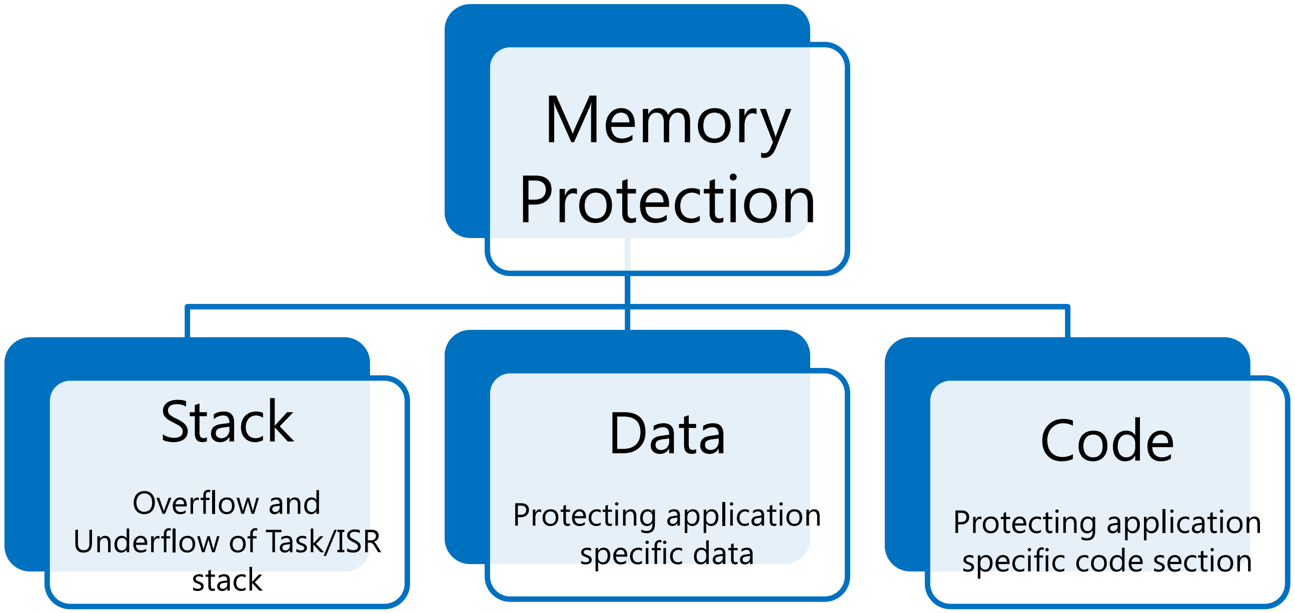
AUTOSAR OS SC3 – Overview and Use Case

KPIT

Scope of the session

1. AUTOSAR OS Memory Protection Features
2. Sample Use Case for Memory Protection Functionality
3. Scalability Class 3(SC3) of AUTOSAR OS and configurations
4. Use of linker and memmap files
5. Memory faults and handling
6. AUTOSAR OS SC4

Memory Protection : Classification


Memory Protection
Stack
Overflow and Underflow of Task/ISR stack
Data
Protecting application specific data
Code
Protecting application specific code section

Features

- ❑ Memory protection is achieved by assigning read, write and execute properties to a memory area and by defining the access rights to an object.

- ❑ The AUTOSAR operating system shall protect data based on the access rights configured

