ANCIT

TECHNICAL WHITEPAPER

AUTOSAR OS: A Complete Guide to Runnable-to-Task Mapping

A comprehensive guide for automotive software engineers

Written by

Dr. Kaarthick Balakrishnan

ANCIT | COIMBATORE, INDIA

In this whitepaper, you will understand the fundamentals of AUTOSAR OS task mapping — from its architecture and task models to the step-by-step process of configuring Runnables in OS Tasks. Whether you're new to AUTOSAR or looking to refine your configuration skills, this guide covers everything you need to know.

What is AUTOSAR OS?

AUTOSAR OS evolved from the well-established **AUTOSAR OS (OSEK)** standard. While the underlying philosophy remains largely the same, AUTOSAR OS introduces critical enhancements for modern automotive software — including multicore support, memory protection, and timing protection.

Two foundational principles carried forward from OSEK are:

- **Standardised Interfaces:** Application components interact with the OS and communication stack through standardised interfaces, regardless of the underlying processor.
- **Portability of Application Software:** These standardised interfaces enable true application software portability across different hardware platforms.

Architecture of AUTOSAR OS

AUTOSAR OS defines three processing levels that govern how code executes on the microcontroller:

- **Interrupt Level**
  - **ISR Category 1 (ISR1):** Interrupts managed without OS intervention. Callbacks can be written directly in the Complex Device Driver (CDD).
  - **ISR Category 2 (ISR2):** Interrupt service routines fully managed by the operating system.
- **Logical Level for Scheduler:** Includes scheduling tables and alarm-based activations.
- **Task Level:** Encompasses Basic Tasks and Extended Tasks — the core execution units of the OS.

Key Differences: OSEK OS vs. AUTOSAR OS

Feature	OSEK OS	AUTOSAR OS
Standard	OSEK/VDX	AUTOSAR
Architecture	Standalone RTOS	Part of AUTOSAR BSW
Compatibility	Original standard	Backward compatible with OSEK
Configuration	Static	Static (via ARXML tools)
Multicore Support	✗ No	✓ Yes
Memory Protection	✗ No	✓ Yes
Timing Protection	✗ No	✓ Yes

Feature	OSEK OS	AUTOSAR OS
Integration	Limited	Fully integrated with AUTOSAR stack
Safety (ISO 26262)	Limited support	Strong support

AUTOSAR OS Task Mapping: A Step-by-Step Guide

AUTOSAR OS task mapping is one of the most critical configuration steps in automotive software development. Here is the complete process:

Step 1: Create a Task

Define an OS Task in your AUTOSAR configuration tool. This task will serve as the container for one or more Runnables.

Step 2: Choose Basic Task or Extended Task

Basic Task — Three-State Model

Basic Tasks operate with three states: **SUSPENDED**, **READY**, and **RUNNING**. They cannot block inside the task body.

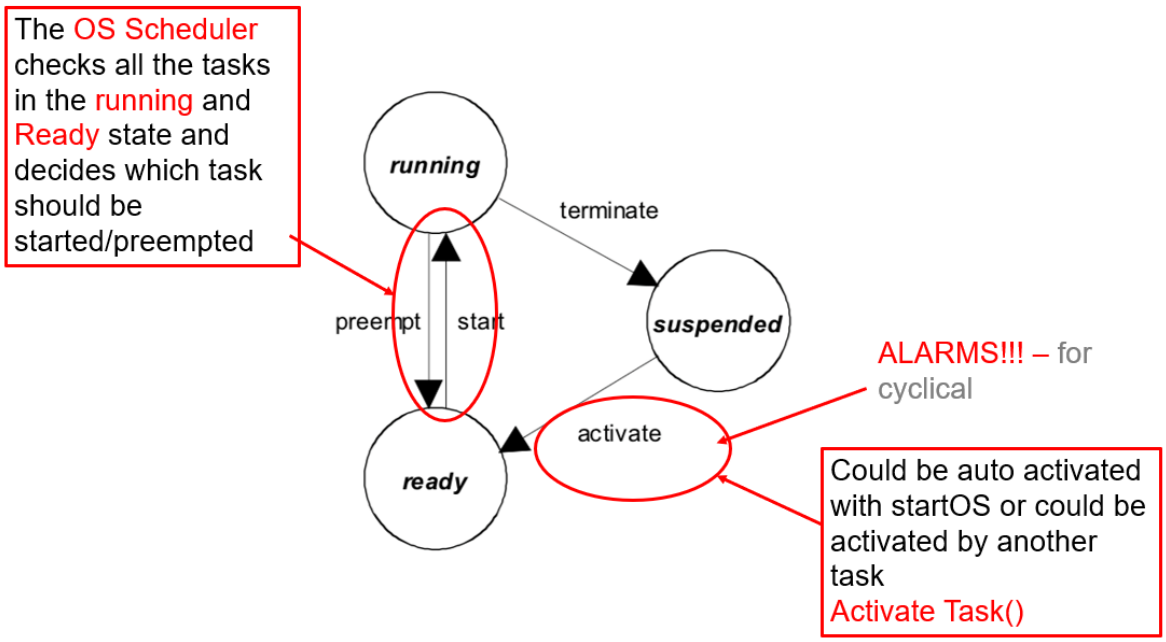

Figure 1: State Model of a Basic Task

Transition	Condition
SUSPENDED → READY	ActivateTask() is called — by an alarm, an ISR, or another task.
READY → RUNNING	The scheduler selects this task as the highest-priority ready task and dispatches it.

Transition	Condition
RUNNING → READY	A higher-priority task becomes ready and preempts this task (full preemptive mode only).
RUNNING → SUSPENDED	The task calls TerminateTask() or ChainTask() at the end of its body.

