# 1. Acronym

|  |  |
| --- | --- |
| IPC | Inter Process Communication |
| RTOS | Real-Time Operating System |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# 2. Glossary

|  |  |
| --- | --- |
| Task | Task is a sequential computational unit that is bounded within in a C function. |
| Periodic task | A periodic task is a task executed in a periodic fashion. The period is statically defined. |
| Kernel | Mechanisms of the operating system, the bootcode and the IPC functions are included within the kernel module. |
| Context switch | Context switch involves the dumping of file registers onto the current task stack and the loading of a different task register values. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# 3. Introduction

This document contains the requirements of a real time operating system (RTOS), identified with the codename *No Name Operating System* (N2OS or N2OS), that is meant to be employed in railway applications. With reference to the CENELEC EN 50128 document, this document is part of the phase 4, that is “Requirements Specification”.

The aim of this document is collecting functional and non-functional software requirements of a RTOS kernel, together with indication of the target architecture and the high-level interface to the user applications.

The remainder of the document is structured as listed below:

Section 4 specifies linked standards, specifications and other related documents

Section 5 lists software requirements

Section 6 lists test requirements

## 3.1 Document Conventions

Please, note that each requirement must be identified by a unique and meaningful tag. It can be composed by alphanumeric characters and the solely admitted character separator is the underscore character, namely “\_”.

In particular, it must be formatted as follows:

Name of the document, omitting the version: SRS

Name of the project, omitting the version: N2OS

Requirement category: {GEN, INT, FUN, NSD, SNF, SYS, …}

A sequential 4-digit number, starting from 0001

For instance, SRS\_N2OS\_GEN\_0001 indicates a general requirement included in the document SRS univocally identified by the numerical string 0001.

The requirement description must be clearly written, with non-ambiguity and any redundancy, within the text itself and in the whole document. The style must be technical and self-explained, while acronym and new word must be reported in Acronym and Glossary sections. Text must be written in US English grammar.

Whenever a requirement needs to be suppressed, its own tag cannot be reused for other tags. Instead, description of suppressed requirements must be removed and substituted by the text “Intentionally suppressed”, followed by the reason.

# 4. Related Standards and Documents

## 4.1 Standards

|  |  |
| --- | --- |
| **Reference Name** | **Title** |
| CEI EN 50128 | Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. |
| MISRA 2012 | MISRA C:2012 Guidelines for the use of the C language in critical systems |

## 4.2 Specifications

|  |  |
| --- | --- |
| **Reference Name** | **Title** |
| ARM CMSIS-RTOS | Standard API for Real-Time OS Vendors |
| - | - |

## 4.3 Other documents

|  |  |
| --- | --- |
| **Reference Name** | **Title** |
|  |  |
| - | - |

# 5. Software requirements

The N2OS is a real-time kernel, meant to be lightweight, i.e. low memory footprint and low CPU overhead, and with a minimal set of features, that surely include:

* Task management: preemption and task switch, periodic task invocation, stack management
* Inter process communication: FIFO mechanism to implement message passing among tasks
* Fixed-Priority scheduling
* Predictability of whole system
* Microkernel-based architecture: no device driver included into the kernel code
* Compliant to CENELEC EN50128 and MISRA-C coding standard

User tasks shall be written in C and bounded within a C function. Device driver and other low-level function shall be provide within the user code.

5.1 General Requirements

SRS\_N2OS\_GEN\_0001

N2OS kernel shall be written in C. In particular, the code shall be coded under the MISRA-C coding standard guidelines.

SRS\_N2OS\_GEN\_0002

No dynamic memory allocation mechanisms shall be employed by N2OS. The memory shall be declared and allocated before the code compilation.

SRS\_N2OS\_GEN\_0003

User tasks shall be initialized before the scheduler starting.

SRS\_N2OS\_GEN\_0004

N2OS shall support ARM V7-M ISA and STM32 compilation toolchain based on ARM-GNU-gcc.

SRS\_N2OS\_GEN\_0005

N2OS code shall include neither C standard libraries nor any third-party library.

