RTI Connext Cert

Integration Manual

Version 2.4.15

Contents:

1 Introduction

This document contains the requirements for the Platform Support Library (PSL) and describes the Platform Integrator responsibilities to integrate with the *RTI Connext Cert* (RCC) Platform Independent Library (PIL).

Creating a Data Distribution Service (DDS) application using RCC for a given platform involves three distinct roles:

- DDS Provider
- Platform Integrator
- Application Developer

RTI fulfills the role of the DDS provider by supplying the RCC Platform Independent Library. The RCC PIL contains all of the platform-independent functionality required to support DDS applications. The Application Developer uses the APIs provided by RCC PIL to create applications that utilize DDS. The Platform Integrator uses the interfaces provided by RCC PIL to integrate RCC into their specific target environment.

The Platform Integrator is assumed to have expert-level understanding of the platform where RCC will run. In order to successfully integrate with RCC, the Platform Integrator needs to fully understand the capability of their chosen platform. RCC provides interfaces to integrate with the platform's operating system and selected network stack. Note that the platform chosen by Platform Integrator may not have an operating system as in the case of bare metal systems. Bare metal platforms can be supported by the provided interfaces as long as the Platform Integrator implements the required functionality. This document presents the interface requirements that must be implemented by the Platform Integrator in order to successfully integrate with RCC.

The Application Developer is responsible for creating applications that meet their product requirements. By using the RCC API documentation, the Application Developer can leverage the features of the RCC DDS implementation to create their application.

The figure below shows a layered view of the three aforementioned roles and where each role and element of the system resides in the development of an application.

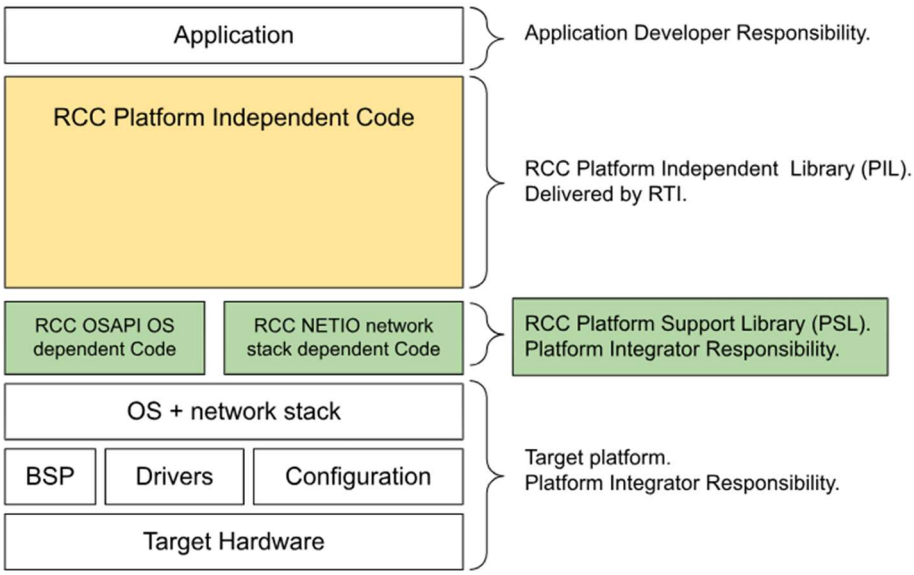


Figure 1-1: Roles and responsibilities to implement RCC

1.1 Terminology

Table 1-1: Acronyms and Abbreviations

Acronym	Description
RCC	*RTI Connext Cert*
OS	Operating System
PSL	Platform Support Library
PIL	Platform Independent Library
QOS	QNX OS for Safety
QTS	Qualification Test Specification **Note:** Dependent on PSL requirements

Table 1-2: Definitions

Phrase	Definition
Platform	A system composed of the language, compiler, operating system, processor, network stack, and supporting hardware.
Target Platform	The platform that will run RCC.
Platform Integrator	End user responsible for platform integration with RCC.

1.1.1 Requirements

All requirement statements shall use the following verbs in the manner defined in this section. Requirements are designed to be contracts between producers and consumers of RTI's software component. The verbs used define the formality of the contract.

Table 1-3: Requirements Terminology

Phrase	Definition
Shall	Mandatory and binding. This is the foundation of a requirement's testable assertions.
Must	Interpreted as "shall". "Must" can be used to define regulations or legal obligations.
Will	Declaration of some future state. These statements are not binding, and are typically used to define limitations of use.
Should	Strongly recommended, but a failure to implement such statements cannot be used as a basis for rejection of the work product. These are not requirement statements and are non-binding.
May	Suggestions or allowances. Statements with the word "may" are not requirements and are non-binding.

1.2 Intended Audience

This document is intended to be used by Platform Integrators.

2 RCC Integration Responsibilities

RCC provides the platform independent implementation of DDS. To integrate with RCC consists of an integration with an Operating System (OS) and network stack. The code that integrates with RCC is referred to as the Platform Support Library (PSL). The remainder of this document describes the Platform Integrator's responsibility to ensure that RCC is integrated correctly with the target platform.

2.1 Global Shared Data Considerations

Special consideration should be given to data that is shared between threads, processes, and processors -- data access protections need to be provided to handle data shared between the entities. This consideration should apply regardless of whether the processors are internal processor cores or external processors.

2.2 Concurrency

All PSL APIs have information about concurrency/thread-safety included in the API reference manuals. It is important that an application does not violate that thread-safety/concurrency guidance. The Platform Integrator may create multiple threads as part of the implementation of the PSL, but from an application point of view, there is only a single critical section protecting all DDS resources within a *DomainParticipant*.

The Platform Integrator is responsible for ensuring that concurrency and re-entrancy requirements for PSL interfaces described in subsequent sections are implemented.

3 PSL OSAPI Software Requirements

3.1 OSAPI Integration (OSAPI)

RCC's OSAPI is an abstract API consisting of a set of functions implemented in the PSL.

The OSAPI is implemented using Operating System dependent APIs, specifically those pertaining to the following areas:

- Primitive data-types
- Memory
- Mutex
- Semaphore
- Strings
- System

3.2 Heap Integration

RCC requires memory to create data structures for managing internal resources and state. RCC defines heap as a region of memory from where an arbitrary number of bytes can be allocated. Memory allocations from the heap need to be thread-safe. The memory allocated by RCC from the heap needs to be readable and writable, but RCC does not require executable memory from the heap. There is no expectation as to where the memory is allocated from. For example, a memory region could be allocated

from a statically defined array or by use of the standard C library malloc function. Heap is defined by RCC as a region of memory that can be allocated at will.

All memory resources allocated by a *DDS_DomainParticipantFactory* and its contained entities need to be managed by a single OS instance. *DomainParticipants* are created by *DomainParticipant* factories. Memory resources that are associated with a *DomainParticipant* have data protection and exclusivity mechanisms that need to be managed by the OS instance that created it.

The Platform Integrator is responsible for implementing the following Heap APIs.

- *OSAPI_Heap_allocate_buffer()*

Data types used by the Heap API:

- *OSAPI_Alignment_T*

OSAPI_Alignment_T is an alias for an *RTI_INT32* type. A parameter of type *OSAPI_Alignment_T* represents the alignment in bytes for the platform. An address is considered aligned when it is a positive integer multiple of the *alignment* parameter. The Platform Integrator provides memory aligned to the CPU bus width if the *alignment* is default, otherwise at least the minimum requested.

R-521.1

The *OSAPI_Heap_allocate_buffer* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Heap_allocate_buffer	out: *char* ***buffer in: *RTI_SIZE_T* size in: *OSAPI_Alignment_T* alignment	*void*

R-521.1.1

The *OSAPI_Heap_allocate_buffer* operation **shall** allocate contiguous memory that RCC is able to read and write and is aligned to *alignment* of size *size*, set the allocated memory region to 0, and return the address of the allocated memory area through the *buffer* pointer variable.

Rationale: *OSAPI_ALIGNMENT_DEFAULT* is an RCC-defined *const*ant that indicates that the requested memory should follow the natural alignment of the architecture (sufficiently aligned to store any C type efficiently).

R-521.1.2

The *OSAPI_Heap_allocate_buffer* operation **shall** set the *buffer* pointer variable to NULL on failure.

Rationale: Execution of the *OSAPI_Heap_allocate_buffer* operation can fail for reasons that are dependent on the target platform. One potential reason that *OSAPI_Heap_allocate_buffer* could fail is due to lack of resources to allocate.

3.3 Mutex Integration

An execution context is a sequence of CPU instructions. An execution context is considered to be non-safe when two contexts can access the same shared resources at the same time, either due to multitasking or due to real parallelism.

RCC uses mutexes to protect access to resources and a*void* race conditions when resources may be shared between different execution contexts.

RCC expects that a mutex can be in one of the following states:

- unlocked - An unlocked mutex does not have an owner.
- locked - A locked mutex has an owner. Only the owner can unlock a mutex.

The Platform Integrator is responsible for implementing the following Mutex APIs.

- *OSAPI_Mutex_new()*
- *OSAPI_Mutex_take()*
- *OSAPI_Mutex_give()*

RCC relies on the Mutex take and give APIs when it needs access to a protected resource or a protected section of code. For protected access, RCC takes the mutex, performs the desired operation, and gives back (releases) the mutex.

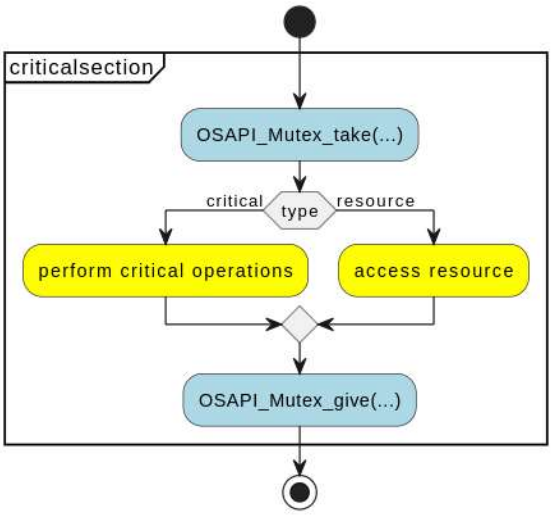


Figure 3-1: Mutex take and give APIs

Data types used by the Mutex API:

- *OSAPI_Mutex_T*

OSAPI_Mutex_T is an alias for the data structure that contains platform specific information used to manage a mutex. The data structure that is to be implemented is called:

- *OSAPI_Mutex*

The composition of the *OSAPI_Mutex* data structure is opaque to RCC.

RCC creates the alias *OSAPI_Mutex_T* from the *OSAPI_Mutex* data structure. RCC does not access the members of the structure directly. The Platform Integrator is responsible for defining the structure to manage the mutex.

R-522.1

The *OSAPI_Mutex_new* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Mutex_new	in: *void*	*OSAPI_Mutex_T**

R-522.1.1

The *OSAPI_Mutex_new* operation **shall** create an unlocked mutex and return a pointer to it on success.

Rationale: The returned address points to the *OSAPI_Mutex_T* object, which has the characteristics of a mutex that guarantees exclusive access when acquired. The mutex is used to protect critical sections of data.

R-522.1.2

The *OSAPI_Mutex_new* operation **shall** return **NULL** on failure.

Rationale: The *OSAPI_Mutex_new* operation is responsible for creating a mutex on the target platform. The creation could fail for numerous reasons. The *OSAPI_Mutex_new* operation should account for the failure mechanisms of the platform.

R-522.2

The *OSAPI_Mutex_take* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Mutex_take	in: *OSAPI_Mutex_T* *mutex	*RTI_BOOL*

R-522.2.1

The *OSAPI_Mutex_take* operation **shall** make the caller of this operation the mutex owner and return *RTI_TRUE* if the mutex was successfully taken **OR** the calling thread is already the owner of the mutex.

Rationale: To be taken, a mutex needs to be either unlocked or locked by the current execution context. A thread can take a mutex that it already owns.

R-522.2.2

The *OSAPI_Mutex_take* operation **shall** return *RTI_FALSE* if the mutex has not been successfully taken.

Rationale: The *OSAPI_Mutex_take* operation will return *RTI_FALSE* when the mutex is invalid or the calling thread cannot be blocked while waiting for the mutex to become unlocked. The *OSAPI_Mutex_take* operation may need to handle other failure mechanisms.

R-522.2.4

The *OSAPI_Mutex_take* operation **shall** lock the mutex if it was successfully taken and was previously unlocked.

Rationale: A locked mutex prevents a shared resource from being acessed by other execution contexts.

R-522.3

The *OSAPI_Mutex_give* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Mutex_give	in: *OSAPI_Mutex_T* *mutex	*RTI_BOOL*

R-522.3.1

The *OSAPI_Mutex_give* operation **shall** return *RTI_TRUE* if the mutex has been successfully given.

R-522.3.2

The *OSAPI_Mutex_give* operation **shall** return *RTI_FALSE* if the mutex has not been successfully given.

Rationale: The *OSAPI_Mutex_give* operation returns *RTI_FALSE* if the mutex is invalid or the mutex cannot be given by the calling execution context. The *OSAPI_Mutex_give* operation may need to handle additional failure mechanisms.

R-522.3.3

The *OSAPI_Mutex_give* operation **shall** unlock the mutex if it was locked and has been successfully given as many times as it was successfully taken in the same execution context.

Rationale: RCC may try to take the mutex many times, even if it is already taken, but it will give it the same amount of times, hence this requirement.

3.4 Semaphore Integration

RCC uses binary semaphores to block executions contexts until an event occurs. The RCC APIs related to semaphores provide the interface for creating, taking (waiting for signal), and giving (signaling) semaphores.

A binary semaphore is only blocked on by at most one execution context when the semaphore is signaled. An execution context that needs to synchronize with an event will attempt to take a semaphore which will cause it to wait for the semaphore to be signaled.

Multiple events can be accumulated until the semaphore is taken.

Once the semaphore is signaled (given) by another execution context, the execution context that was waiting on the semaphore will be allowed to take the semaphore.

When the waiting execution context takes the semaphore, it is allowed to continue.

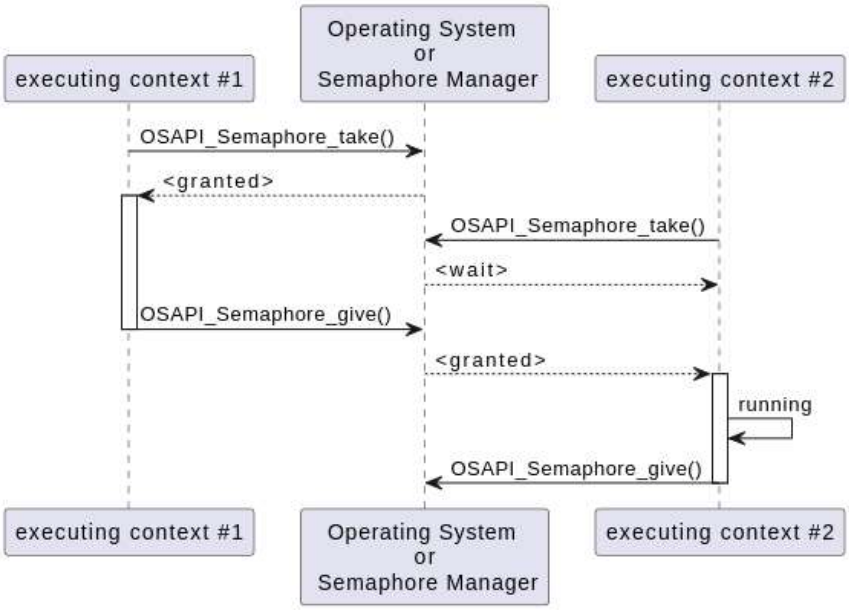


Figure 3-2: Semaphore management

A binary semaphore can be in one of the following two states:

- signaled - A signaled semaphore allows a waiting execution context to take the semaphore and continue execution.
- unsignaled - An unsignaled semaphore causes an execution context to wait when it attempts to take the semaphore.

The Platform Integrator is responsible for implementing the following Semaphore APIs.