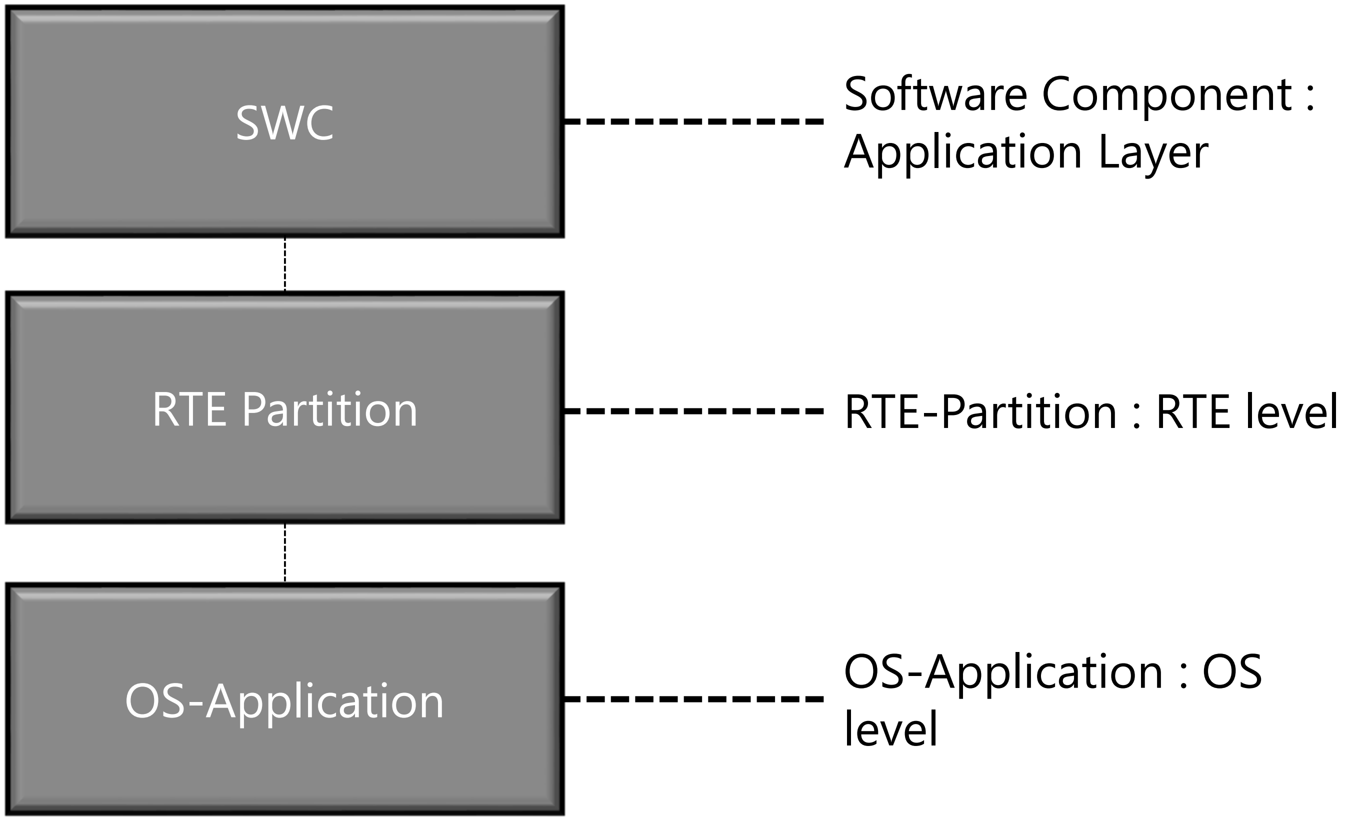
- ❑ Provides a protection against the illegal/unintended access of the memory area