SRS\_N2OS\_GEN\_0006

The user code and the N2OS kernel shall be compiled in a single execution file.

SRS\_N2OS\_GEN\_0007

N2OS shall occupy as less memory as possible. Introduced overhead, such as the time taken by context switch, shall be as short as possible.

SRS\_N2OS\_GEN\_0008

N2OS shall provide inter-process communication to support message passing mechanisms.

SRS\_N2OS\_GEN\_0009

Tasks priorities shall be statically defined and they shall not change at run-time

SRS\_N2OS\_GEN\_0010

N2OS tasks shall be activate periodically in an automatic fashion. Scheduler shall eventually execute it by means of fixed-priority scheduling algorithm.

SRS\_N2OS\_GEN\_0011

N2OS kernel shall handle tasks in a task list structure. The size of the list is statically defined and cannot change at run-time.

SRS\_N2OS\_GEN\_0012

N2OS kernel shall be structured in order to ease porting operations for supporting different architectures, compilers and toolchains.

SRS\_N2OS\_GEN\_0013

N2OS kernel shall be structured to support different scheduler algorithms.

SRS\_N2OS\_GEN\_0014

Tasks of N2OS kernel shall never terminate.

SRS\_N2OS\_GEN\_0014

N2OS kernel shall expose different macros to end-users to properly define task code.

SRS\_N2OS\_GEN\_0014

N2OS kernel shall implement an internal clock for its inherent scheduling mechanisms and for user tasks that needs time reference.

SRS\_N2OS\_GEN\_0015

N2OS kernel shall implement inter-task communication primitives in order to let tasks exchanging messages at runtime.

SRS\_N2OS\_GEN\_0015

N2OS kernel shall adopt a unique identifier for queues that are involved as IPC.

5.2 Interface Requirements

SRS\_N2OS\_INT\_0001

N2OS shall provide at user application a CMSIS-RTOS programming interface. In particular, the user shall be able to: declare, create and initialize tasks, launch the kernel, create record for calling setup functions by employing CMSIS-RTOS library.

SRS\_N2OS\_INT\_0002

N2OS kernel shall provide a set of system call that allows users to interact with kernel functionalities and mechanisms that it offers, in particular:

* Create and initialize a task, with a given priority, periodicity, a stack area and size
* Start the scheduler
* Read and write from a communication channel in order to let tasks communicate each other
* Enter and exit into and from a critical section

SRS\_N2OS\_INT\_0003

N2OS shall provide memory mapping mechanisms and the interruption system for implementing device driver code.

SRS\_N2OS\_INT\_0004

Tasks status of N2OS kernel shall be defined as ready, active, wait and suspend. The semantic of each status shall be defined as follows:

* Ready: the task is ready to be scheduled, hence the scheduler can take it into consideration among other ready tasks.
* Active: the task is currently being executed by the CPU. Just one task at a time can be active.
* Wait: the task requested to be delayed for some clock ticks that have not yet been elapsing. Once the time passed, the task is forced as ready.
* Suspend (or blocked): the task has voluntary performed a yield due to a resource contest, such as an IPC access. Once the contest is resolved, the task is forced to ready status.

SRS\_N2OS\_INT\_0005

N2OS kernel shall support the porting to other architectures and toolchain. In particular, the functions for implementing the loading of the first task, the context switch, the stack initialization, the system tick implementation and the critical sections shall be implemented in a separate C implementation file.

SRS\_N2OS\_INT\_0006

Task handlers of N2OS shall be defined as function pointer in C, with no arguments and returning void.

SRS\_N2OS\_INT\_0007

N2OS shall expose a macro to declare and define a task. The macro shall accept as input parameter:

* The task name
* The stack size
* The task priority
* The task period

The task code shall be wrapped around subsequently braces.

SRS\_N2OS\_INT\_0008

N2OS shall expose macros to declare the period code portion of tasks. Code written before the periodic block shall be executed once, while code written just after the period block shall be dead code and never executed.

SRS\_N2OS\_INT\_0008

N2OS shall expose to tasks an accessory function to obtain the kernel internal clock.