- *OSAPI_Semaphore_new()*
- *OSAPI_Semaphore_take()*
- *OSAPI_Semaphore_give()*

Data types used by the Semaphore API:

- *OSAPI_Semaphore_T*

OSAPI_Semaphore_T is an alias for the data structure that contains platform specific information used to manage a semaphore. The data structure that is to be implemented is called:

- *OSAPI_Semaphore*

The composition of the *OSAPI_Semaphore* data structure is opaque to RCC.

RCC creates the alias *OSAPI_Semaphore_T* from the *OSAPI_Semaphore* data structure. RCC does not access the members of the structure directly. The Platform Integrator is responsible for defining the structure to manage the semaphore.

R-523.1

The *OSAPI_Semaphore_new* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_new	in: *void*	*OSAPI_Semaphore_T**

R-523.1.1

The *OSAPI_Semaphore_new* operation **shall** create an unsignaled semaphore and return a pointer to it on success.

R-523.1.2

The *OSAPI_Semaphore_new* operation **shall** return NULL on failure.

Rationale: The *OSAPI_Semaphore_new* operation attempts to create a new semaphore in an unsignaled state. If this operations fails, then the operation will return NULL indicating that a failure has occurred with the creation of the semaphore. Failure mechanisms are specific to the target platform. Examples of possible reasons for failure include the following:

- Lack of permission / privilege to modify or control a resource
- Insufficient resources
- System is busy

Refer to the documentation for the target platform for applicable failures.

R-523.2

The *OSAPI_Semaphore_take* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_take	in: *OSAPI_Semaphore_T* *self in: *RTI_INT32* timeout out: *RTI_INT32* *fail_reason	*RTI_BOOL*

Rationale: *timeout* is in milliseconds. RCC defines the *const*ant *OSAPI_SEMAPHORE_TIMEOUT_INFINITE*.

OSAPI_SEMAPHORE_TIMEOUT_INFINITE is used to indicate that operation does not timeout while waiting to take the semaphore.

R-523.2.1

If the semaphore has been successfully taken, the *OSAPI_Semaphore_take* operation **shall**:

1. return *RTI_TRUE*
2. set the *fail_reason* to *OSAPI_SEMAPHORE_RESULT_OK* if the *fail_reason* pointer is not NULL.

Rationale: *OSAPI_SEMAPHORE_RESULT_OK* is defined by RCC.

R-523.2.2

If the attempt to take a semaphore times out, the *OSAPI_Semaphore_take* operation **shall**:

1. return *RTI_TRUE*
2. set the *fail_reason* to *OSAPI_SEMAPHORE_RESULT_TIMEOUT* if the *fail_reason* pointer is not NULL.

Rationale: *OSAPI_SEMAPHORE_RESULT_TIMEOUT* is a *const*ant defined by RCC.

R-523.2.3

If the semaphore has not been successfully taken due to a failure other than an expired timeout, the *OSAPI_Semaphore_take* operation **shall**:

1. return *RTI_FALSE*
2. set the *fail_reason* to *OSAPI_SEMAPHORE_RESULT_ERROR* if the *fail_reason* pointer is not NULL.

Rationale: *OSAPI_SEMAPHORE_RESULT_ERROR* is a *const*ant defined by RCC.

R-523.3

The *OSAPI_Semaphore_give* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_give	in: *OSAPI_Semaphore_T* *self	*RTI_BOOL*

R-523.3.1

The *OSAPI_Semaphore_give* operation **shall** return *RTI_TRUE* if the semaphore has been successfully given.

R-523.3.2

The *OSAPI_Semaphore_give* operation **shall** return *RTI_FALSE* on failure.

Rationale: The *OSAPI_Semaphore_give* operation can fail. A common failure mechanism is attempting to give a semaphore that is invalid. Other failure mechanisms are dependent on the target platform and implementation.

3.5 Process Integration

RCC performs limited introspection on the running process. It uses information about the running process within the DDS implementation.

The Platform Integrator is responsible for implementing the following Process API:

- *OSAPI_Process_getpid()*

The *OSAPI_Process_getpid* operation returns a value of type *OSAPI_ProcessId*.

Data types used by the Process API:

- *OSAPI_ProcessId*

OSAPI_ProcessId is an alias for an *RTI_UINT64* type. The Platform Integrator is responsible for providing the process ID for the application.

R-524

The *OSAPI_Process_getpid* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Process_getpid	in: *void*	*OSAPI_ProcessId*

R-524.1

The *OSAPI_Process_getpid* operation **shall** return the process ID of the caller.

Rationale: The returned process ID is the ID that identifies the executable environment that holds all the execution contexts, including RCC and the client logic. RCC uses the process ID returned by the *OSAPI_Process_getpid* operation to create a unique ID. Systems that do not implement the process abstraction may return 0. The operation will always succeed.

3.6 Memory Integration

RCC declares interfaces for memory manipulation such as comparison, copying, zeroing, searching, and moving.

The Platform Integrator is responsible for implementing the following Memory manipulation APIs.

- *OSAPI_Memory_copy()*
- *OSAPI_Memory_zero()*
- *OSAPI_Memory_compare()*
- *OSAPI_Memory_fndchr()*

- *OSAPI_Memory_move()*

R-525.1

The *OSAPI_Memory_copy* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Memory_copy	out: *void* *dest in: *const void* *src in: *RTI_SIZE_T* size	*void*

R-525.1.1

The *OSAPI_Memory_copy* operation **shall** copy *size* number of bytes from the *src* memory region to the *dest* memory region where the *src* and *dest* memory regions specified by the caller are valid and do not overlap.

Rationale: It is the caller's responsibility to ensure that the memory regions are valid and do not overlap.

The *OSAPI_Memory_copy* operation does not account for scenarios such where memory regions overlap. The *OSAPI_Memory_move* operation handles cases where *src* and *dest* overlap.

R-525.2

The *OSAPI_Memory_zero* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Memory_zero	out: *void* *mem in: *RTI_SIZE_T* size	*void*

R-525.2.1

The *OSAPI_Memory_zero* operation **shall** set each byte in the memory region [*mem*,*mem* + *size* - 1] to zero.

R-525.3

The *OSAPI_Memory_compare* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Memory_compare	in: *const void* *left in: *const void* *right in: *RTI_SIZE_T* size	*RTI_INT32*

R-525.3.1

The *OSAPI_Memory_compare* operation **shall** perform a lexicographical byte-wise comparison of two memory regions of *size* bytes and return less than 0 if *left* is less than *right* interpreted as a sequence of unsigned 8-bit values.

Rationale: *left* memory region: [*left*,*left* + *size* - 1], *right* memory region: [*right*,*right* + *size* - 1]

R-525.3.2

The *OSAPI_Memory_compare* operation **shall** perform a lexicographical byte-wise comparison of two memory regions of *size* bytes and return 0 if *left* is equal to *right* interpreted as a sequence of unsigned 8-bit values.

Rationale: *left* memory region: [*left*,*left* + *size* - 1], *right* memory region: [*right*,*right* + *size* - 1]

R-525.3.3

The *OSAPI_Memory_compare* operation **shall** perform a lexicographical byte-wise comparison of two memory regions of *size* bytes and return greater than 0 if *left* is greater than *right* interpreted as a sequence of unsigned 8-bit values.

Rationale: *left* memory region: [*left*,*left* + *size* - 1], *right* memory region: [*right*,*right* + *size* - 1]

R-525.4

The *OSAPI_Memory_move* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Memory_move	out: *void* *dest in: *const void* *source in: *RTI_SIZE_T* size	*void*

R-525.4.1

The *OSAPI_Memory_move* operation **shall** copy *size* number of bytes from a valid *source* memory region to a valid *dest* memory region, including *source* and *dest* memory regions that overlap.

Rationale: *source* memory region: [*source*,*source* + *size* - 1], *dest* memory region: [*dest*,*dest* + *size* - 1]

The caller is responsible for ensuring that valid regions are provided for the *source* and *dest* memory regions.

R-525.5

The *OSAPI_Memory_fndchr* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_Memory_fndchr	in: *const void* *s	*void* *

	in: *RTI_INT32* c in: *RTI_SIZE_T* size	

R-525.5.1

The *OSAPI_Memory_fndchr* operation **shall** search for *c* interpreted as a byte starting at location *s* for a maximum of *RTI_SIZE_T size* in length and return a pointer to the first occurrence of *c* if *c* is found.

Rationale: *s* memory region: [*s*,*s* + *size* - 1]

R-525.5.2

The *OSAPI_Memory_fndchr* operation **shall** search for *c* interpreted as a byte starting at location *s* for a maximum of *RTI_SIZE_T size* in length and return NULL if *c* is not found.

3.7 String Integration

RCC declares interfaces for finding the length of strings and making string comparisons. The strings need to be encoded as ASCIIZ (NUL terminated ASCII arrays)

The Platform Integrator is responsible for implementing the following String APIs.

- *OSAPI_String_length()*
- *OSAPI_String_cmp()*
- *OSAPI_String_ncmp()*

R-525.6

The *OSAPI_String_length* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_String_length	in: *const char* *s	*RTI_SIZE_T*

R-525.6.1

The *OSAPI_String_length* operation **shall** return the length of an ASCIIZ string not including the NUL terminator.

R-525.7

The *OSAPI_String_cmp* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_String_cmp	in: *const char* *l in: *const char* *r	*RTI_INT32*

R-525.7.1

The *OSAPI_String_cmp* operation shall lexicographically compare the ASCIIZ strings *l* and *r* and return less than 0 if *l* is lexicographically less than *r* when interpreted as a sequence of unsigned 8-bit values.

Rationale: The comparison ends when a difference is found, or when NUL is reached in any of the input strings, whichever happens first.

R-525.7.2

The *OSAPI_String_cmp* operation **shall** lexicographically compare the ASCIIZ strings *l* and *r* and return 0 if *I* is lexicographically equal to *r* when interpreted as a sequence of unsigned 8-bit values.

Rationale: The comparison ends when a difference is found, or when NUL is reached in any of the input strings, whichever happens first.

R-525.7.3

The *OSAPI_String_cmp* operation **shall** lexicographically compare the ASCIIZ strings *l* and *r* and return greater than 0 if *l* is lexicographically greater than *r* when interpreted as a sequence of unsigned 8-bit values.

Rationale: The comparison ends when a difference is found, or when NUL is reached in any of the input strings, whichever happens first.

R-525.8

The *OSAPI_String_ncmp* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_String_ncmp	in: *const char* *l in: *const char* *r in: *RTI_SIZE_T* num	*RTI_INT32*

R-525.8.1

The *OSAPI_String_ncmp* operation **shall** lexicographically compare up to a maximum of *num* characters in the ASCIIZ strings *l* and *r*, interpreted as a sequence of unsigned 8-bit values, and return less than 0 if *l* is less than *r*.

Rationale: The comparison ends when a difference is found, when num characters have been compared, or when NUL is reached in any of the input strings, whichever happens first.

R-525.8.2

The *OSAPI_String_ncmp* operation **shall** lexicographically compare up to a maximum of *num* characters in the ASCIIZ strings *l* and *r*, interpreted as a sequence of unsigned 8-bit values, and return zero if *l* is equal to *r*.

Rationale: The comparison ends when a difference is found, when num characters have been compared, or when NUL is reached in any of the input strings, whichever happens first.

R-525.8.3

The *OSAPI_String_ncmp* operation **shall** lexicographically compare up to a maximum of *num* characters in the ASCIIZ strings *l* and *r*, interpreted as a sequence of unsigned 8-bit values, and return greater than zero if *l* is greater than *r*.

Rationale: The comparison ends when a difference is found, when num characters have been compared, or when NUL is reached in any of the input strings, whichever happens first.

3.8 Operating System API Integration

The System API consists of functions which are more closely related to the target platform hardware than to the operating system. The System API is implemented as an interface using function pointers which is different from the rest of the OSAPI API. Except for the *OSAPI_System* portion of the OSAPI API, most of the API is implemented by creating concrete function using the OSAPI function prototypes.

RCC provides a default implementation for *OSAPI_System*, but the Platform Integrator may create their own implementation. When the default implementation is used, the Platform Integrator uses either the *OSAPI_System_clock_tick* or *OSAPI_System_clock_tick_from_time* operations provided by RCC to advance timers. Even though RCC provides a generic implementation that the Platform Integrator may use, the Platform Integrator needs to provide implementations for *OSAPI_SystemI.get_time* and *OSAPI_SystemI.get_timer_resolution* operations.

Custom implementations can provide their own means for advancing timers. The default implementation supports up to 8 *DomainParticipants* since a timer is created for each *DomainParticipant*. The maximum number of timers that can be created by the default implementation is 8.

The System interface is defined by the *OSAPI_SystemI* type. The *OSAPI_System_get_native_interface* operation needs to be implemented and is a pure function implementation. RCC invokes the *OSAPI_System_get_native_interface* operation to retrieve the System interface. The Platform Integrator is responsible for setting up the System Interface.

If the Platform Integrator opts to implement their own *OSAPI_System* they are responsible for implementing each function of the *OSAPI_SystemI* structure function pointer members and assign the implemented functions to the *OSAPI_SystemI* appropriately.

This *OSAPI_System_get_native_interface* operation is expected to return the system interface for the target platform.

R-526.0

An instance of the *OSAPI_SystemI* structure **shall** be created and its member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Operation	Parameters	Type
OSAPI_SystemI.get_timer_resolution	in: *void*	*RTI_INT32*

Operation	Parameters	Type
OSAPI_SystemI.get_time	in: *OSAPI_NtpTime* *now	*RTI_BOOL*
OSAPI_SystemI.start_timer	in: *OSAPI_Timer_T* self in: *OSAPI_TimerTickHandlerFunction* tick_handler	*RTI_BOOL*
OSAPI_SystemI.generate_uuid	out: *struct OSAPI_SystemUUID* *uuid_out	*RTI_BOOL*
OSAPI_SystemI.get_hostname	out: *char* **const* hostname	*RTI_BOOL*
OSAPI_SystemI.initialize	in: *void*	*RTI_BOOL*
OSAPI_SystemI.get_ticktime	out: *RTI_INT32* *sec out: *RTI_UINT32* *nanosec	*RTI_BOOL*

R-526.1

The *OSAPI_SystemI.start_timer* method **shall** configure the OSAPI_System instance to start invoking the *tick_handler* method parameter on the self *OSAPI_Timer_T* parameter at the frequency returned by the *OSAPI_SystemI.get_timer_resolution* operation, and return *RTI_TRUE* on success.

Rationale: RCC owns and maintains the *OSAPI_Timer_T* type. The *OSAPI_Timer_T* type is not public and should be handled as an abstract data type. The *OSAPI_TimerTickHandlerFunction* prototype and implementation is defined by RCC; this API should handle it as an abstract data type.

RCC will call the *start_timer* method and supply a *tick_handler* function to the *start_timer* method. The Platform Integrator should not call *start_timer* manually nor implement a *tick_handler* function. Instead, the Platform Integrator needs to store *OSAPISYSTEM_MAX_TIMERS* instances of the *self* and *tick_handler* parameters. The Platform Integrator will use these stored parameters to periodically call the *tick_handler* operation and use the stored *self* instance as the parameter for the tick handler.

RCC provides the *OSAPISYSTEM_MAX_TIMERS* literal. It is used as a limit to the amount of timers RCC will create and use.

The timer resolution returned by the *OSAPI_SystemI.get_timer_resolution* operation is in nanoseconds.

R-526.1.1

The *OSAPI_SystemI.start_timer* operation **shall** not configure the *OSAPI_System* instance to start invoking the *tick_handler* method and return *RTI_FALSE* on failure or if the system has not been initialized.

R-526.1.2

The Platform Integrator **shall** call the *tick_handler* method for each of the *OSAPISYSTEM_MAX_TIMERS* instances at least at the timer resolution rate returned by the *OSAPI_SystemI.get_timer_resolution* operation.

Rationale: The Platform Integrator needs to periodically call the *tick_handler* method. The *tick_handler* is exposed to the Platform Integrator through the call to the *OSAPI_SystemI.start_timer* operation.

The same tick handler method is used for each of the *OSAPISYSTEM_MAX_TIMERS* instances. The Platform Integrator needs to iterate through the instances of the timer and call the *tick_handler* method for each instance.

RCC places no requirements on how the periodic calling is performed. The only requirement is that the tick handlers are called at the rate of the timer resolution or faster.