- ❑ To achieve such memory protection Operating System needs hardware support

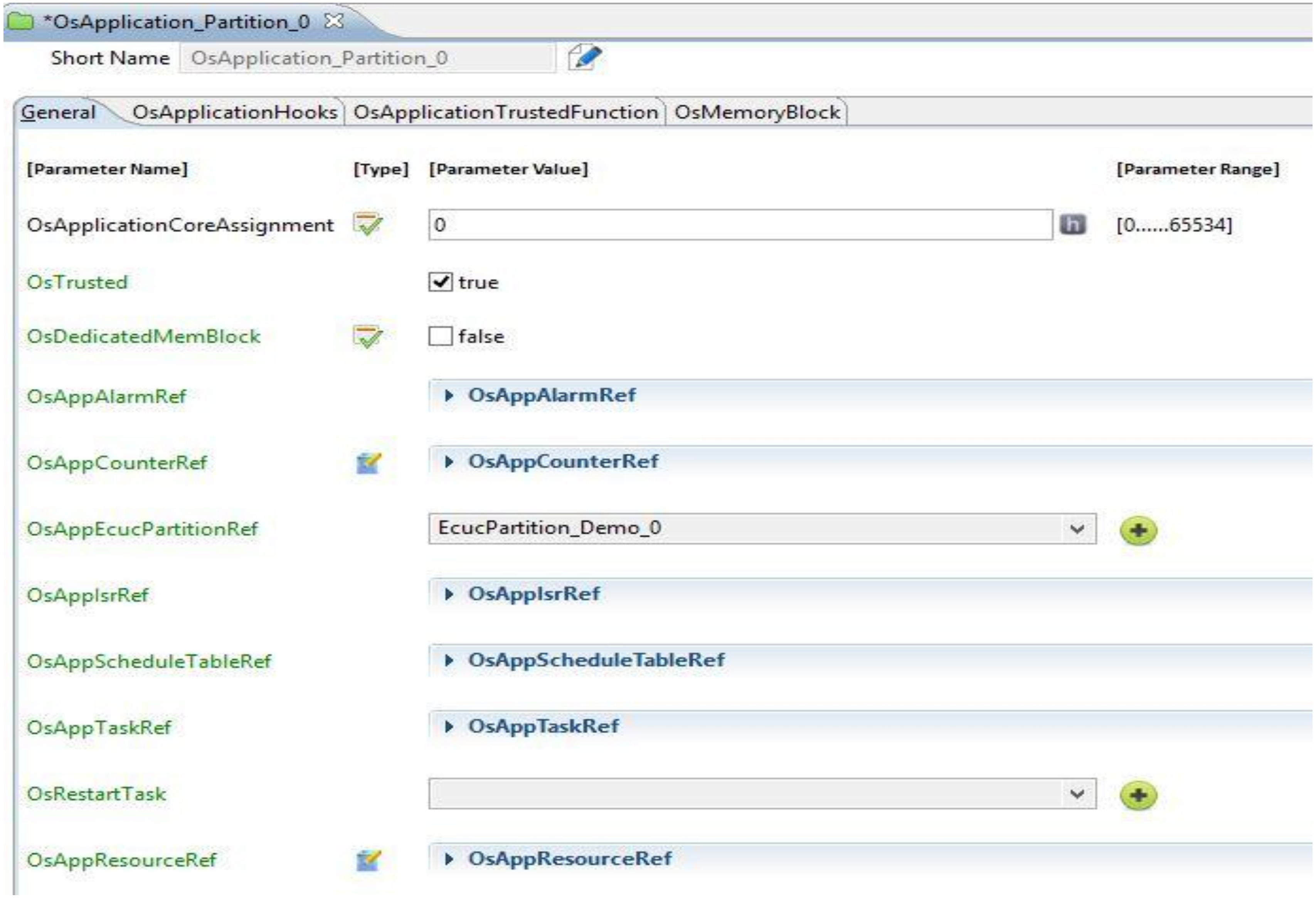
The Concept of Os Application

- ❑ Os Applications are the representative of a partition at system level. They are also called as “Locatable Entities”
- ❑ Os Application can have various Os objects like Tasks, ISRs, Alarms etc.
- ❑ An Os object (like Task) shall belong to a single Os Application but multiple Os Applications may access this Task
- ❑ Based on the access rights that are given to an application it is further classified
- ❑ Os Applications are statically configured for all the possible memory or service requests

Hierarchy at system level


SWC
RTE Partition
OS-Application
Software Component : Application Layer
RTE-Partition : RTE level
OS-Application : OS level