SRS\_N2OS\_INT\_0009

N2OS kernel shall be provide a function to create queues as IPC mechanism. The function shall accept:

* A not-null pointer to the data buffer of the queue
* The size of the queue (greater than 0)

SRS\_N2OS\_INT\_0010

N2OS kernel shall implement accessory function to queues as

* Non-blocking: the task that is trying to perform an operation that cannot be completed shall return from the accessory method by means of a proper error code
* Blocking: the task that is trying to perform an operation that cannot be completed shall be preempted by the N2OS kernel. The task shall be eventually scheduled as ready task.

SRS\_N2OS\_INT\_0011

N2OS kernel shall be provide a function to enqueue a message to a given queue. The function shall accept:

* The unique identifier of the queue
* A non-null pointer to the message to transmit
* The size of the message (maximum allowed message is 4.294.967.296 byte)
* The operation mode (blocking and non-blocking)

The identifier shall point to a proper initialized queue object. The function returns an error code in case of non-blocking mode (busy queue or full queue), otherwise a success code.

SRS\_N2OS\_INT\_0012

N2OS kernel shall be provide a function to dequeue a message from a given queue. The function shall accept:

* The unique identifier of the queue
* A non-null pointer to the buffer of receiving message
* The size of the message (maximum allowed message is 4.294.967.296 byte)
* The operation mode (blocking and non-blocking)

The identifier shall point to a proper initialized queue object. The function returns an error code in case of non-blocking mode (busy queue or empty queue), otherwise a success code.

5.3 Functional Requirements

SRS\_N2OS\_FUN\_0001

N2OS kernel shall provide a default task, named idle task, that is always executed whenever no other tasks are ready. The idle task has the lowest priority of the system

SRS\_N2OS\_FUN\_0002

N2OS kernel shall define the task stack in order to:

1. Support the ARM V7-M architecture
   1. Special-purpose Program Status Registers (xPSR) se to 0x01000000
   2. The PC register set to the user task handler
   3. The LR register set to a termination task, defined by the N2OS kernel
2. Push and pop general registers onto and from it (12 registers)
3. Guarantee at least a space of 64 bytes to allows the context switch

SRS\_N2OS\_FUN\_0003

N2OS kernel shall implement the initialization function for a given user task. In particular, the function shall accept as input:

* A non-null function handler to the task
* A non-null pointer to the bottom of the stack
* The size of the stack, greater than the minimum required stack
* The defined priority, defined accordingly to the minimum and maximum supported priority
* The periodicity, defined in milliseconds, defined in the range 0 - 65.535 ms (0 ms corresponds to aperiodic task)

The function shall return -1 if the task lists is full, -2 if the stack size is less than the minimum required, a positive number that represents the ID that has been assigned to the initialized task.

The function parameters shall be not modified at runtime.

SRS\_N2OS\_FUN\_0004

N2OS kernel shall expose an initialization function that performs a zeroization of the tasks list and creates the idle task.

SRS\_N2OS\_FUN\_0005

N2OS kernel shall expose a function that allows tasks to be suspended until the next period.

SRS\_N2OS\_FUN\_0006

N2OS kernel shall allow user to statically define a maximum number of tasks that can be supported at run time. The maximum number of tasks shall not be modified at runtime.

SRS\_N2OS\_FUN\_0007

The idle task shall have 0 as task ID, minimum priority and shall be declared as aperiodic task.

SRS\_N2OS\_FUN\_0008

N2OS kernel shall schedule tasks exploiting the fixed-priority scheduler algorithm. In particular, each task shall have its own fixed and static priority computed off-line. The scheduler shall pick from ready-task queue the most priority task. Whenever none of tasks is ready, the scheduler shall pick the idle task.

SRS\_N2OS\_FUN\_0009

N2OS kernel scheduler shall be executed periodically. Its handler shall be invoked by a timer interrupt routine, meant to be periodic, statically configured before launch the first task of the kernel. The scheduler activation period shall be a configuration parameter of the N2OS kernel.