R-526.2

The *OSAPI_SystemI.get_timer_resolution* operation **shall** return a value greater than 0 (zero) for the resolution of the clock used for the system timer.

Rationale: The timer resolution is in nanoseconds.

R-526.3

The *OSAPI_SystemI.get_timer_resolution* operation **shall** return 0 if the system has not been initialized.

R-526.4

The *OSAPI_SystemI*.get_time operation **shall** return the current system time in NtpTime format through the *now* pointer and return *RTI_TRUE* on success.

Rationale: The *OSAPI_NtpTime* type is defined by RCC. RCC relies on a system clock that monotonically increases.

R-526.5

The *OSAPI_SystemI.get_time* operation **shall** return R*TI_FALSE* on failure.

Rationale: The *OSAPI_SystemI.get_time* operation attempts to retrieve time from a clock implemented on the target platform. The Platform Integrator is responsible for implementing time keeping. The Platform Integrator is responsible for implementing the possible failure mechanisms for the target platform.

Possible failure mechanisms include:

- Invalid system time
- Non-monotonic time

R-526.6

The *OSAPI_SystemI.initialize* operation **shall** initialize the system and return *RTI_TRUE* on success.

Rationale: RCC provides an interface that will be called to initialize the system. RCC does not have any requirements on how the system has to be initialized.

The Platform Integrator could initialize variables and start periodic processes during initializations.

R-526.7

The *OSAPI_SystemI.initialize* operation **shall** return *RTI_FALSE* on failure.

Rationale: The *OSAPI_SystemI.initialize* operation can be used to call initialization sequences specified by the Platform Integrator. The Platform Integrator is responsible for determining whether the initialization was successful.

Possible failure mechanisms could include:

- Incomplete initialization
- Unsuccessful subsystem initialization

R-526.8

The *OSAPI_SystemI.generate_uuid* operation **shall** provide a unique system id through the *uuid_out* pointer and return *RTI_TRUE* on success.

Rationale: The *uuid_out* parameter has the type *struct* OSAPI_SystemUUID. The *struct OSAPI_SystemUUID* declares a 128-bit value.

RCC overwrites the first 2 bytes and the last 4 bytes of this value.

The UUID is treated as an array of 16 octets:

```
MSB:
16|             12|                 8|                 4|             0|
```

Bytes 14-15 are overwritten with the RTI vendor ID.

Bytes 0-3 are overwritten with the DDS Entity ID.

R-526.9

The *OSAPI_SystemI.generate_uuid* operation **shall** return *RTI_FALSE* on failure.

Rationale: The *OSAPI_SystemI.generate_uuid* operation will attempt to provide a UUID as implemented by the Platform Integrator. The Platform Integrator is responsible for determining the possible failure mechanisms.

R-526.10

The *OSAPI_SystemI.get_hostname* operation **shall** return a hostname through the *hostname* pointer of up to *OSAPI_SYSTEM_MAX_HOSTNAME* bytes and return *RTI_TRUE* on success.

Rationale: RCC does not interpret the hostname, it is only sent as informational knowledge during participant discovery.

OSAPI_SYSTEM_MAX_HOSTNAME is defined by RCC.

R-526.11

The *OSAPI_SystemI.get_hostname* operation **shall** return *RTI_FALSE* on failure.

Rationale:

The Platform Integrator is responsible for determining possible failure mechanisms.

Failure mechanisms could include:

- The *hostname* is NULL
- The *hostname* can not be retrieved

R-526.13

The *OSAPI_SystemI.get_ticktime* operation **shall** return the number of seconds (*sec*) and nanoseconds (*nanosec)* in increments of the timer resolution since the system started and return *RTI_TRUE* on success.

R-526.14

The *OSAPI_SystemI.get_ticktime* operation **shall** return *RTI_FALSE* on failure or if the system has not been initialized.

Rationale: The *OSAPI_SystemI.get_ticktime* operation attempts to retrieve the current time in ticks. The Platform Integrator is responsible for determining the possible failure mechanisms.

Possible failure mechanism include:

- System has not been initialized
- NULL pointer for the *sec* parameter
- NULL pointer for the *nanosec* parameter

R-526.15

The *OSAPI_System_get_native_interface* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_System_get_native_interface	out: *OSAPI_SystemI* *intf	*void*

R-526.15.1

The *OSAPI_System_get_native_interface* operation **shall** return a pointer to an *OSAPI_SystemI* object that implements the *OSAPI_SystemI* interface through the *intf* parameter.

R-526.15.2

The *OSAPI_System_get_native_interface* operation **shall** return without updating the parameter *intf* if *intf* is 'nil'.

3.8.1 Global Data

R-526.16

The PSL **shall** initialize the global variable *OSAPI_System_gv_System* to an initialized singleton of type *struct OSAPI_System*.

Rationale: The type *OSAPI_System* is defined by RCC. It contains state information on the instance of the system. The global pointer variable *OSAPI_System_gv_System* is forward declared by RCC. The Platform Integrator is responsible for initializing the global variable *OSAPI_System_gv_System* to a singleton that inherits from *OSAPI_System*.

The *OSAPI_System_gv_System* variable is the mechanism RCC uses to get access to the user defined *OSAPI_System* instance. As C doesn't support object orientation, this mechanism is supported by the implementation of an inheritance method that uses the inherited type as the first member of the derived type. This way the Platform Integrator will be able to share information with RCC by using the same instance of the subclass of an *OSAPI_System*, with the extended attributes in the case of the derived type.

An example of the system state structure is as follows:

```
struct OSAPI_SystemPosix
{
/* base-class */
struct OSAPI_System _parent;

struct OSAPI_SystemTimerHandler timer_handler[OSAPISYSTEM_MAX_TIMERS];
OSAPI_Semaphore_T timed_sem_head;
OSAPI_Semaphore_T *timer_sem;
struct OSAPI_Thread *timer_thread;
clockid_t rt_clock;
struct OSPSL_SystemProperty psl_property;
/*... */
};
```

An example of the user defined structure is:

```
RTI_PRIVATE struct OSAPI_SystemPosix OSAPI_System_g =
{
._parent = OSAPI_System_INITIALIZER,
.psl_property = OSPSL_SystemProperty_INITIALIZER
/*... */
};
```

An example of the assignment to the *OSAPI_System_gv_System* is shown below:

```
struct OSAPI_System *OSAPI_System_gv_System = &OSAPI_System_g._parent;
```

The Platform Integrator is responsible for not accessing the derived singleton using the global variable *OSAPI_System_gv_System* because RCC might update the variable *OSAPI_System_gv_System* to point to an internal singleton defined by RCC.

The Platform Integrator is responsible for not accessing the private base class attributes.

R-526.17

The PSL **shall** have an instance of an *OSAPI_System* object.

Rationale: An *OSAPI_System* object is used to initialize the global pointer variable *OSAPI_System_gv_System* and should endure for the lifetime of the application.

R-526.18

The PSL **shall** initialize its instance of the *OSAPI_System* object with *OSAPI_System_INITIALIZER*.

Rationale: *OSAPI_System_INITIALIZER* is defined by RCC.

3.9 Concurrency

R-527.1

The following PSL operations **shall** be reentrant and thread-safe:

- *OSAPI_Heap_allocate_buffer*
- *OSAPI_Mutex_new*
- *OSAPI_Mutex_take*
- *OSAPI_Mutex_give*
- *OSAPI_Semaphore_new*
- *OSAPI_Semaphore_take*
- *OSAPI_Semaphore_give*
- *OSAPI_Process_getpid*
- *OSAPI_Memory_copy*
- *OSAPI_Memory_zero*
- *OSAPI_Memory_compare*
- *OSAPI_Memory_fndchr*
- *OSAPI_Memory_move*
- *OSAPI_String_length*
- *OSAPI_String_cmp*
- *OSAPI_String_ncmp*
- *OSAPI_SystemI.get_timer_resolution*
- *OSAPI_SystemI.get_time*

- *OSAPI_SystemI.start_timer*
- *OSAPI_SystemI.generate_uuid*
- *OSAPI_SystemI.get_hostname*
- *OSAPI_SystemI.initialize*
- *OSAPI_SystemI.get_ticktime*

Rationale: An operation is thread-safe if it guarantees safe execution by multiple threads at the same time, either multitasking or real parallelism. A function is reentrant if it can be interrupted at any point during its execution and then safely called again before its previous invocations complete execution, for example, a recursive function.

3.10 Debug support

Debug Support Integration

The Platform Integrator is responsible for implementing the following Debug support APIs.

- *OSAPI_Log_get_last_error_code()*
- *OSAPI_Log_set_last_error_code()*

R-528.1

The *OSAPI_Log_get_last_error_code* operation **shall** have the following signature when debugging is enabled:

Operations	Parameters	Type
OSAPI_Log_get_last_error_code	in: *void*	*RTI_INT32*

Rationale: This function is only defined when debugging is enabled for the build. If debugging is disabled, it should not be defined.

R-528.1.1

The *OSAPI_Log_get_last_error_code* operation **shall** return the last error code tracked by the platform.

R-528.1.2

The *OSAPI_Log_get_last_error_code* operation **shall** only be defined when debugging is enabled.

R-528.10

The *OSAPI_Log_set_last_error_code* operation **shall** have the following signature when debugging is enabled:

Operations	Parameters	Type
OSAPI_Log_set_last_error_code	in: *RTI_INT32* err	*void*

Rationale: This function is only defined when debugging is enabled for the build. If debugging is disabled, it should not be defined.

R-528.10.1

The *OSAPI_Log_set_last_error_code* operation **shall** set the last error code tracked by the platform to the value of the parameter *err*.

R-528.10.2

The *OSAPI_Log_set_last_error_code* operation **shall** only be defined when debugging is enabled.

4 PSL NETIO Datagram (DGRAM) Software Requirements

4.1 NETIO Datagram (DGRAM) Transport plugin

RCC provides a generic transport plugin service called the NETIO Datagram (DGRAM) Transport plugin. The DGRAM Transport plugin supports transmission and reception of RTPS messages over a connectionless network link.

It is not required to integrate with NETIO_DGRAM. However, without an integration with NETIO_DGRAM it is only possible to communicate within the same *DomainParticipant*. The NETIO_DGRAM transport plugin provides an interface used to enable support for a network stack. The DGRAM Transport plugin API only provides functionality related to connectionless network links. Functionality related to establishing connections, such as TCP, is not provided.

The interface provided by the DGRAM transport plugin is defined in the *NETIO_DGRAM_InterfaceI* data structure. An implementation for each of the interfaces defined in the *NETIO_DGRAM_InterfaceI* data structure is provided by the Platform Integrator. The Platform Integrator uses the interface to incorporate the network stack of their choosing.

The NETIO_DGRAM consists of an upstream and downstream layer:

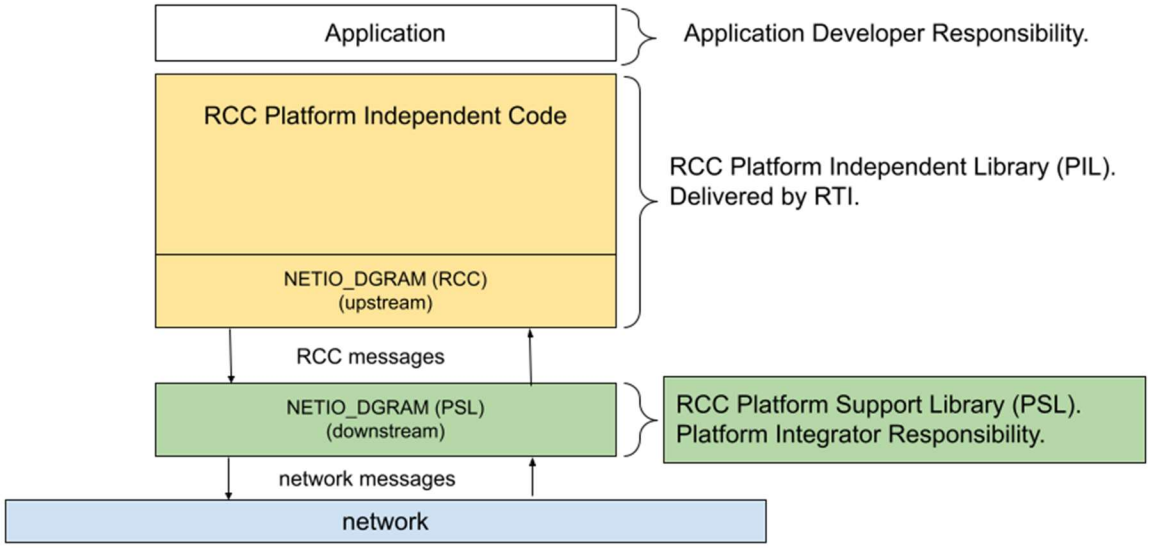


Figure 4-1: NETIO Datagram Layer

An RCC message is an opaque sequence of N 8-bit quantities:

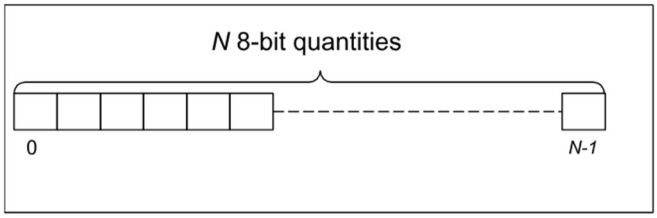


Figure 4-2: RCC message

The structure of the network message is defined by the PSL. Note that N it is allowed but is not required to be a multiple of 2. Any requirements to message length and alignment by the PSL shall be implemented by the PSL.

The upstream layer integrates with RCC and provides management functionality. This layer is provided by RTI as part of the RCC library.

The downstream layer integrates with NETIO_DGRAM and is responsible for sending messages from the upstream layer to the network and receiving messages from the network and forwarding them to the upstream layer.

Note: The current architecture of the RCC NETIO DGRAM does not support reliability when transmitting and receiving network data within the same execution context. The Platform Integrator should a*void* utilizing reliability Quality of Service configurations when testing using a loopback that has calls to transmit and receive operations within the same execution context. A Best Effort Quality of Service may be appropriate.

4.1.1 NETIO_DGRAM Life-Cycle

The figure below shows how the NETIO_DGRAM transport is created and initialized by a *DomainParticipant*.

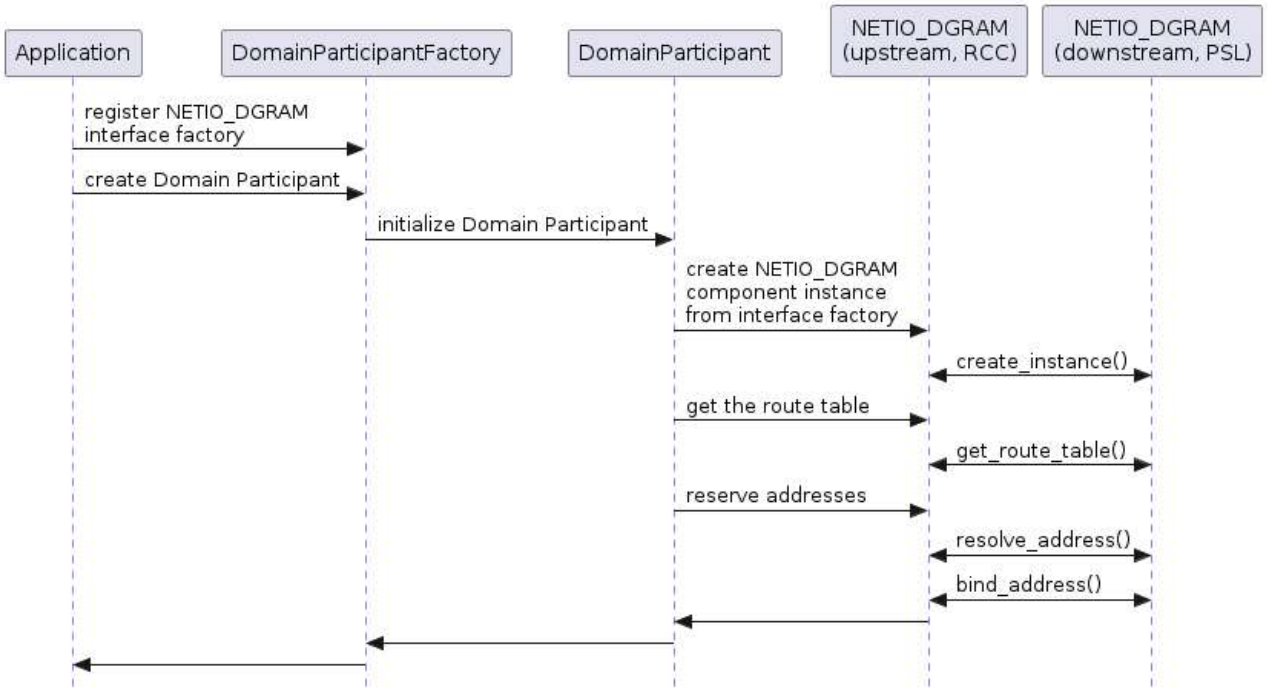


Figure 4-3: NETIO_DGRAM transport creation and initialization

All messages in the sequence from the "Application" through "NETIO_DGRAM (upstream, RCC)" use only descriptions of the calls that are used. The messages between "NETIO_DGRAM (upstream, RCC)" and "NETIO_DGRAM (upstream, PSL)" use the actual attribute names of the NETIO DGRAM Interface.