OS-Application configuration



*OsApplication_Partition_0

Short Name: OsApplication_Partition_0

General | OsApplicationHooks | OsApplicationTrustedFunction | OsMemoryBlock

[Parameter Name]	[Type]	[Parameter Value]	[Parameter Range]
OsApplicationCoreAssignment	✓	0	[0......65534]
OsTrusted		✓ true	
OsDedicatedMemBlock	✓	☐ false	
OsAppAlarmRef		▸ OsAppAlarmRef	
OsAppCounterRef		▸ OsAppCounterRef	
OsAppEcucPartitionRef		EcucPartition_Demo_0	
OsAppIsrRef		▸ OsAppIsrRef	
OsAppScheduleTableRef		▸ OsAppScheduleTableRef	
OsAppTaskRef		▸ OsAppTaskRef	
OsRestartTask			
OsAppResourceRef		▸ OsAppResourceRef	

OS-Application configuration details

- **OsApplicationCoreAssignment**: The partition can be mapped to required core using this parameter.
- **OsTrusted**: This parameter shall decide whether a partition is trusted or non trusted.
- **OsDedicatedMemBlock**: Parameter to reserve memory blocks for the partition
- **OsRestartTask**: Reference to OS-task that needs to be invoked when partition is terminated and restarted.
- OsMemoryBlock is the container that needs to be configured with memory blocks if the Os application/Partition is non trusted.
- The configuration has reference to all OS objects that belong to the specific partition.

Sample Use-Case for Memory Protection

- The application contains two software components, one of which is safety critical and the other is non safety critical.
- Examples for :
    - Safety critical(ASIL D) – Reads sensor information, drives actuator/motor etc.
    - Non safety critical(QM software) – perform some calculations etc

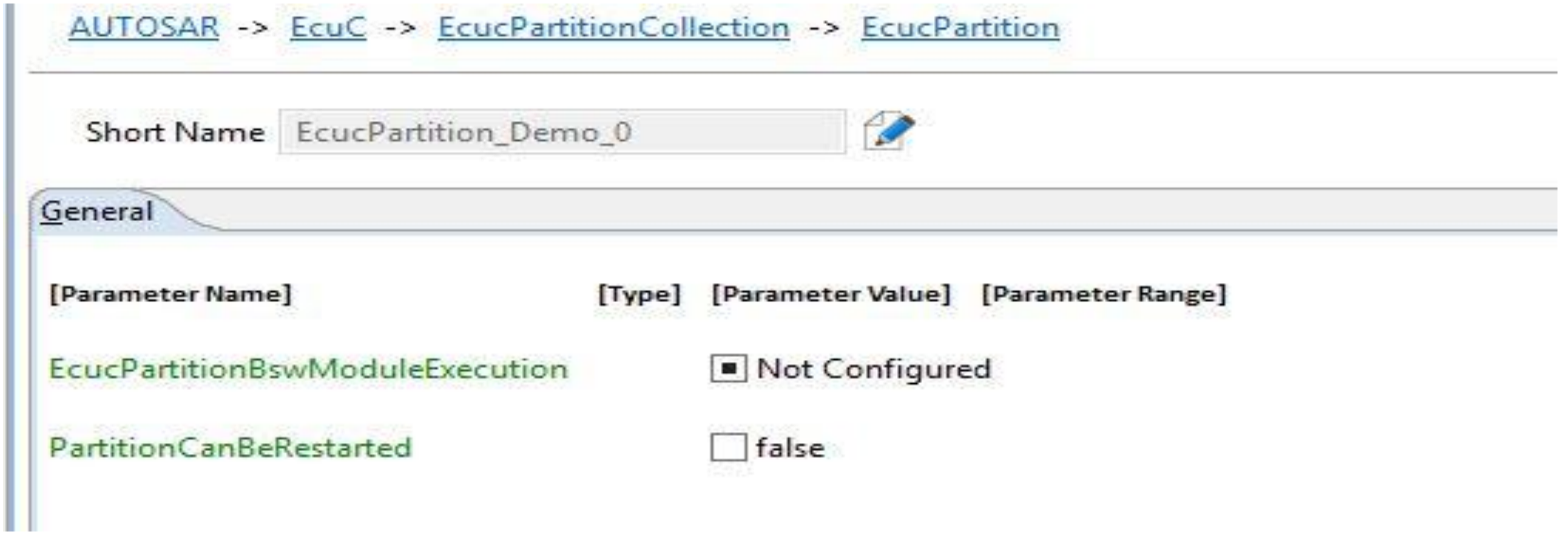
APPLICATION CONSTRAINT: The safety critical SWC needs to be isolated from the non safety critical SWC to ensure ‘freedom from interference’.

