, Tasks and ISRs which are configured to be part of a trusted OS Application.
Non-trusted software	Defined by AUTOSAR specification: Part of a non-trusted application. Non-trusted software is running with memory protection enabled and in non-privileged mode. Therefore non-trusted software might be implemented on QM level but ASIL software which does not need unlimited memory access or privileged access can also be configured as non-trusted.
Privileged mode	CPU mode with unlimited access to CPU system registers. Software running in privileged mode is e.g. able to reconfigure the MPU and to enable or disable interrupts. Therefore software running in privileged mode must be implemented on ASIL level.
Non-privileged mode	CPU mode without or with limited access to CPU system registers. Software running in privileged mode is not able to reconfigure the MPU or to enable or disable interrupts. Therefore software running in privileged mode might be implemented on QM level.
Silent principle	The idea of "Silent" code is not to disturb other software by means of unintended memory writes. To provide high performance, this is not achieved by a hardware protection mechanism (which would require MPU reconfiguration for each API call) but by analysis of the OS code.

Table 12-1 Glossary

## 12.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CRC	Cyclic Redundancy Check
ECU	Electronic Control Unit
IRQ	Interrupt Request
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MPU	Memory Protection Unit (realized in hardware by the processor)
MSSV	MICROSAR Safe Silence Verifier
ORTI	OSEK Run Time Interface
OS	Operating System
QM	Quality Management (used for software parts developed following only a standard quality management process)
SC	Scalability Class (of AUTOSAR OS)
SEooC	Safety Element out of Context: a safety-related element which is not developed for a specific item

Table 12-2 Abbreviations

## 13 Contact

Visit our website for more information on

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

**[www.vector.com](http://www.vector.com)**

For support requests and issue notifications write to **[osek-support@vector.com](mailto:osek-support@vector.com)**