SRS\_N2OS\_FUN\_0010

Each time N2OS kernel scheduler shall check if tasks have missed their own deadline. Whenever a deadline miss occurs, a user level handler shall be invoked, reporting the task id, its period, its priority and the amount of exceeding time.

SRS\_N2OS\_FUN\_0011

N2OS kernel shall expose a delay function that accepts a time value in milliseconds. Each time a task invokes such a delay function, the N2OS scheduler suspends the task. Once the specified amount of time has been elapsed, the scheduler shall set the task status to ready. The suspended task shall be eventually executed at least after the specified time, that belongs to the range 1- 4.294.967.296 ms.

SRS\_N2OS\_FUN\_0012

N2OS kernel shall have an internal clock system. The value of the clock shall be 0 at boot and incremented by the scheduler in a monotonic fashion. Each tick shall correspond to a single execution of the scheduler. Overflow shall be managed by a reset operation.

SRS\_N2OS\_FUN\_0013

N2OS kernel shall suspend tasks that performs blocking operations on queues. In details, whenever a task requests for an accessory operation, but the queue is busy over another operation, the task shall be suspended. Once the former blocking operation completes, the suspended task shall be set to ready task. This mechanism shall be implemented in case of receiving operation on an empty queue and on transmitting operation on a full queue as well.

# 5. Software Test Requirements

SRS\_N2OS\_INT\_0011

N2OS kernel shall be provide a function to enqueue a message to a given queue. The function shall accept:

* The unique identifier of the queue
* A non-null pointer to the message to transmit
* The size of the message (maximum allowed message is 4.294.967.296 byte)
* The operation mode (blocking and non-blocking)

The identifier shall point to a proper initialized queue object. The function returns an error code in case of non-blocking mode (busy queue or full queue), otherwise a success code.

SRS\_N2OS\_TEST\_INT\_0011\_0001

|  |  |
| --- | --- |
| Requirement | SRS\_N2OS\_INT\_0011 |
| Aim of the test | Testing the enqueue interface by passing to the function: a valid identifier to a properly initialized queue, a not-null pointer to a message to enqueue, a size in the valid range and selecting a non-blocking mode. The function shall return a success code and properly enqueue the message. |
| Pre-conditions | The queue has enough space to enqueue the passing message. |
| Input parameters | * Queue\_ID: a valid queue ID * Message: a valid non-null pointer * Size: 10 byte * Mode: non-blocking |
| Post conditions | A message of 10 bytes has been enqueued on the queue. The enqueue did not alter the other previously enqueued message. The task did never reach the suspended status. The passing message has not been altered. |
| Test oracle | Output code: SUCCESS |

SRS\_N2OS\_TEST\_INT\_0011\_0002

|  |  |
| --- | --- |
| Requirement | SRS\_N2OS\_INT\_0011 |
| Aim of the test | Testing the enqueue interface by passing to the function: a valid identifier to a properly initialized queue, a not-null pointer to a message to enqueue, a size in the valid range and selecting a blocking mode. The function shall return a success code and properly enqueue the message. |
| Pre-conditions | The queue has enough space to enqueue the passing message. |
| Input parameters | * Queue\_ID: a valid queue ID * Message: a valid non-null pointer * Size: 10 byte * Mode: blocking |
| Post conditions | A message of 10 bytes has been enqueued on the queue. The enqueue did not alter the other previously enqueued message. The task did never reach the suspended status. The passing message has not been altered. |
| Test oracle | Output code: SUCCESS |

SRS\_N2OS\_TEST\_INT\_0011\_0003

|  |  |
| --- | --- |
| Requirement | SRS\_N2OS\_INT\_0011 |
| Aim of the test | Testing the enqueue interface by passing to the function: a valid identifier to a properly initialized queue, a not-null pointer to a message to enqueue, a size in the valid range and selecting a blocking mode. The function shall return a success code and properly enqueue the message. |
| Pre-conditions | The queue has not enough space to enqueue the passing message. |
| Input parameters | * Queue\_ID: a valid queue ID * Message: a valid non-null pointer * Size: 10 byte * Mode: blocking |
| Post conditions | The task did reach the suspended status |
| Test oracle | Output code: - |