SAFETY VIEW : It is necessary to ensure the non safety critical SWC does not interfere with safety critical SWC.

AUTOSAR Solution : Use memory protection feature and make non safety critical SWC as non trusted partition to restrict its access

Configuration for the scenario

- OS needs to be configured in SC3 mode to enable memory protection functionalities
- Partition needs to be created in EcuC module and Os-Application needs to be configured
- Safety critical partition/Os application shall be configured as trusted while Non safety critical software shall be configured as non trusted
- Memory blocks needs to be configured explicitly for non trusted partition and the appropriately MemMap and linker need to be edited.



AUTOSAR -> EcuC -> EcucPartitionCollection -> EcucPartition

Short Name: EcucPartition_Demo_0

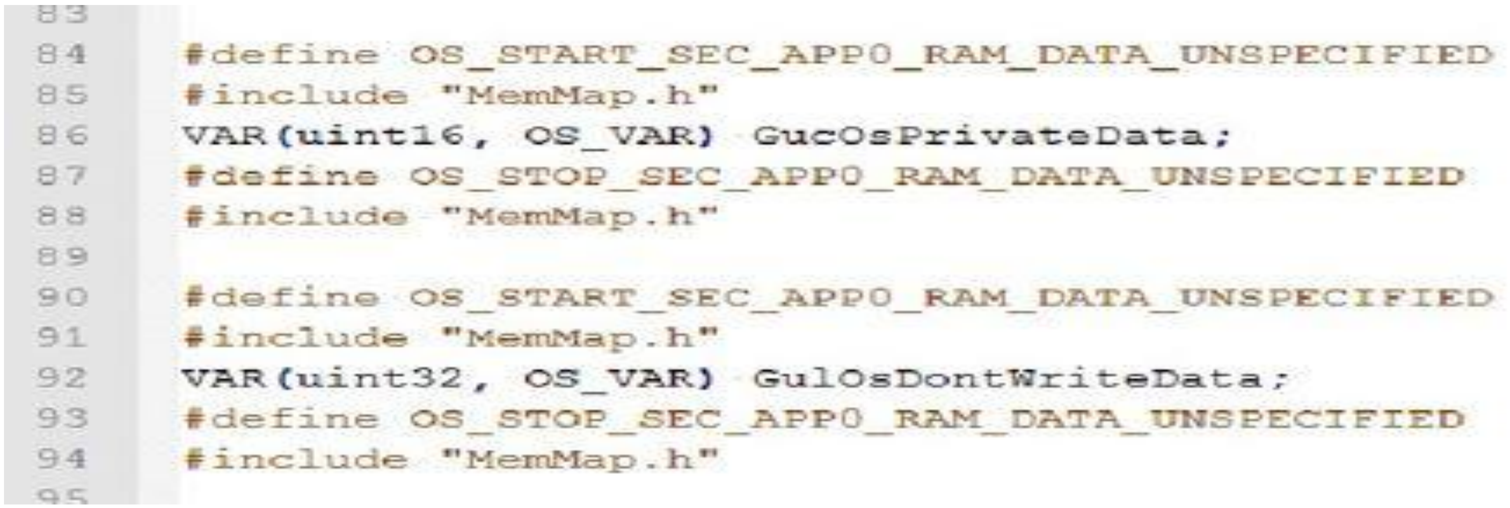
General

[Parameter Name]	[Type]	[Parameter Value]	[Parameter Range]
EcucPartitionBswModuleExecution		▣ Not Configured	
PartitionCanBeRestarted		☐ false	