A *DomainParticipant* creates one instance of each transport that has been enabled in the *DDS_DomainParticipantQos.transports.enabled_transports*.

The figure below shows how RCC messages are sent and received.

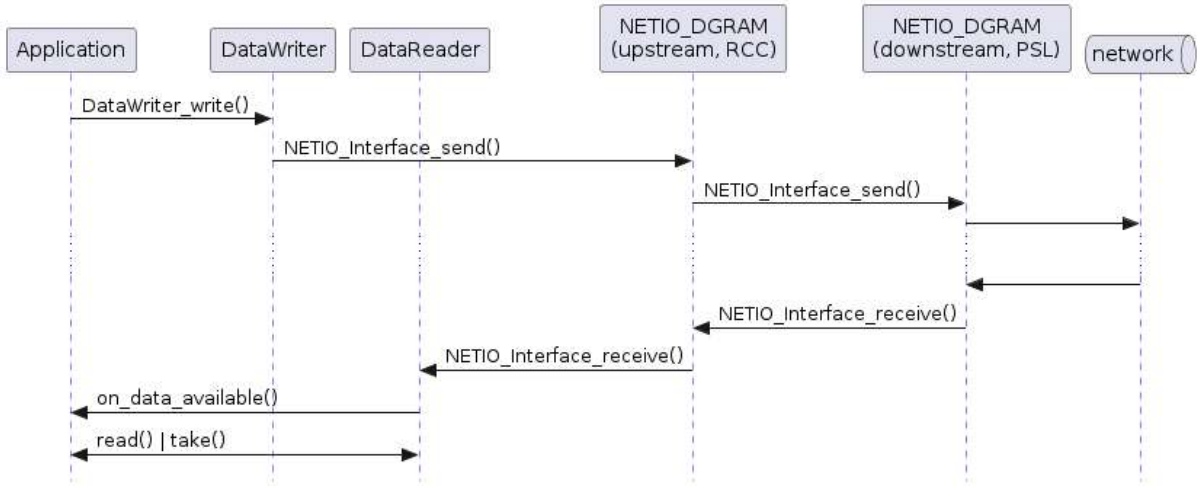


Figure 4-4: Sending and receiving RCC messages

The sequence diagram above shows the actual names of the operations used during the sending and reception of data.

4.1.2 DGRAM Transport Plugin API

The NETIO DGRAM Transport Plugin is defined by the members of the *NETIO_DGRAM_InterfaceI* structure.

The NETIO DGRAM Transport API consists of functions which are used to implement a user specified datagram transport. The NETIO DGRAM Transport API is implemented as an interface using function pointers instead of an implementation using function prototypes.

The NETIO_DGRAM transport interface is defined by the *NETIO_DGRAM_InterfaceI* type. The user's datagram transport will be registered in the application by a call to the *NETIO_DGRAM_InterfaceFactory_register* function which is provided by RCC.

The Platform Integrator creates functions to implement each of the function pointer structure members in the *NETIO_DGRAM_InterfaceI* structure. The Platform Integrator assigns each of the created functions to the appropriate member of the *NETIO_DGRAM_InterfaceI* structure.

R-550.1

To create a NETIO_DGRAM transport plugin implementation, an instance of the *NETIO_DGRAM_InterfaceI* structure **shall** be created and its member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Attribute	Parameters	Type
create_instance	in: *NETIO_Interface_T* *upstream in: *void* *property	*NETIO_Interface_T**

Attribute	Parameters	Type
get_interface_list	in: *NETIO_Interface_T* *user_intf inout: *struct NETIO_DGRAM_InterfaceTableEntrySeq* *if_table	*RTI_BOOL*
release_address	in: *NETIO_Interface_T* *self in: *struct NETIO_Address* *src_addr	*RTI_BOOL*
resolve_address	in: *NETIO_Interface_T* *netio_intf in: *const struct NETIO_DGRAM_InterfaceTableEntry* *if_entry in: *const* char *address_string out: *struct NETIO_Address* *address_value out: *RTI_BOOL* *invalid	*RTI_BOOL*
send	in: *NETIO_Interface_T* *self in: *struct NETIO_Interface* *source in: *struct NETIO_Address* *destination in: *NETIO_Packet_T* *packet	*RTI_BOOL*
get_route_table	in: *NETIO_Interface_T* *netio_intf inout: *struct NETIO_AddressSeq* *address inout: *struct NETIO_NetmaskSeq* *netmask	*RTI_BOOL*
bind_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source	*RTI_BOOL*

Rationale: RCC provides the definition of the *NETIO_DGRAM_InterfaceI* structure. To implement the interface, functions need to be created that can be assigned to the attributes of the structure. The names of the functions created to implement the interface are arbitrary and are at the discretion of the implementer since function pointers are used to assign those functions to the member attributes.

The Platform Integrator is allowed to assign the resolve_address attribute to NULL if desired. If the *resolve_address* attribute is NULL, then the RCC provided implementation for the *resolve_address* operation will be used. If *resolve_address* is NULL then the RCC will resolve addresses in the dotted IPv4 notation. When the RCC provided implementation is used, then the *resolve_address* operation will not convert the IPv4 *address_string* if the *if_entry.locator_kind* is different from *NETIO_ADDRESS_KIND_UDPv4*.

R-550.1.1

The *NETIO_DGRAM_InterfaceI.create_instance* operation **shall** return a non-NULL pointer to an instance of an interface of type *NETIO_Interface_T* on success.

Rationale: The structure member *NETIO_DGRAM_InterfaceI.create_instance* is used to allocate and instantiate any resources needed for the user's desired transport. The created *NETIO_Interface_T* instance will later be used to send packets of type *NETIO_Packet_T*. RCC will try to send packets using the *NETIO_DGRAM_InterfaceI.send operation*. RCC will receive packets when this implementation calls the *NETIO_DGRAM_Interface_upstream_receive* operation with upstream as the *netio_intf* parameter, meaning that upstream has to be saved.

The property parameter carries the information passed to the *NETIO_DGRAM_InterfaceFactory_register* operation as the user properties. The property parameter can be used to provide properties needed by the *NETIO_DGRAM_InterfaceI.create_instance*. The use of the property parameter is at the discretion of the implementer. A pointer to the property structure could have a NULL value.

The user can leverage a derived type from the base type *NETIO_Interface_T* in order to save all the needed resources for the internal operation of the interface (such as the upstream parameter), but will have to return a pointer to a downcasted version of it.

R-550.1.1.1

The PSL **shall** not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *NETIO_DGRAM_InterfaceI.create_instance*.

Rationale: The instance of an interface of type *NETIO_Interface_T* is not fully initialized by the time the *NETIO_DGRAM_InterfaceI.create_instance* is called by RCC.

R-550.1.2

The *NETIO_DGRAM_InterfaceI.create_instance* operation **shall** return NULL on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- Insufficient memory for allocations
- Resources are not available
- Insufficient access permissions
- Improper values for parameters

R-550.1.3

The *NETIO_DGRAM_InterfaceI.get_interface_list* operation **shall** return the available network interfaces to the *if_table* parameter for the NETIO_DGRAM interface passed in *user_intf*, where for each element of the if_table <*entry*> the following criteria are met:

- <*entry.address*> is not within <*entry.multicast_group*> **AND**
- <*entry.multicast_group.netmask.bits*> is not less than 0 and not greater than 128 **AND**
- <*entry.multicast_group.netmask.mask*> all array elements are zeroes **AND**
- <*entry.locator_kind*>
    - is greater than 0 and less than *NETIO_ADDRESS_RTPS_RESERVED_LOW* **OR**
    - is greater than *NETIO_ADDRESS_RTPS_RESERVED_HIGH*

and return *RTI_TRUE* on success.

Rationale: The *NETIO_DGRAM_InterfaceI.get_interface_list* operation will retrieve only available interfaces that meet the criteria and append them to the *if_table* sequence.

RCC provides the definition of *NETIO_ADDRESS_RTPS_RESERVED_LOW* and *NETIO_ADDRESS_RTPS_RESERVED_HIGH*.

The <e*ntry.multicast_group.netmask.bits*> field is of type *RTI_UINT32*.

R-550.1.3.1

The *NETIO_DGRAM_InterfaceI.get_interface_list* operation **shall** return *RTI_TRUE* without updating the if_table sequence If there are no available interfaces.

R-550.1.4

The *NETIO_DGRAM_InterfaceI.get_interface_list* operation **shall** return *RTI_FALSE* on failure.

Rationale: The possible sources of failure for this method are at the discretion of the Platform Integrator.

One possible failure could be insufficient space in the *if_table* to append additional interfaces.
On a failure, the Platform Integrator may elect to update the *if_table* sequence. A failure of the *NETIO_DGRAM_InterfaceI.get_interface_list* operation will result in a failure during initialization.

R-550.1.5

The *NETIO_DGRAM_InterfaceI.release_address* operation **shall** cause the NETIO_DGRAM interface specified by *self* to stop listening to messages from the address *src_addr* and return *RTI_TRUE* on success.

Rationale: When the resources associated with the *src_addr* are no longer needed by the NETIO_DGRAM transport, this function is called by the upstream interface so the resources can be released. When an address is released, no more messages are expected from this address. The resources associated with the *src_addr* were previously created by a successful call to the *NETIO_DGRAM_InterfaceI.bind_address* operation.

R-550.1.6

The *NETIO_DGRAM_InterfaceI.release_address* operation **shall** return *RTI_FALSE* on failure.

R-550.1.7

The *NETIO_DGRAM_InterfaceI.resolve_address* operation **shall** convert an *address_string* that is a valid address for the NETIO_DGRAM interface specified by *netio_intf* to an *address_value* using the locator kind information stored in the interface table *if_entry*, set the value pointed to by *invalid* to *RTI_FALSE*, **AND** return *RTI_TRUE* on successful conversion.

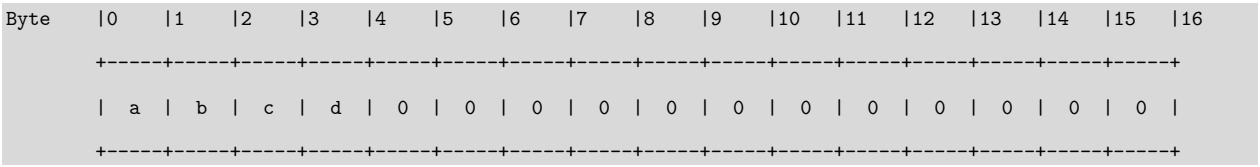
Rationale: An address is valid when it meets the requirements of the user's desired transport.

The *NETIO_DGRAM_InterfaceI.resolve_address* operation is successful only when *address_string* is valid.

Note that RCC ignores the values returned by PSL in the *address_value.port* and *address_value.kind* attributes. RCC overwrites the values assigned to the *address_value.port* and *address_value.kind* attributes during the execution of the *NETIO_DGRAM_InterfaceI.resolve_address* operation.

R-550.1.7.1

If the *locator_kind* of the interface entry *if_entry* is *NETIO_ADDRESS_KIND_UDPv4* and the *address_string* is an IPv4 address, then the *NETIO_DGRAM_InterfaceI.resolve_address* operation **shall** convert the *address_string* in dotted notation "a.b.c.d" to *address_value.value.address.octet* in the following form in network order:



Rationale: IPv4 address means, it is specified in dotted notation "a.b.c.d", where a, b, c, d are numbers not greater than 255.

The *locator_kind* information is stored in the interface table *if_entry.*

When the *locator_kind* is *NETIO_ADDRESS_KIND_UDPv6*, then the format is specified in the RTPS specification.

R-550.1.8

The *NETIO_DGRAM_InterfaceI.resolve_address* operation **shall** return *RTI_FALSE* on failure.

Rationale: Failure modes are determined by the implementer of this method. Examples of failures can include:

1. Invalid address in the *address_string*
2. Conversion failure of a valid address string

R-550.1.8.1

If the *address_string* provided to the NETIO_DGRAM_InterfaceI.resolve_address operation is an invalid address for the NETIO_DGRAM interface specified by *netio_intf*, the *NETIO_DGRAM_InterfaceI.resolve_address* operation **shall** set the value pointed to by *invalid* to *RTI_TRUE*, otherwise set it to *RTI_FALSE*.

Rationale: This requirements describes how the *invalid* parameter is set depending on the failure that occur. This failure situation will return *RTI_FALSE*.

R-550.1.9

The *NETIO_DGRAM_InterfaceI.send* operation **shall** send the *packet* from *source* to the *destination* using the *NETIO_Interface_T* object *self* and return *RTI_TRUE* on success.

Rationale: Sending data from RCC to the DDS databus is accomplished when RCC calls the *NETIO_DGRAM_InterfaceI.send operation*. The *self* parameter is a pointer to a *NETIO_Interface_T* which was previously created during the call to the *NETIO_DGRAM_InterfaceI.create_instance* operation. The operation should not modify the contents of the *NETIO_Packet_T* packet.

R-550.1.10

The *NETIO_DGRAM_InterfaceI.send* operation **shall** return *RTI_FALSE* on failure.

Rationale: The implementation of the *NETIO_DGRAM_InterfaceI.send* operation is at the discretion of the Platform Integrator. RCC uses the return code provided by the operation to handle success or failure. The Platform Integrator determines when and how the *NETIO_DGRAM_InterfaceI.send* operation fails.

R-550.1.11

The *NETIO_DGRAM_InterfaceI.get_route_table* operation **shall** retrieve a sequence of all addresses as *address* and a sequence of all netmasks as *netmask* from the routing table that the NETIO_DGRAM interface can send to and return *RTI_TRUE* on success.

Rationale: The *NETIO_DGRAM_InterfaceI.get_route_table* operation is used to retrieve the current route table associated with the interface *netio_intf* and this information will be used by RCC to determine if the interface is able to send packets to a specific *NETIO_Address* destination. The implementor is responsible for determining and storing the routes applicable for the desired transport. Some platforms may have fixed routes while others may be determined dynamically.

The route table could be stored separately from the *netio_intf* or in conjunction with it—as part of the property *user_property* (optionally provided during the registration of the DGRAM transport; i.e., *NETIO_DGRAM_InterfaceFactory_register* function call) which automatically associates property data with the registered DGRAM transport.

R-550.1.12

The *NETIO_DGRAM_InterfaceI.get_route_table* operation **shall** return *RTI_FALSE* on failure.

R-550.1.13

The *NETIO_DGRAM_InterfaceI.bind_address* operation **shall** cause the interface specified by *user_intf* to listen for messages from the address specified by *source* and return *RTI_TRUE* on success.

Rationale: When an address is bound to an interface, the interface will listen for messages sent from the address indicated from *source*. Before attempting to bind to the *source* address, the Platform Integrator should check that the *source* address is not already bound on the interface provided in *user_intf* according to the requirements of the desired transport.

R-550.1.14

The *NETIO_DGRAM_InterfaceI.bind_address* operation **shall** return *RTI_FALSE* on failure.

R-550.1.15

The Platform Integrator **shall** use the *NETIO_DGRAM_Interface_upstream_receive* operation to forward a *NETIO_Packet_T packet* that contains payload data received from the network and a *NETIO_Address source* to the upstream interface specified by the *netio_intf* parameter upon reception of data from the platform's network stack.