KEY RULE: A Basic Task must always end with TerminateTask() or ChainTask(). Falling off the end of the task function without calling one of these results in an OS error (E_OS_MISSINGEND).

Extended Task — Four-State Model

Extended Tasks add the **WAITING** state. The task can call WaitEvent(EventMask) inside its body, which suspends execution until another entity sets the awaited event.

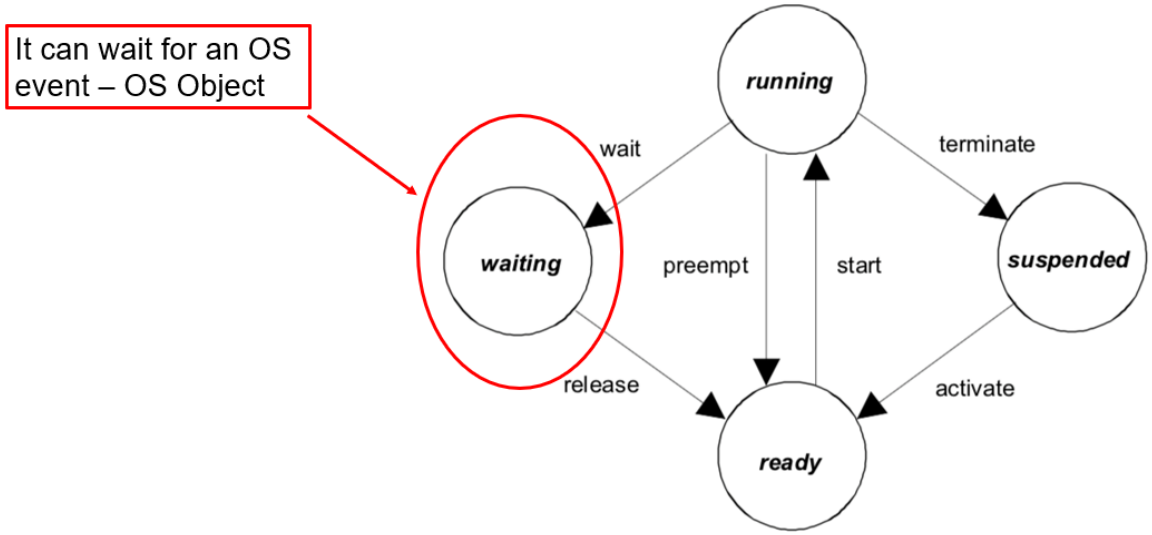

Figure 2: State Model of an Extended Task

Transition	Condition
SUSPENDED → READY	ActivateTask() moves the task into the READY queue.
READY → RUNNING	The scheduler dispatches the task.
RUNNING → WAITING	Task calls WaitEvent(). OS saves the full context, moves the task to WAITING, and dispatches the next ready task.
WAITING → READY	Another entity calls SetEvent() for the awaited event mask.
RUNNING → SUSPENDED	Task calls TerminateTask() after processing.

> *Think of a **Basic Task** as a delivery person — they come only when called, deliver, and leave. An **Extended Task** is like a security guard — always present, sitting quietly, waiting for something to happen. When an event occurs, they act and then go back to waiting.*

Step 3: Assign Task Priorities

Task priorities control the execution order and ensure time-critical operations run before less critical ones.

- **Higher numerical value** → **Higher priority** (in most AUTOSAR implementations)
- **Lower numerical value** → **Lower priority**

AUTOSAR OS uses **priority-based preemptive scheduling** — the scheduler always runs the highest-priority READY task.

Step 4: Choose Preemptive or Non-Preemptive

- **Preemptive Tasks:** Can be interrupted by higher-priority tasks. Suitable for time-critical operations.
- **Non-Preemptive Tasks:** Cannot be interrupted once started. Best used when data consistency is critical.

Step 5: Enable AutoStart

Tasks with **AutoStart=TRUE** are activated at system startup. A Basic Task starts in SUSPENDED state and gets activated cyclically by an alarm, while an Extended Task starts once at startup and never terminates.

Step 6: Map Runnables to the OS Task

Assign your Runnables to the appropriate task based on their trigger type, criticality, and timing requirements.

Step 7: Set Runnable Execution Order

The **position** of Runnables within a task determines their execution order. For example, if Runnable A is at position 1, Runnable B at position 2, and Runnable C at position 3 — they will execute in that exact sequence.

10 Best Practices for AUTOSAR OS Task Mapping

01	Avoid mixing different triggers for Runnables mapped under a Basic Task.
02	Use AutoStart option for only Extended Tasks — Basic Tasks should be alarm-activated.
03	Prefer Basic Tasks wherever possible — they are simpler and more predictable.
04	Map Category 2 Runnables to Extended Tasks and other Runnables to Basic Tasks.
05	Higher cycle frequency = Higher task priority. Fast Runnables need higher-priority tasks.
06	Higher priority tasks should have shorter execution times to avoid blocking the system.
07	Use ISR Category 2 for most interrupts. Avoid ISR1 unless absolutely necessary.
08	Distribute Runnables across multiple tasks. Packing too many long Runnables into few tasks makes scheduling infeasible.
09	Use schedule points in non-preemptive tasks. Configure points where the task checks for higher-priority tasks waiting to execute.
10	Synchronous Server Runnables do not need OS Task mapping. They are called directly by the RTE.

*This whitepaper is authored by **Dr. Kaarthick Balakrishnan** and published by **ANCIT** — a leading automotive engineering training company specialising in AUTOSAR, Embedded Systems, Functional Safety, and Software-Defined Vehicles. For training enquiries, contact us at **beeshma@ancitconsulting.com** or call **+91 9840378602**.*