Usage of MemMap and Linker Script to configure memory block

- Following are the steps to configure Memory Block:
  - Encapsulate the code / data in appropriate section (.c files)
  - Create an entry for the section in Memory map with appropriate linker symbol (Memmap file)
  - Assign the linker symbol in appropriate group (Linker file)

Configuration of Memory Block : Encapsulation

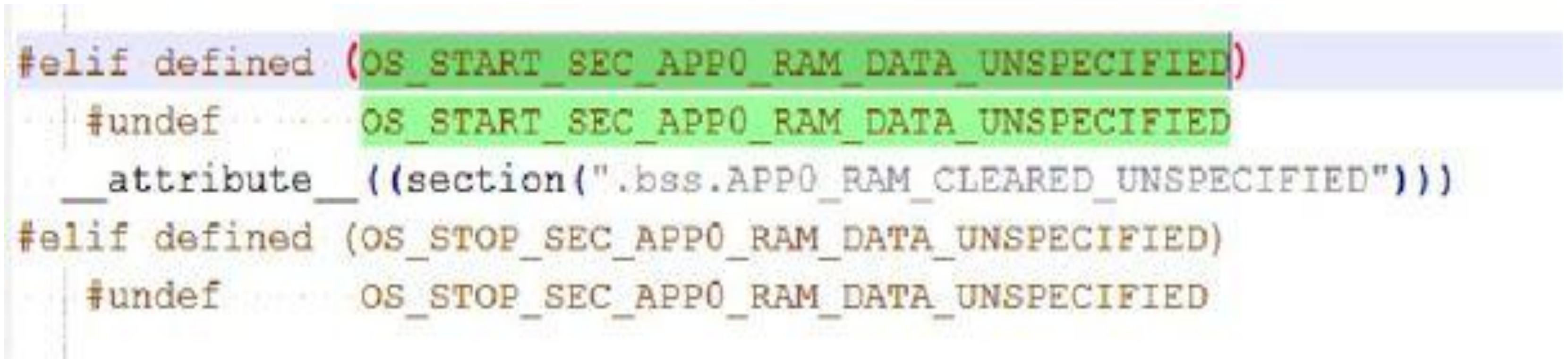


```
#define OS_START_SEC_APP0_RAM_DATA_UNSPECIFIED
#include "MemMap.h"
VAR(uint16, OS_VAR) GucOsPrivateData;
#define OS_STOP_SEC_APP0_RAM_DATA_UNSPECIFIED
#include "MemMap.h"

#define OS_START_SEC_APP0_RAM_DATA_UNSPECIFIED
#include "MemMap.h"
VAR(uint32, OS_VAR) GulOsDontWriteData;
#define OS_STOP_SEC_APP0_RAM_DATA_UNSPECIFIED
#include "MemMap.h"
```

- Provide appropriate start and end sections for the data / code
- These declarations are important to push a variable into particular data region in the memory map