Rationale: The *NETIO_DGRAM_Interface_upstream_receive* operation is provided by RCC.

RCC does not place any requirements on how data is received from the network. The Platform Integrator has complete authority on how it is collected from the network.

The Platform Integrator can use the *NETIO_Packet_set_payload* operation provided by RCC to set the payload in the *NETIO_Packet_T packet*.

4.2 DGRAM Component Registration

Once the Platform Integrator implements the API defined in the *NETIO_DGRAM_InterfaceI struct*, the DGRAM transport plugin is to be registered.

A RCC component called DGRAM is created when the DGRAM transport plugin is registered. To create the DGRAM component, a call to the *NETIO_DGRAM_InterfaceFactory_register* function is made by the application developer. The *NETIO_DGRAM_InterfaceFactory_register* function registers the DGRAM Transport Plugin and links the *NETIO_DGRAM_InterfaceI* implementation.

4.3 Concurrency

R-551.1

The following *NETIO_DGRAM_InterfaceI* operations shall be reentrant and thread-safe:

- *NETIO_DGRAM_InterfaceI.create_instance*
- *NETIO_DGRAM_InterfaceI.get_interface_list*
- *NETIO_DGRAM_InterfaceI.release_address*
- *NETIO_DGRAM_InterfaceI.resolve_address*
- *NETIO_DGRAM_InterfaceI.send*
- *NETIO_DGRAM_InterfaceI.get_route_table*
- *NETIO_DGRAM_InterfaceI.bind_address*

Rationale: An operation is thread-safe if it guarantees safe execution by multiple threads at the same time, either multitasking or real parallelism. A function is reentrant if it can be interrupted at any point

during its execution and then safely called again before its previous invocations complete execution, for example, a recursive function.

5 PSL Data Types Software Requirements

R-560.1

The PSL **shall** access the following RCC defined data types atomically where noted and ensure that the minimum required alignment is met:

Type	Size (bit)	Minimum Required Alignment (in bits)	Atomic access
RTI_INT8	8	8	Yes
RTI_UINT8	8	8	Yes
RTI_INT16	16	16	Yes
RTI_UINT16	16	16	Yes
RTI_INT32	32	32	Yes
RTI_UINT32	32	32	Yes
RTI_INT64	64	32	No
RTI_UINT64	64	32	No
RTI_FLOAT32	32	32	No
RTI_DOUBLE64	64	32	No
RTI_DOUBLE128	128	32	No
RTI_BOOL	32	32	Yes
RTI_SIZE_T	32	32	Yes
OSAPI_Alignment_T	32	32	Yes

Rationale: Atomic access means that access to a variable of a given type is guaranteed to be atomic. This typically means single cycle bus access or a bus arbiter that ensures that updating a primitive type is an atomic bus transaction.

6 PSL Type Plugin Requirements

RCC provides a generic Type service called the Type plugin. The Type plugin support includes the following operations for data types: serialization, deserialization, sample creation and sample deletion.

For RCC to publish and subscribe to topics of user-defined types, the types need to be defined and programmatically registered with RCC. A registered type is then serialized and deserialized by RCC through a pluggable Type interface that the Platform Integrator must implement for each type.

The interface provided by the Type plugin is defined in the *NDDS_Type_Plugin* data structure. An implementation for each of the interfaces defined in the *NDDS_Type_Plugin* data structure is provided by the Platform Integrator.

The Platform Integrator may manually implement each new type, but there is a strong suggestion to use the *rtiddsgen* utility provided by RCC for automatically generating type support code.

The Type Plugin requirements are not obligatory to be implemented as *rtiddsgen* should preferably be used. They are provided in the Integration Manual for completness, in case the user or integrator chooses to implement the plugins without the assistance of *rtiddsgen*. They are also used to verify type support code on the target platform.

R-700.0

To create a Type plugin implementation, an instance of the NDDS_Type_Plugin structure **shall** be created with the following member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Attribute	Parameters	Type
serialize_data	in: *struct CDR_Stream_t* *stream in: *const void* *sample in: *void* *param	*RTI_BOOL*
deserialize_data	in: *struct CDR_Stream_t* *stream out: *void* *sample in: *void* *param	*RTI_BOOL*
serialize_key	in: *struct CDR_Stream_t* *stream in: *const void* *sample in: *void* *param	*RTI_BOOL*
deserialize_key	in: *struct CDR_Stream_t* *stream out: *void* *sample in: *void* *param	*RTI_BOOL*
get_serialized_sample_max_size	in: *struct NDDS_Type_Plugin* *plugin in: *RTI_UINT32* current_alignment in: *void* *param	*RTI_UINT32*
get_serialized_key_max_size	in: *struct NDDS_Type_Plugin* *plugin in: *RTI_UINT32* current_alignment in: *void* *param	*RTI_UINT32*
create_sample	in: *struct NDDS_Type_Plugin* *plugin out: *void* **sample in: *void* *param	*RTI_BOOL*
delete_sample	in: *struct NDDS_Type_Plugin* *plugin in: *void* *sample in: *void* *param	*RTI_BOOL*
copy_sample	in: *struct NDDS_Type_Plugin* *plugin out: *void* *dst in: *void* *src in: *void* *param	*RTI_BOOL*
get_key_kind	in: *struct NDDS_Type_Plugin* *plugin in: *void* *param	*NDDS_TypePluginKeyKind*
instance_to_keyhash	in: *struct NDDS_Type_Plugin* *plugin in: *struct CDR_Stream_t* *stream out: *DDS_KeyHash_t* *key_hash in: *const void* *instance in: *void* *param	*RTI_BOOL*

Attribute	Parameters	Type
create_typed_datareader	in: *void* *reader	*void**
create_typed_datawriter	in: *void* *writer	*void**
delete_typed_datareader	in: *void* *wrapper	*void*
delete_typed_datawriter	in: *void* *wrapper	*void*

Rationale: RCC provides the definition of the *NDDS_Type_Plugin* structure. To implement the interface, functions need to be created that can be assigned to the attributes of the structure. The names of the functions created to implement the interface are arbitrary and are at the discretion of the implementer since function pointers are used to assign those functions to the member attributes.

The *rtiddsgen* utility is provided by RCC for automatically generating type support code rather than manually implementing each new type by the Platform Integrator.

6.1 serialize_data

R-700.1

The *serialize_data* operation **shall** serialize a sample into a *Stream_t* instance, performing a transformation from its in-memory representation to the equivalent network representation obtained by using serialization rules defined by the CDR specification.

6.2 deserialize_data

R-700.2

The *deserialize_data* operation **shall** deserialize a sample from a *Stream_t* instance, performing a transformation from its network representation obtained by using serialization rules defined by the CDR specification to its equivalent in-memory representation.

6.3 get_serialized_sample_max_size

R-700.3

The *get_serialized_sample_max_size* operation **shall** retrieve the maximum size (in bytes) of a sample when serialized into network representation, using serialization rules defined by the CDR specification.

6.4 create_sample

R-700.4

The *create_sample* operation **shall** retrieve a sample, possibly allocating and initializing the memory required to store it.

6.5 copy_sample

R-700.5

The *copy_sample* operation **shall** perform a deep copy of all attributes from one sample to another.

6.6 serialize_key

R-700.6

The *serialize_key* operation **shall** serialize a key provided as *sample* into a *Stream_t* instance, performing a transformation from its in-memory representation to the equivalent network representation obtained by using serialization rules defined by the CDR specification.

6.7 deserialize_key

R-700.7

The *deserialize_key* operation **shall** deserialize a key from a *Stream_t* instance to *sample*, performing a transformation from its network representation obtained by using serialization rules defined by the CDR specification to its equivalent in-memory representation.

6.8 get_serialized_key_max_size

R-700.8

The *get_serialized_key_max_size* operation **shall** retrieve the maximum size (in bytes) of a key when serialized into network representation, using serialization rules defined by the CDR specification.

7 Zero Copy Transfer Requirements

This section describes the Platform Integrator's responsibilities to integrate the Zero Copy v2 transfer implementation with the *RTI Connext Cert* (RCC) Platform Independent Library (PIL).The following sections contain information about Zero Copy v2 transfer and the requirements for supporting the Zero Copy v2 transport within the Platform Support Library.

In the context of *RTI Connext Cert* (RCC), "Zero Copy v2" refers to a data transport that can effectively present data, written by some DDS *DataWriter*, to a DDS *DataReader*, with very low latency compared to traditional UDP-based transports.

The Zero Copy v2 transport gets its name from the fact that no user data is actually "moved" over a network, which would involve multiple copies of that data at various stages (i.e., between the user application and the middleware, and between the middleware and the operating system). Instead, when using the Zero Copy v2 transport, the producer of the data (a DDS *DataWriter*) writes the instance of that data to some location in memory that is also accessible to the data consumer (one or more DDS *DataReaders*.) With this configuration, all consumers observe the actual memory being modified by the producer. A signaling mechanism referred to as the SHMEM Notification Mechanism (*Notif*) alerts consumers that new data is available. This mechanism is very lightweight and utilizes shared memory similarly to the user data itself. In the Zero Copy v2 Transport, *Notif* refers to the shared memory mechanism that allows signaling and synchronization between matched writers and readers.

The reduced latency resulting from fewer copies benefits a broad range of applications.

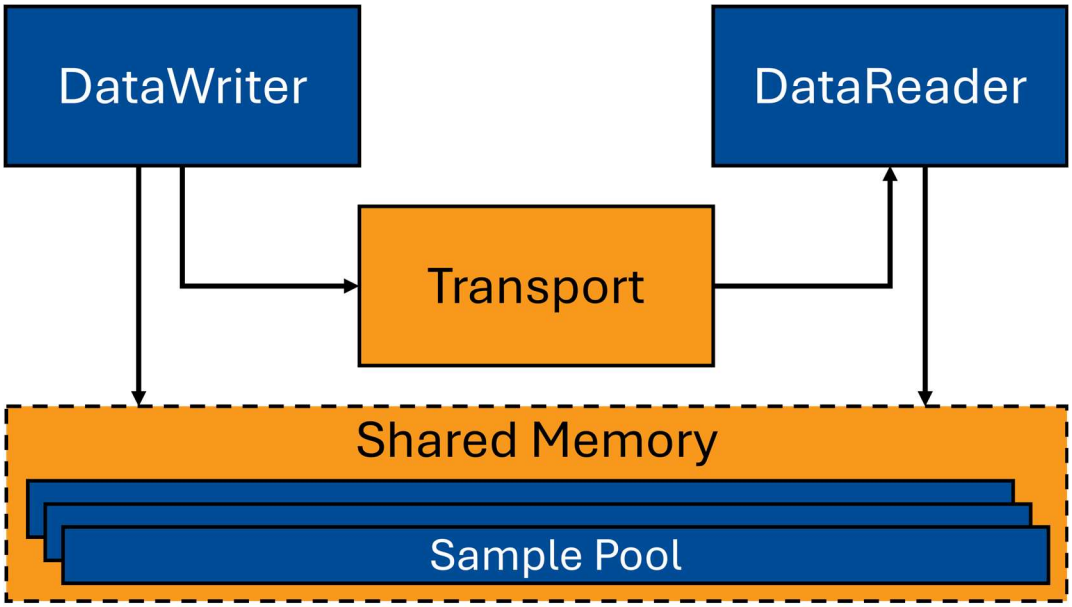


Figure 5-1: Zero Copy v2 Transport

7.1 Terminology

Acronyms and Abbreviations

Acronym	Description
ZC	Zero Copy v2
SHMEM	Shared Memory
Notif	Notification Mechanism

7.2 Concurrency

All PSL APIs have information about concurrency/thread-safety included in the API reference manuals. It is important that an application does not violate thread-safety/concurrency guidance. The Platform Integrator may create multiple threads as part of the implementation of the PSL, but from an application point of view, there is only a single critical section protecting all DDS resources within a *DomainParticipant*.

The Platform Integrator is responsible for implementing the concurrency and reentrancy requirements for PSL interfaces described in subsequent sections.

7.3 Zero Copy v2 PSL OSAPI Integration

RCC's ZCv2 OSAPI is an abstract API consisting of a set of functions implemented in the Platform Support Library (PSL) to satisfy the interface requirements of the Platform Independent Library (PIL) for the Zero Copy transfer feature.

The ZCv2 OSAPI is implemented using Operating System-dependent APIs, specifically those pertaining to the following areas:

- Shared memory segment
- ZCv2 OSAPI Type Plugin

7.3.1 SHMEM Segment Integration

RCC provides API for creating, managing, and interrogating segments of shared memory. Shared memory segments allow data access between processes that share memory.

RCC creates shared memory segments to facilitate data exchange without the need to copy data.

The Platform Integrator is responsible for implementing the following SHMEM Segment APIs:

- *OSAPI_SharedMemorySegment_exists()*
- *OSAPI_SharedMemorySegment_get_max_size()*
- *OSAPI_SharedMemorySegment_is_owner_alive()*
- *OSAPI_SharedMemorySegmentHandle_get_size()*
- *OSAPI_SharedMemorySegmentHeader_get_size()*
- *OSAPI_SharedMemorySegment_create_impl()*
- *OSAPI_SharedMemorySegment_delete_impl()*
- *OSAPI_SharedMemorySegment_attach_impl()*
- *OSAPI_SharedMemorySegment_detach_impl()*
- *OSAPI_SharedMemorySegment_lock_impl()*
- *OSAPI_SharedMemorySegment_unlock_impl()*
- *OSAPI_SharedMemorySegment_mark_consistent_impl()*

Data types used by the SHMEM Segment API:

- *struct OSAPI_SharedMemorySegmentHandle*
- *OSAPI_SHMEM_MODE*
- *OSAPI_SHMEM_STATUS*

The *OSAPI_SharedMemorySegmentHandle* data structure is used to interact with a shared memory segment. It is initialized by the *OSAPI_SharedMemorySegment_create_impl* and *OSAPI_SharedMemorySegment_attach_impl* operations. The composition of the *OSAPI_SharedMemorySegmentHandle* data structure is defined by RCC. It contains information necessary to interact with a specific shared memory segment.

OSAPI_SHMEM_MODE is an enumerated type that defines specific capabilities of a shared memory segment.

OSAPI_SHMEM_STATUS is an enumerated type that defines the possible states of a shared memory segment.

R-601.1

The *OSAPI_SharedMemorySegment_exists* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_exists	in: *const char* *name	*RTI_BOOL*

Rationale: *name* is the OSAPI_SharedMemorySegment name to lookup in the system.

R-601.1.1

The *OSAPI_SharedMemorySegment_exists* operation **shall** return *RTI_TRUE* if the *OSAPI_SharedMemorySegment* referenced by the name parameter exists.

R-601.1.2

The *OSAPI_SharedMemorySegment_exists* operation **shall** return *RTI_FALSE* if the *OSAPI_SharedMemorySegment* referenced by the *name* parameter does not exist or if it cannot be determined.

R-601.2

The *OSAPI_SharedMemorySegment_get_max_size* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_get_max_size	in: *void*	*RTI_UINT64*

R-601.2.1

The *OSAPI_SharedMemorySegment_get_max_size* operation **shall** return the maximum number of bytes an *OSAPI_SharedMemorySegment* can hold.

Rationale: This operation is infallible.

R-601.3

The *OSAPI_SharedMemorySegment_is_owner_alive* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_is_owner_alive	in: *struct OSAPI_SharedMemorySegmentHandle* *handle out: *RTI_BOOL* *alive	*RTI_BOOL*

Rationale: *handle* is the handle to the OSAPI_SharedMemorySegment. *alive* carries out information about the liveliness of the OSAPI_SharedMemorySegment owner.

R-601.3.1

The *OSAPI_SharedMemorySegment_is_owner_alive* operation, upon success, **shall** set alive to *RTI_TRUE* if the owner of the *OSAPI_SharedMemorySegment* referenced by its *handle* parameter is alive or to *RTI_FALSE* otherwise, and return *RTI_TRUE*.

Rationale: The owner is the execution context that creates the shared *OSAPI_SharedMemorySegment*.

R-601.3.2

The *OSAPI_SharedMemorySegment_is_owner_alive* operation **shall** return *RTI_FALSE* if the liveliness of the *OSAPI_SharedMemorySegment* owner referenced by the *handle* parameter cannot be determined.