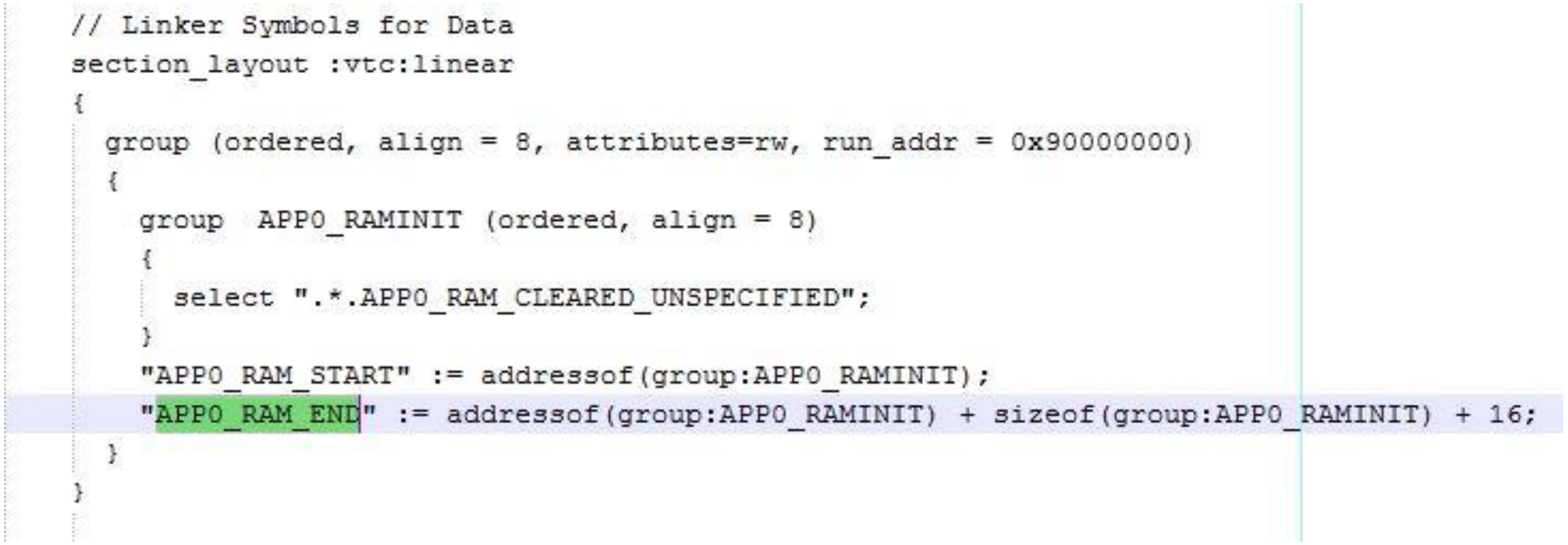
Configuration of Memory Block : MemMap entry



```
#elif defined (OS_START_SEC_APP0_RAM_DATA_UNSPECIFIED)
    #undef      OS_START_SEC_APP0_RAM_DATA_UNSPECIFIED
  __attribute__((section(".bss.APP0_RAM_CLEARED_UNSPECIFIED")))
#elif defined (OS_STOP_SEC_APP0_RAM_DATA_UNSPECIFIED)
    #undef      OS_STOP_SEC_APP0_RAM_DATA_UNSPECIFIED
```

- Place the code/data section in appropriate linker symbol
- These examples are compiler specific and shown examples are for TASKING compiler