R-601.4

The *OSAPI_SharedMemorySegmentHandle_get_size* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegmentHandle_get_size	in: *OSAPI_SHMEM_MODE* mode	*RTI_SIZE_T*

Rationale: *mode* is the requested *OSAPI_SharedMemorySegmentHandle* mode for the query. The *OSAPI_SHMEM_MODE* enum is declared by the PIL.

R-601.4.1

The *OSAPI_SharedMemorySegmentHandle_get_size* operation **shall** return the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is higher than or equal to the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter.

Rationale: This operation is infallible. The OSAPI_SHARED_MEMORY_ALIGNMENT value is declared by the PIL.

R-601.5

The *OSAPI_SharedMemorySegmentHeader_get_size* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegmentHeader_get_size	in: *OSAPI_SHMEM_MODE* mode	*RTI_SIZE_T*

Rationale: *mode* is the requested *OSAPI_SharedMemorySegmentHeader* mode for the query. The *OSAPI_SHMEM_MODE* enum is declared by the PIL.

R-601.5.1

The *OSAPI_SharedMemorySegmentHeader_get_size* operation **shall** return the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is higher than or equal to the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the mode parameter.

Rationale: This operation is infallible. The *OSAPI_SHARED_MEMORY_ALIGNMENT* value is declared by the PIL.

R-601.6

The *OSAPI_SharedMemorySegment_create_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_create_impl	inout: *struct OSAPI_SharedMemorySegmentHandle* *handle in: *const char* *name in: *RTI_UINT64* size in: *OSAPI_SHMEM_MODE* mode out: *RTI_BOOL* *exists	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*. *name* is the name to create the *OSAPI_SharedMemorySegment* with. *size* is the minimum size to create the *OSAPI_SharedMemorySegment* with. *mode* is the mode to create the *OSAPI_SharedMemorySegment* with. *exists* carries out information about the current existence of another *OSAPI_SharedMemorySegment* with the same name.

R-601.6.1

The *OSAPI_SharedMemorySegment_create_impl* operation, upon success, **shall:**

- create a new initialized *OSAPI_SharedMemorySegment*:
  - accessible to other processes;
  - named according to the name specified by the *name* parameter;
  - of a size that is equal to or greater than the size specified by the *size* parameter;
  - with a mode specified by the *mode* parameter;
- and then:
  - populate the handle with the *OSAPI_SharedMemorySegment* information;
  - set the *exists* parameter to *RTI_FALSE*; and
  - return *RTI_TRUE*.

R-601.6.2

If there is already an *OSAPI_SharedMemorySegment* in the system identified by the name specified by the *name* parameter, the *OSAPI_SharedMemorySegment_create_impl* operation **shall** set the *exists* parameter to *RTI_TRUE* and return *RTI_FALSE*.

Rationale: Solved.

R-601.6.3

If the *OSAPI_SharedMemorySegment_create_impl* operation cannot create the new *OSAPI_SharedMemorySegment* it **shall** return *RTI_FALSE*.

R-601.7

The *OSAPI_SharedMemorySegment_delete_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_delete_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*.

R-601.7.1

The *OSAPI_SharedMemorySegment_delete_impl* operation, upon success, **shall** delete the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter and return *RTI_TRUE*.

Rationale: An *OSAPI_SharedMemorySegment* can no longer be used after returning from a successful call to *OSAPI_SharedMemorySegment_delete_impl*. The caller of *OSAPI_SharedMemorySegment_delete_impl* cannot expect to retrieve the information held by the deleted *OSAPI_SharedMemorySegment* even if the *OSAPI_SharedMemorySegment* is re-created with the same arguments.

R-601.7.2

If the *OSAPI_SharedMemorySegment_delete_impl* operation cannot delete the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter, it **shall** return *RTI_FALSE*.

R-601.8

The *OSAPI_SharedMemorySegment_attach_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_attach_impl	inout: *struct OSAPI_SharedMemorySegmentHandle* *handle in: *const char* *name, in: *OSAPI_SHMEM_MODE* mode	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*. *name* is the name of the *OSAPI_SharedMemorySegment* to attach to. *mode* is the mode in which the *OSAPI_SharedMemorySegment* will be attached to.

R-601.8.1

The *OSAPI_SharedMemorySegment_attach_impl* operation, if succeeds, **shall** attach the *OSAPI_SharedMemorySegmentHandle* specified by the *handle* parameter, in the mode specified by the *mode* parameter, to an existing *OSAPI_SharedMemorySegment* with a name equal to the *name* parameter, and return *RTI_TRUE*.

Rationale: The caller of this operation should be capable of attaching the *OSAPI_SharedMemorySegment* more than once.

R-601.8.2

If the *OSAPI_SharedMemorySegment_attach_impl* operation cannot attach, in the mode specified by the *mode* parameter, to an *OSAPI_SharedMemorySegment* with a name equal to the *name* parameter, it **shall** return *RTI_FALSE*.

R-601.9

The *OSAPI_SharedMemorySegment_detach_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_detach_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*.

R-601.9.1

The *OSAPI_SharedMemorySegment_detach_impl* operation, upon success, **shall** detach the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter and return *RTI_TRUE*.

Rationale: An *OSAPI_SharedMemorySegment* can no longer be used by an endpoint after returning from a successful call to the *OSAPI_SharedMemorySegment_detach_impl* operation.

The *OSAPI_SharedMemorySegment_detach_impl* operation causes the resource to be "disconnected" from an endpoint without being deleted. An endpoint can not use a detached shared memory segment to communicate. It may be possible for an endpoint to re-attach to a detached shared memory segment.

An *OSAPI_SharedMemorySegment* can be reused (i.e. the information is persisted) if it hasn't been deleted by calling the *OSAPI_SharedMemorySegment_delete_impl* operation.

R-601.9.2

If the *OSAPI_SharedMemorySegment_detach_impl* operation cannot detach the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter, it **shall** return *RTI_FALSE*.

R-601.10

The *OSAPI_SharedMemorySegment_lock_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_lock_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle out: *OSAPI_SHMEM_STATUS* *status_out	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*. *status_out* carries out information about the *OSAPI_SharedMemorySegment* status.

R-601.10.1

The *OSAPI_SharedMemorySegment_lock_impl* operation **shall** lock the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter if the *OSAPI_SharedMemorySegment* is not locked yet.

Rationale: This operation will only be called if the *OSAPI_SharedMemorySegment* was created with a mode as strict or stricter than *OSAPI_SHMEM_MODE_LOCKABLE*.

R-601.10.2

If the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter is already locked, then the *OSAPI_SharedMemorySegment_lock_impl* operation **shall** block the calling execution context until the lock is released and then lock the *OSAPI_SharedMemorySegment*.

R-601.10.3

The *OSAPI_SharedMemorySegment_lock_impl* operation, upon success, **shall**:

1. return *RTI_TRUE*.
2. return *OSAPI_SHMEM_STATUS_OK* through the *status_out* parameter.

R-601.10.4

The *OSAPI_SharedMemorySegment_lock_impl* operation **shall** return *OSAPI_SHMEM_STATUS_OWNER_DEAD* through the *status_out* parameter if the previous owner of the *OSAPI_SharedMemorySegment* terminated while holding a lock on the *OSAPI_SharedMemorySegment*.

Rationale: The owner is the execution context that locked the shared *OSAPI_SharedMemorySegment*.

R-601.10.5

If the *OSAPI_SharedMemorySegment* cannot be locked, the *OSAPI_SharedMemorySegment_lock_impl* operation **shall** return *RTI_FALSE*.

R-601.11

The *OSAPI_SharedMemorySegment_unlock_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_unlock_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*.

R-601.11.1

The *OSAPI_SharedMemorySegment_unlock_impl* operation, upon success, **shall** unlock the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter and return *RTI_TRUE*.

Rationale: This operation will only be called if the *OSAPI_SharedMemorySegment* was created with a mode as strict or stricter than *OSAPI_SHMEM_MODE_LOCKABLE*.

R-601.11.2

If the *OSAPI_SharedMemorySegment* cannot be unlocked, the *OSAPI_SharedMemorySegment_unlock_impl* operation **shall** return *RTI_FALSE*.

R-601.12

The *OSAPI_SharedMemorySegment_mark_consistent_impl* operation **shall** have the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_mark_consistent_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

Rationale: *handle* is the handle to the *OSAPI_SharedMemorySegment*.

R-601.12.1

The *OSAPI_SharedMemorySegment_mark_consistent_impl* operation, upon success, **shall** mark the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter as consistent and return *RTI_TRUE*.

Rationale:

This operation will only be called if the *OSAPI_SharedMemorySegment* was created with a mode as strict or stricter than *OSAPI_SHMEM_MODE_LOCKABLE* and if *OSAPI_ROBUST_LOCKING_ENABLED* is *ON*.

R-601.12.2

If the *OSAPI_SharedMemorySegment* cannot be marked as consistent, or the support for robust locking is not enabled, the *OSAPI_SharedMemorySegment_mark_consistent_impl* operation **shall** return *RTI_FALSE*.

7.3.2 Type Plugin

RCC provides a generic Type service called the Type plugin. The Type plugin support includes the following operations for data types used when the Zero Copy feature is enabled: get and discard loan, create managed pool, and get sample ID.

The Platform Integrator may manually implement each new type, but there is a strong suggestion to use the *rtiddsgen* utility provided by RCC for automatically generating type support code.

The Type Plugin requirements are not obligatory to be implemented as *rtiddsgen* should preferably be used. They are provided in the Integration Manual for completeness, in case the user or integrator chooses to implement the plugins without the assistance of *rtiddsgen*. They are also used to verify type support code on the target platform.

For any further details regarding the Type plugin, please refer to **6 PSL Type Plugin Requirements**.

R-604.0

To create a Type plugin implementation for Zero Copy transport, the following additional member attributes assigned function pointers to functions that implement the interfaces **shall** be set while creating an instance of the *NDDS_Type_Plugin* structure:

Attribute	Parameters	Type
get_loan	in: *void* *managed_pool out: *void*** sample_out	*ReturnCode_t*
discard_loan	in: *void* *managed_pool in: *RTI_UINT32* sample_id	*RTI_BOOL*
create_managed_pool	in: *struct NDDS_Type_Plugin* *plugin in: *void* *datawriter in: *void* *param out: *void* **managed_pool_out	*RTI_BOOL*
delete_managed_pool	in: *void* *managed_pool	*void*
get_sample_id	in: *void* *managed_pool in: *const void* *user_data out: *RTI_UINT32* *sample_id_out	*RTI_BOOL*
needs_opaque_samples	in: *void* *reader in: *void* *param	*RTI_BOOL*

Rationale: This subset of attributes is only for Zero Copy transport. The remaining attributes are described in R-700.0.

To implement the interface, functions need to be created that can be assigned to the attributes of the structure. The names of the functions created to implement the interface are arbitrary and are at the discretion of the implementer since function pointers are used to assign those functions to the member attributes.

The *rtiddsgen* utility is provided by RCC for automatically generating type support code rather than manually implementing each new type by the Platform Integrator.

7.3.2.1 get_loan

R-604.1

The *get_loan* operation **shall** request a loan of a sample from a managed pool of samples.

Rationale: This operation is only needed if the Zero Copy component is present.

7.3.2.2 discard_loan

R-604.2

The *discard_loan* operation **shall** return a loaned sample previously retrieved by *get_loan* operation to a managed pool of samples.

Rationale: This operation is only needed if the Zero Copy component is present.

7.3.2.3 create_managed_pool

R-604.3

The *create_managed_pool* operation **shall** create a managed pool of samples if it is needed.

Rationale: The criterion for the need of a managed pool to be created is the implementation detail. This function can successfully exit without creating a managed pool.

This operation is only needed if the Zero Copy component is present.

7.3.2.4 get_sample_id

R-604.4

The *get_sample_id* operation **shall** lookup the unique ID of a loaned sample given its user data memory region.

Rationale: This operation is only needed if the Zero Copy component is present.

7.3.2.5 needs_opaque_samples

R-604.5

The *needs_opaque_samples* operation **shall** check whether the additional opaque samples are needed.

Rationale: This operation is only needed if the Zero Copy component is present.

7.3.3 Concurrency

R-605.1

The following PSL operations **shall** be reentrant and thread-safe:

- *OSAPI_SharedMemorySegment_exists*
- *OSAPI_SharedMemorySegment_get_max_size*
- *OSAPI_SharedMemorySegment_is_owner_alive*
- *OSAPI_SharedMemorySegmentHandle_get_size*
- *OSAPI_SharedMemorySegmentHeader_get_size*
- *OSAPI_SharedMemorySegment_create_impl*
- *OSAPI_SharedMemorySegment_delete_impl*
- *OSAPI_SharedMemorySegment_attach_impl*
- *OSAPI_SharedMemorySegment_detach_impl*
- *OSAPI_SharedMemorySegment_lock_impl*
- *OSAPI_SharedMemorySegment_unlock_impl*
- *OSAPI_SharedMemorySegment_mark_consistent_impl*

Rationale: An operation is thread-safe if it guarantees safe execution by multiple threads at the same time, either multitasking or real parallelism. A function is reentrant if it can be interrupted at any point during its execution and then safely called again before its previous invocations complete execution, such as a recursive function.

7.4 Zero Copy v2 Notification Transport Integration

The Zero Copy v2 Notification Transport is used to notify processes that shared memory associated events have occurred.

7.4.1 Notification Transport Plugin API

The Zero Copy v2 Notification Transport API consists of functions which are used to implement a user specified Notification transport. The Zero Copy v2 Notification Transport API is implemented as an interface using function pointers instead of an implementation using function prototypes.

The Zero Copy v2 Notification transport interface is defined by the *ZCOPY_NotifUserInterfaceI* type. The user's notification transport will be registered in the application by a call to the *ZCOPY_NotifInterfaceFactory_register* function which is provided by RCC.

The Platform Integrator creates functions to implement each of the function pointer structure members in the *ZCOPY_NotifUserInterfaceI* structure. The Platform Integrator assigns each of the created functions to the appropriate member of the *ZCOPY_NotifUserInterfaceI* structure.

R-602.1

To create a Notification transport plugin implementation, an instance of the *ZCOPY_NotifUserInterfaceI* structure **shall** be created and its member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Attribute	Parameters	Type	Description (non-normative)
create_instance	in: *NETIO_Interface_T* *upstream in: *void* *property	*NETIO_Interface_T**	Create an instance of the Notification Mechanism interface with *upstream* as the owner.
reserve_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr out: *void*** port_entry_out	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to setup resources for listening to messages on the address *src_addr and return a port_entry_out.* The *port_entry_out* will be provided to a *bind* call.
release_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *void** port_entry	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to release resources for listening to messages on the address *src_addr.*
resolve_address	in: *NETIO_Interface_T* *user_intf in: *const* char *address_string out: *struct NETIO_Address* *address_value out: *RTI_BOOL* *invalid	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by user_*intf* to determine if the address string *address_string* is a valid address and return the result in *address_value.*

Attribute	Parameters	Type	Description (non-normative)
send	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination in: *void** route_entry	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by user_intf to send a notification to the *destination using the route_entry.*
get_route_table	in: *NETIO_Interface_T* *netio_intf inout: *struct NETIO_AddressSeq* *address inout: *struct NETIO_NetmaskSeq* *netmask	*RTI_BOOL*	Instruct the Notification Mechanism interface *netio_intf* to return a sequence of *address* and *netmask* pairs this interface can send to.
bind	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *struct NETIO_Address* *dst_addr in: *void* *port_entry out: *void* **bind_entry_out	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to start listening for messages on the address specified by *src_addr on the given port_entry.*
unbind	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *struct NETIO_Address* *dst_addr in: *void* *port_entry in: *void* *bind_entry	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to stop listening for messages on the address specified by *src_addr.*

Attribute	Parameters	Type	Description (non-normative)
add_route	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination out: *void*** route_entry_out	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to add a route from the *source* to the *destination*. The *route_entry_out* will be provided to the user later when calling *send*
delete_route	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination in: *void** route_entry	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to remove a route from the *source* to the *destination*.
notify_recv_port	in: *NETIO_Interface_T* *user_intf in: *void** port_entry	*RTI_BOOL*	Instruct the Notification Mechanism interface specified by *user_intf* to notify the receive port specified by *port_entry*.

Rationale: RCC provides the definition of the *ZCOPY_NotifUserInterfaceI* structure. To implement the interface, functions need to be created that can be assigned to the attributes of the structure. The names of the functions created to implement the interface are arbitrary and are at the discretion of the implementer since function pointers are used to assign those functions to the member attributes.

7.4.1.1 create_instance

R-602.1.1

The *ZCOPY_NotifUserInterfaceI.create_instance* operation **shall** return a non-NULL pointer to an instance of an interface of type *NETIO_Interface_T* on success.

Rationale: The structure member *ZCOPY_NotifUserInterfaceI.create_instance* is used to allocate and instantiate any resources needed for the user's desired transport. The created *NETIO_Interface_T* instance will later be used to send packets of type *NETIO_Packet_T*. RCC will try to send packets using the *ZCOPY_NotifUserInterfaceI.send operation*. RCC will receive packets when this implementation calls the *NETIO_DGRAM_Interface_upstream_receive* operation with upstream as the *netio_intf* parameter, meaning that upstream has to be saved.

The property parameter carries the information passed to the *ZCOPY_NotifMechanism_register* operation as the user properties. The property parameter can be used to provide properties needed by the *ZCOPY_NotifUserInterfaceI.create_instance*. The use of the property parameter is at the discretion of the implementer. A pointer to the property structure could have a NULL value.

The user can leverage a derived type from the base type *NETIO_Interface_T* in order to save all the needed resources for the internal operation of the interface (such as the upstream parameter), but will have to return a pointer to a downcasted version of it.

R-602.1.2

The *ZCOPY_NotifUserInterfaceI.create_instance* operation **shall** return NULL on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- Insufficient memory for allocations
- Resources are not available
- Insufficient access permissions
- Improper values for parameters

R-602.1.3

The PSL **shall** not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *ZCOPY_NotifUserInterfaceI.create_instance*.

Rationale: The instance of an interface of type *NETIO_Interface_T* is not fully initialized by the time the *ZCOPY_NotifUserInterfaceI.create_instance* is called by RCC.

7.4.1.2 reserve_address

R-602.1.11

The *ZCOPY_NotifUserInterfaceI.reserve_address* operation **shall** cause the interface specified by *user_intf* to configure resources for listening to messages on address *src_addr and* return a pointer to a port entry specified by *port_entry_out* and return *RTI_TRUE* on success.

Rationale: When an interface reserves an address, the interface will setup the resources needed to listen for messages sent to that address from the address indicated by *source.*

R-602.1.12

The *ZCOPY_NotifUserInterfaceI.reserve_address* operation **shall** return NULL through the port entry specified by *port_entry_out* and return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation.

Examples of potential failures are:

- Insufficient memory
- Resources are not available
- Insufficient access permissions
- Improper values for parameters

R-602.1.13

The *ZCOPY_NotifUserInterfaceI.reserve_address* operation **shall** return a pointer to a port entry specified by *port_entry_out* on success.

Rationale: The *port_entry_out* pointer will be provided as a parameter to the *ZCOPY_NotifUserInterfaceI.bind* operation.

R-602.1.14

The *ZCOPY_NotifUserInterfaceI.reserve_address* operation **shall** return a NULL_PTR as the port entry specified by *port_entry_out* on failure.

7.4.1.3 release_address

R-602.1.21

The *ZCOPY_NotifUserInterfaceI.release_address* operation **shall** cause the interface specified by *user_intf* to release resources associated with the port entry specified by *port_entry* used for listening to messages on address *src_addr* and return *RTI_TRUE* on success.

Rationale: When an interface releases an address, the interface will release the resources needed to listen for messages sent to that address from the address indicated by *src_addr*. This operation is the compliment of *ZCOPY_NotifUserInterfaceI.reserve_address*.

R-602.1.22

The *ZCOPY_NotifUserInterfaceI.release_address* operation **shall** return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation.

Examples of potential failures are:

- Resources already released
- Insufficient access permissions
- Improper values for parameters

7.4.1.4 resolve_address

R-602.1.31

The *ZCOPY_NotifUserInterfaceI.resolve_address* operation **shall** set *address_value* to a valid address for the Notification Mechanism interface specified by *user_intf,* set the value pointed to by *invalid* to *RTI_FALSE*, **AND** return *RTI_TRUE* on success.

Rationale: An address is valid when it meets the requirements of the user's desired transport. The possible failure modes are implementation dependent and are specified by the implementer. The *invalid* parameter should always return *RTI_FALSE* based on the current design of the PIL. This allows other transports to resolve the address if the implementation of the Notification Mechanism transport does not understand how to interpret the address. The *address_string* parameter provided by RCC can be used by the Platform Integrator to determine the *address_value.*

R-602.1.32

The *ZCOPY_NotifUserInterfaceI.resolve_address* operation **shall** return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation.

One example of failure is improper values for *address_string.*

R-602.1.33

If the *address_string* provided to the *ZCOPY_NotifUserInterfaceI.resolve_address* operation is an invalid address for the Notif Mechanism interface specified by *user_intf*, the *ZCOPY_NotifUserInterfaceI.resolve_address* operation **shall** set the value pointed to by *invalid* to *RTI_TRUE*, otherwise set it to *RTI_FALSE*.

Rationale: This requirements describes how the *invalid* parameter is set depending on the failure that occurs.

7.4.1.5 send

R-602.1.41

The *ZCOPY_NotifUserInterfaceI.send* operation **shall** cause the interface specified by *user_intf* to send a notification from the source specified by *source* to the destination specified by *destination* using the route entry specified by *route_entry* and return *RTI_TRUE* on success.

R-602.1.42

The *ZCOPY_NotifUserInterfaceI.send* operation **shall** return *RTI_FALSE* on failure.

Rationale: Failure modes are determined by the implementer of this method. Examples of failures can include:

- An invalid *route_entry* pointer
- Failure to acquire exclusive access to Notification Mechanism's shared memory
- Failure to post a notification in the shared memory

7.4.1.6 get_route_table

R-602.1.51

On success the *ZCOPY_NotifUserInterfaceI.get_route_table* operation **shall**:

- retrieve a sequence of all addresses as *address* and a sequence of all netmasks as *netmask* from the routing table to which the Notification Mechanism interface specified by *netio_intf* can send
- return *RTI_TRUE*

Rationale: The *ZCOPY_NotifUserInterfaceI.get_route_table* operation is used to retrieve the current route table associated with the interface *netio_intf* and this information will be used by RCC to determine if the interface is able to send packets to a specific *NETIO_Address* destination. The implementor is responsible for determining and storing the routes applicable for the desired transport. Some platforms may have fixed routes while others may be determined dynamically.

The route table could be stored separately from the *netio_intf* or in conjunction with it—as a part of the property *user_property* (optionally provided during the registration of the Notification transport; i.e., during a *ZCOPY_NotifMechanism_register* function call) which automatically associates property data with the registered Notification transport.

R-602.1.52

The *ZCOPY_NotifUserInterfaceI.get_route_table* operation **shall** not modify the *address* and *netmask* sequences and return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation.

Examples of potential failures are:

- Invalid pointers to the parameters (i.e. NULL pointers)
- Insufficient access permissions

7.4.1.7 bind

R-602.1.61

The *ZCOPY_NotifUserInterfaceI.bind* operation shall cause the implementation to listen for notification messages and return *RTI_TRUE* on success.

Rationale: RCC provides several parameters that the Platform Integrator may use if required by their implementation of the Notification Mechanism.

user_intf, *src_addr*, *dst_addr*, and *port_entry* contain useful state information that the Platform Integrator may use in the implementation of the *ZCOPY_NotifUserInterfaceI.bind* operation.

bind_entry_out can be used by the Platform Integrator to provide a reference to a bind entry utilized within the *ZCOPY_NotifUserInterfaceI.bind* operation.

R-602.1.62

The *ZCOPY_NotifUserInterfaceI.bind* operation **shall** return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- Insufficient memory
- Resources are not available
- Insufficient access permissions
- Improper values for parameters

7.4.1.8 unbind

R-602.1.71

The *ZCOPY_NotifUserInterfaceI.unbind* operation **shall** return *RTI_TRUE*.

Rationale: RCC provides the *ZCOPY_NotifUserInterfaceI.unbind* operation interface as a mechanism that allows Platform Integrators to stop listening to messages from the address *src_addr* on the Notification Mechanism interface specified by *user_intf*.

If the Platform Integrator's implementation of the Notification Mechanism supports the ability to stop listening to messages, then they will use the *ZCOPY_NotifUserInterfaceI.unbind* operation to implement the unbind behavior. The *ZCOPY_NotifUserInterfaceI.unbind* operation will always return *RTI_TRUE* on completion. If there are any errors that occur during the *ZCOPY_NotifUserInterfaceI.unbind* operation, then the Platform Integrator is responsible for handling errors as part of the implementation.

If, however, the Platform Integrator's implementation does not support the unbind behavior or if the Platform Integrator does not wish to support the unbind behavior, then the Platform Integrator should cause the *ZCOPY_NotifUserInterfaceI.unbind* operation to return *RTI_TRUE*.

user_intf, *src_addr*, *dst_addr*, *port_entry,* and *bind_entry* contain useful state information that the Platform Integrator may use in the implementation of the *ZCOPY_NotifUserInterfaceI.unbind* operation.

R-602.1.72

The *ZCOPY_NotifUserInterfaceI.unbind* operation **shall** return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- Insufficient memory

- Resources are not available
- Insufficient access permissions
- Improper values for parameters

7.4.1.9 add_route

R-602.1.81

The *ZCOPY_NotifUserInterfaceI.add_route* operation **shall** cause the interface specified by *user_intf* to add a route from the source specified by *source* to the destination specified by *destination and* return a pointer to a route entry specified by *route_entry_out* and return *RTI_TRUE* on success.

Rationale: The *route_entry_out* will be provided to the user later when calling the *ZCOPY_NotifUserInterfaceI.send* operation. The *ZCOPY_NotifUserInterfaceI.add_route* operation provides a mechanism for routing notifications to the *destination* address.

R-602.1.82

The *ZCOPY_NotifUserInterfaceI.add_route* operation **shall** return NULL through the port entry specified by *route_entry_out* and return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- Insufficient memory
- Resources are not available
- Insufficient access permissions
- Improper values for parameters

R-602.1.83

The *ZCOPY_NotifUserInterfaceI.add_route* operation **shall** return a pointer to a route entry specified by *route_entry_out* on success.

Rationale: The route_*entry_out* pointer will be provided as a parameter to the *ZCOPY_NotifUserInterfaceI.send* operation.

R-602.1.84

The *ZCOPY_NotifUserInterfaceI.add_route* operation **shall** return a NULL_*PTR* as the port entry specified by *route_entry_out* on failure.

7.4.1.10 delete_route

R-602.1.91

The *ZCOPY_NotifUserInterfaceI.delete_route* operation **shall** cause the interface specified by *user_intf* to remove the *route_entry* from the source specified by *source* and the destination specified by *destination* and return *RTI_TRUE* on success.

R-602.1.92

The *ZCOPY_NotifUserInterfaceI.delete_route* operation **shall** return *RTI_FALSE* on failure.

Rationale: The Platform Integrator determines what failure mechanisms are relevant to their transport implementation. Examples of potential failures are:

- An invalid *route_entry* pointer
- Failure to remove the route to the Notification Mechanism's shared memory

7.4.1.11 notify_recv_port

R-602.1.101

The *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation **shall** cause the interface specified by *user_intf* to notify the receive port specified by *port_entry* and return *RTI_TRUE* on success.

R-602.1.102

The *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation **shall** return *RTI_FALSE* on failure.

Rationale: Failure modes are determined by the implementer of this method. Examples of failures can include:

- An invalid *port_entry* pointer
- Failure to issue a notification

7.4.2 Concurrency

R-606.1

The following ZCOPY_NotifUserInterfaceI operations shall be reentrant and thread-safe:

- *ZCOPY_NotifUserInterfaceI.create_instance*
- *ZCOPY_NotifUserInterfaceI.reserve_address*
- *ZCOPY_NotifUserInterfaceI.release_address*
- *ZCOPY_NotifUserInterfaceI.resolve_address*

- *ZCOPY_NotifUserInterfaceI.send*
- *ZCOPY_NotifUserInterfaceI.get_route_table*
- *ZCOPY_NotifUserInterfaceI.bind*
- *ZCOPY_NotifUserInterfaceI.unbind*
- *ZCOPY_NotifUserInterfaceI.add_route*
- *ZCOPY_NotifUserInterfaceI.delete_route*
- *ZCOPY_NotifUserInterfaceI.notify_recv_port*

Rationale: An operation is thread-safe if it guarantees safe execution by multiple threads at the same time, either multitasking or real parallelism. A function is reentrant if it can be interrupted at any point during its execution and then safely called again before its previous invocations complete execution, such as a recursive function.

8 PSL OSAPI Test Specifications

8.1 Heap

QTS-R-521.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Heap_allocate_buffer* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Heap_allocate_buffer* has the following signature:

Operations	Parameters	Type
OSAPI_Heap_allocate_buffer	out: *char* ***buffer in: *RTI_SIZE_T* size in: *OSAPI_Alignment_T* alignment	*void*

QTS-R-521.1.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Heap_allocate_buffer* allocates memory.

TC.1

- **Preconditions:**
  - *OSAPI_Heap_allocate_buffer* succeeds when called.
- **Actions:**
  - call *OSAPI_Heap_allocate_buffer*.
- **Check that:**
  - the address of the allocated memory area has been returned via the *buffer* parameter.
  - the allocated memory area is aligned to *alignment*.
  - the allocated memory area is of *size* contiguous bytes, each of which has been set to zero.

QTS-R-521.1.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Heap_allocate_buffer* fails.

TC.1

- **Preconditions:**
  - *OSAPI_Heap_allocate_buffer* fails when called.
- **Actions:**
  - call *OSAPI_Heap_allocate_buffer*.
- **Check that:**
  - a NULL pointer is returned via the *buffer* parameter.

8.2 Mutex

QTS-R-522.1

Analysis: The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_new* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Mutex_new* has the following signature:

Operations	Parameters	Type
OSAPI_Mutex_new	in: *void*	*OSAPI_Mutex_T**

QTS-R-522.1.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_new* successfully creates a new mutex.

TC.1

- **Preconditions:**
  - *OSAPI_Mutex_new* succeeds when called.
- **Actions:**
  - call *OSAPI_Mutex_new*.
- **Check that:**
  - *OSAPI_Mutex_new* returns a pointer to the newly created mutex.
  - the newly created mutex can guarantee exclusive access.

QTS-R-522.1.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_new* fails to create a new mutex.

TC.1

- **Preconditions:**
  - *OSAPI_Mutex_new* fails when called.
- **Actions:**
  - call *OSAPI_Mutex_new*.
- **Check that:**
  - *OSAPI_Mutex_new* returns NULL.

QTS-R-522.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_T*ake has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Mutex_T*ake has the following signature:

Operations	Parameters	Type
*OSAPI_Mutex_T*ake	in: *OSAPI_Mutex_T* *mutex	*RTI_BOOL*

QTS-R-522.2.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_Take* takes an unlocked mutex.
- [**TC.2**] Operation *OSAPI_Mutex_Take* takes a mutex locked by the current thread.

TC.1

- **Preconditions:**
  - *OSAPI_Mutex_Take* succeeds when called.
  - *mutex* is unlocked.
- **Actions:**
  - call *OSAPI_Mutex_Take* with *mutex*.
- **Check that:**
  - *OSAPI_Mutex_Take* returns *RTI_TRUE*.
  - the current execution context is made owner of *mutex*.

TC.2

- **Preconditions:**
  - *OSAPI_Mutex_Take* succeeds when called.
  - *mutex* is locked by the current thread.
- **Actions:**
  - call *OSAPI_Mutex_Take* with *mutex*.
- **Check that:**
  - *OSAPI_Mutex_Take* returns *RTI_TRUE*.
  - the current execution context remains owner of *mutex*.

QTS-R-522.2.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_Take* fails to take a mutex.