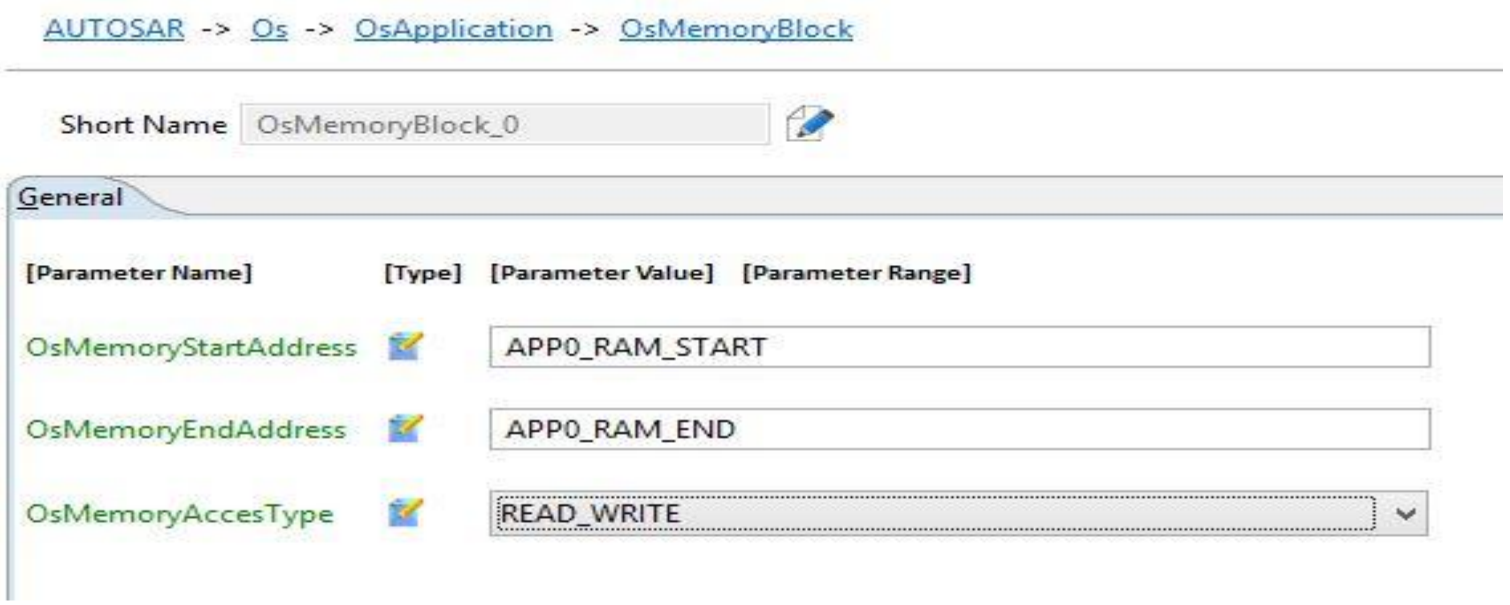
Configuration of Memory blocks : Linker File Update



```
// Linker Symbols for Data
section_layout :vtc:linear
{
  group (ordered, align = 8, attributes=rw, run_addr = 0x90000000)
  {
    group  APP0_RAMINIT (ordered, align = 8)
    {
      select ".*.APP0_RAM_CLEARED_UNSPECIFIED";
    }
    "APP0_RAM_START" := addressof(group:APP0_RAMINIT);
    "APP0_RAM_END" := addressof(group:APP0_RAMINIT) + sizeof(group:APP0_RAMINIT) + 16;
  }
}
```

- Provide appropriate section in the linker
- These examples are compiler specific and shown examples are for TASKING compiler

Configuration of Memory Block


AUTOSAR -> Os -> OsApplication -> OsMemoryBlock

Short Name: OsMemoryBlock_0

General

[Parameter Name]	[Type]	[Parameter Value]	[Parameter Range]
OsMemoryStartAddress		APP0_RAM_START	
OsMemoryEndAddress		APP0_RAM_END	
OsMemoryAccesType		READ_WRITE	

- For the non trusted application/partition, memory block needs to be configured
- Type of blocks : Data (Read/Read & Write permission), Code (Execute permission)

Memory Faults

- Whenever a non trusted application tries to access a variable that is not configured in its assigned memory block, the OS shall trigger a memory fault.
- This leads to invocation of Protection Hook and based on return value from protection hook, OS shall take appropriate action.
- Permissible return values from protection hook in case of memory fault:
    - PRO_TERMINATEAPPL
    - PRO_TERMINATEAPPL_RESTART
    - PRO_SHUTDOWN

AUTOSAR OS SC4

- SC2 configurations allow usage of timing protection features of OS.
- SC3 configurations allow usage of memory protection features of OS.
- In order to use both the feature of OS, the OS needs to be configured to scalability class 4.
- AUTOSAR recommends usage of these features to achieve maximum safety integrity levels.

Questions

Thank you

www.kpit.com