TC.1

- **Preconditions:**
  - *mutex* is unable to be taken.
- **Actions:**
  - call *OSAPI_Mutex_Take*.
- **Check that:**
  - *OSAPI_Mutex_Take* returns *RTI_FALSE*.

QTS-R-522.2.4

Analysis

The requirement is met when:

- [**TC.1**] *OSAPI_Mutex_Take* locks an unlocked mutex on success.

TC.1

- **Preconditions:**
  - *mutex* is unlocked.
  - *OSAPI_Mutex_Take* succeeds when called.
- **Actions:**
  - call *OSAPI_Mutex_Take* with *mutex*.
- **Check that:**
  - *mutex* is locked by the current execution context.

QTS-R-522.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_give* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Mutex_give* has the following signature:

Operations	Parameters	Type
OSAPI_Mutex_give	in: *OSAPI_Mutex_T* *mutex	*RTI_BOOL*

QTS-R-522.3.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_give* successfully gives a mutex.

TC.1

- **Preconditions:**
  - *mutex* is able to be given.
- **Actions:**
  - call *OSAPI_Mutex_give*.
- **Check that:**
  - *OSAPI_Mutex_give* returns *RTI_TRUE*.

QTS-R-522.3.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Mutex_give* fails to give a mutex.

TC.1

- **Preconditions:**
  - *mutex* is unable to be given.
- **Actions:**
  - call *OSAPI_Mutex_give*.
- **Check that:**
  - *OSAPI_Mutex_give* returns *RTI_FALSE*.

QTS-R-522.3.3

Analysis

The requirement is met when:

- [**TC.1**] A locked mutex is given as many times as it was taken and becomes unlocked.

TC.1

- **Preconditions:**
  - *mutex* is locked by the current execution context.
    - *mutex* has been successfully taken by the current execution context X times.
    - *mutex* has been successfully given by the current execution context X-1 times.
  - *OSAPI_Mutex_give* succeeds when called
- **Actions:**
  - call *OSAPI_Mutex_give* with *mutex*.
- **Check that:**
  - *mutex* is unlocked.

8.3 Semaphore

QTS-R-523.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_new* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Semaphore_new* has the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_new	in: *void*	*OSAPI_Semaphore_T**

QTS-R-523.1.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_new* successfully creates a new semaphore.

TC.1

- **Preconditions:**
  - *OSAPI_Semaphore_new* succeeds when called.
- **Actions:**
  - call *OSAPI_Semaphore_new*.
- **Check that:**
  - *OSAPI_Semaphore_new* returns a pointer to the newly created semaphore.
  - the newly created semaphore is in an unsignaled state.

QTS-R-523.1.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_new* fails to create a new semaphore.

TC.1

- **Preconditions:**
  - *OSAPI_Semaphore_new* fails when called.
- **Actions:**
  - call *OSAPI_Semaphore_new*.
- **Check that:**
  - *OSAPI_Semaphore_new* returns NULL

QTS-R-523.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_take* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Semaphore_take* has the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_take	in: *OSAPI_Semaphore_T* *self in: *RTI_INT32* timeout out: *RTI_INT32* *fail_reason	*RTI_BOOL*

QTS-R-523.2.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_take* succeeds with a NULL *fail_reason* parameter.
- [**TC.2**] Operation *OSAPI_Semaphore_take* succeeds with a non-NULL *fail_reason* parameter.

TC.1

- **Preconditions:**
  - *fail_reason* is NULL.
  - semaphore *self* can be taken.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - *fail_reason* is not NULL.
  - semaphore *self* can be taken.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_TRUE*.
  - *fail_reason*'s value is set to OSAPI_SEMAPHORE_RESULT_OK.

QTS-R-523.2.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_take* times out with a NULL *fail_reason* parameter.
- [**TC.2**] Operation *OSAPI_Semaphore_take* times out with a non-NULL *fail_reason* parameter.

TC.1

- **Preconditions:**
  - *fail_reason* is NULL.
  - the attempt to take semaphore *self* will time out during the call.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - *fail_reason* is not NULL.
  - the attempt to take semaphore *self* will time out during the call.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_TRUE*.
  - *fail_reason*'s value is set to *OSAPI_SEMAPHORE_RESULT_TIMEOUT*.

QTS-R-523.2.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_take* fails with a NULL *fail_reason* parameter.
- [**TC.2**] Operation *OSAPI_Semaphore_take* fails with a non-NULL *fail_reason* parameter.

TC.1

- **Preconditions:**
  - *fail_reason* is NULL.
  - *OSAPI_Semaphore_take* fails for any reason besides an expired timeout when called.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_FALSE*.

TC.2

- **Preconditions:**
  - *fail_reason* is not NULL.
  - *OSAPI_Semaphore_take* fails for any reason besides an expired timeout when called.
- **Actions:**
  - call *OSAPI_Semaphore_take*.
- **Check that:**
  - *OSAPI_Semaphore_take* returns *RTI_FALSE*.
  - *fail_reason*'s value is set to *OSAPI_SEMAPHORE_RESULT_ERROR*.

QTS-R-523.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_give* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Semaphore_give* has the following signature:

Operations	Parameters	Type
OSAPI_Semaphore_give	in: *OSAPI_Semaphore_T* *self	*RTI_BOOL*

QTS-R-523.3.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_give* succeeds.

TC.1

- **Preconditions:**
  - semaphore *self* can be successfully given.
- **Actions:**
  - call *OSAPI_Semaphore_give*.
- **Check that:**
  - *OSAPI_Semaphore_give* returns *RTI_TRUE*.

QTS-R-523.3.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Semaphore_give* fails.

TC.1

- **Preconditions:**
  - *OSAPI_Semaphore_give* fails when called.
- **Actions:**
  - call *OSAPI_Semaphore_give*.
- **Check that:**
  - *OSAPI_Semaphore_give* returns *RTI_FALSE*.

8.4 Process

QTS-R-524

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_Process_getpid* operation has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the *OSAPI_Process_getpid* operation has the following signature:

Operations	Parameters	Type
OSAPI_Process_getpid	*void*	*OSAPI_ProcessId*

QTS-R-524.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_Process_getpid* operation returns the process ID of the caller.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call the *OSAPI_Process_getpid* operation.
- **Check that:**
  - the process ID of the caller is returned.

8.5 Memory

QTS-R-525.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_copy* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Memory_copy* has the following signature:

Operations	Parameters	Type
OSAPI_Memory_copy	out: *void* *dest in: *const void* *src in: *RTI_SIZE_T* size	*void*

QTS-R-525.1.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_copy* copies memory successfully.

TC.1

- **Preconditions:**
  - *src* and *dest* are non-overlapping memory regions of *size* bytes.
- **Actions:**
  - call *OSAPI_Memory_copy*.
- **Check that:**
  - The *size* bytes currently in *dest* are a copy of the *size* bytes in *src*.

QTS-R-525.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_zero* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Memory_zero* has the following signature:

Operations	Parameters	Type
OSAPI_Memory_zero	out: *void* *mem in: *RTI_SIZE_T* size	*void*

QTS-R-525.2.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_zero* zeroes memory correctly.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call *OSAPI_Memory_zero*.
- **Check that:**
  - Every byte in the memory region [*mem*, *mem* + *size* - 1] is set to zero.

QTS-R-525.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_compare* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Memory_compare* has the following signature:

Operations	Parameters	Type
OSAPI_Memory_compare	in: *const void* *left in: *const void* *right in: *RTI_SIZE_T* size	*RTI_INT32*

QTS-R-525.3.1

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_compare* returns that *left* is less than *right*.

TC.1

- **Preconditions:**
  - the memory region defined by [*left*, *left* + *size* - 1] is less than the one defined by [*right*, *right* + *size* - 1] when interpreted as a sequence of 8-bit values.
- **Actions:**
  - call *OSAPI_Memory_compare*.
- **Check that:**
  - *OSAPI_Memory_compare* returns a value less than zero.

QTS-R-525.3.2

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_compare* returns that *left* is equal to *right*.

TC.1

- **Preconditions:**
  - the memory region defined by [*left*, *left* + *size* - 1] is equal to the one defined by [*right*, *right* + *size* - 1] when interpreted as a sequence of 8-bit values.
- **Actions:**
  - call *OSAPI_Memory_compare*.
- **Check that:**
  - *OSAPI_Memory_compare* returns zero.

QTS-R-525.3.3

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_compare* returns that *left* is greater than *right*.

TC.1

- **Preconditions:**
  - the memory region defined by [*left*, *left* + *size* - 1] is greater than the one defined by [*right*, *right* + *size* - 1] when interpreted as a sequence of 8-bit values.
- **Actions:**
  - call *OSAPI_Memory_compare*.
- **Check that:**
  - *OSAPI_Memory_compare* returns a value greater than zero.

QTS-R-525.4

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_move* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Memory_move* has the following signature:

Operations	Parameters	Type
OSAPI_Memory_move	out: *void* *dest in: *const void* *source in: *RTI_SIZE_T* size	*void*

QTS-R-525.4.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_move* successfully moves overlapping memory.
- [**TC.2**] Operation *OSAPI_Memory_move* successfully moves non-overlapping memory.

TC.1

- **Preconditions:**
  - *source* and *dest* are distinct overlapping *size*-byte memory regions.
    - [TC.1.1] *source* begins within *dest*.
    - [TC.1.2] *dest* begins within *source*.
- **Actions:**
  - call *OSAPI_Memory_move*.
- **Check that:**
  - the *size* bytes currently stored in *dest* are equal to the bytes initially stored in *source*.

TC.2

- **Preconditions:**
  - *source* and *dest* are non-overlapping *size*-byte memory regions.
- **Actions:**
  - call *OSAPI_Memory_move*.
- **Check that:**
  - the *size* bytes currently stored in *dest* are equal to the bytes initially stored in *source*.

QTS-R-525.5

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_fndchr* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_Memory_fndchr* has the following signature:

Operations	Parameters	Type
OSAPI_Memory_fndchr	in: *const void* *s in: *RTI_INT32* c in: *RTI_SIZE_T* size	*void* *

QTS-R-525.5.1

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_Memory_fndchr* successfully finds the character *c*.

TC.1

- **Preconditions:**
  - the memory region defined as [*s*, *s* + *size*- 1] contains at least one instance of the byte *c*.
- **Actions:**
  - call *OSAPI_Memory_fndchr*.
- **Check that:**
  - *OSAPI_Memory_fndchr* returns a pointer to the first instance of the byte *c*.

QTS-R-525.5.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_Memory_fndchr* fails to find the character *c*.

TC.1

- **Preconditions:**
  - the memory region defined as [*s*, *s* + *size* - 1] does not contain an instance of the byte *c*.
- **Actions:**
  - call *OSAPI_Memory_fndchr*.
- **Check that:**
  - *OSAPI_Memory_fndchr* returns NULL.

8.6 String

QTS-R-525.6

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_length* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_String_length* has the following signature:

Operations	Parameters	Type
OSAPI_String_length	in: *const char* *s	*RTI_SIZE_T*

QTS-R-525.6.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_length* returns the length of a string.

TC.1

- **Preconditions:**
  - *s* is an ASCIIZ string.
- **Actions:**
  - call *OSAPI_String_length*.
- **Check that:**
  - *OSAPI_String_length* returns the length of *s*, excluding the NUL terminator.

QTS-R-525.7

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_cmp* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_String_cmp* has the following signature:

Operations	Parameters	Type
OSAPI_String_cmp	in: *const char* *l in: *const char* *r	*RTI_INT32*

QTS-R-525.7.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_cmp* returns less than 0 when *l* is less than *r*.

TC.1

- **Preconditions:**
  - the ASCIIZ string *l* is lexicographically less than the ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_cmp*.
- **Check that:**
  - *OSAPI_String_cmp* returns a value less than 0.

QTS-R-525.7.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_cmp* returns 0 when *l* is equal to *r*.

TC.1

- **Preconditions:**
  - the ASCIIZ string *l* is lexicographically equal to the ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_cmp*.
- **Check that:**
  - *OSAPI_String_cmp* returns 0.

QTS-R-525.7.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_cmp* returns greater than 0 when *l* is greater than *r*.

TC.1

- **Preconditions:**
  - the ASCIIZ string *l* is lexicographically greater than the ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_cmp*.
- **Check that:**
  - *OSAPI_String_cmp* returns a value greater than 0.

QTS-R-525.8

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_ncmp* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_String_ncmp* has the following signature:

Operations	Parameters	Type
OSAPI_String_ncmp	in: *const char* *l in: *const char* *r in: *RTI_SIZE_T* num	*RTI_INT32*

QTS-R-525.8.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_ncmp* returns less than 0 when the first *num* bytes of *l* are less than the first *num* bytes of *r*.
- [**TC.2**] Operation *OSAPI_String_ncmp* returns less than 0 when *l* is shorter than *r* and has less than *num* bytes.

TC.1

- **Preconditions:**
  - *l* and *r* are longer than *num*.
  - the first *num* bytes of ASCIIZ string *l* are lexicographically less than the first *num* bytes of ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns a value less than 0.

TC.2

- **Preconditions:**
  - *l* is shorter than *r* and *num*.
  - the bytes of *l* and *r*, until reaching *l*'s NUL terminator, are lexicographically equal.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns a value less than 0.

QTS-R-525.8.2

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_ncmp* returns 0 when the first *num* bytes of *l* are equal to the first *num* bytes of *r*.
- [**TC.2**] Operation *OSAPI_String_ncmp* returns 0 when *l* and *r* have less than *num* bytes and are equal.

TC.1

- **Preconditions:**
  - *l* and *r* are longer than *num*.
  - the first *num* bytes of ASCIIZ string *l* is lexicographically equal to the first *num* bytes of ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns 0.

TC.2

- **Preconditions:**
  - *l* and *r* are shorter than *num*.
  - the ASCIIZ string *l* is lexicographically equal to the ASCIIZ string *r*, up to and including their NUL terminators, when taken as a sequence of unsigned 8-bit values.
  - [**TC.2.1**] the bytes following the NUL terminator of *l* and *r*, until *num* bytes from the start of each string, are not equal.
  - [**TC.2.2**] the bytes following the NUL terminator of *l* and *r*, until *num* bytes from the start of each string, are equal.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns 0.

QTS-R-525.8.3

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_String_ncmp* returns greater than 0 when the first *num* bytes of *l* are greater than than the first *num* bytes of *r*.
- [**TC.2**] Operation *OSAPI_String_ncmp* returns greater than 0 when *r* is shorter than *l* and has less than *num* bytes.

TC.1

- **Preconditions:**
  - *l* and *r* are longer than *num*.
  - the first *num* bytes of ASCIIZ string *l* are lexicographically greater than the first *num* bytes of ASCIIZ string *r* when taken as a sequence of unsigned 8-bit values.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns a value greater than 0.

TC.2

- **Preconditions:**
  - *r* is shorter than *l* and *num*.
  - the bytes of *l* and *r*, until reaching *r*'s NUL terminator, are lexicographically equal.
- **Actions:**
  - call *OSAPI_String_ncmp*.
- **Check that:**
  - *OSAPI_String_ncmp* returns a value greater than 0.

8.7 OSAPI_System

QTS-R-526.0

Analysis

The requirement is met when an instance of the *OSAPI_SystemI* structure implements:

- [**TC.1**] *get_timer_resolution* operation.
- [**TC.2**] *get_time* operation.
- [**TC.3**] *start_timer* operation.
- [**TC.4**] *generate_uuid* operation.
- [**TC.5**] *get_hostname* operation.
- [**TC.6**] *initialize* operation.
- [**TC.7**] *get_ticktime* operation.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *get_timer_resolution* operation returning *RTI_INT32* value and not accepting any parameters.

TC.2

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *get_time* operation returning *RTI_BOOL* value and accepting the following parameters:
    - in: OSAPI_NtpTime *now

TC.3

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *start_timer* operation returning *RTI_BOOL* value and accepting the following parameters:
    - in: OSAPI_Timer_T self
    - in: OSAPI_TimerTickHandlerFunction tick_handler

TC.4

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *generate_uuid* operation returning *RTI_BOOL* value and accepting the following parameters:
    - out: *struct* OSAPI_SystemUUID *uuid_out

TC.5

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *get_hostname* operation returning *RTI_BOOL* value and accepting the following parameters:
    - out: char **const* hostname

TC.6

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *initialize* operation returning *RTI_BOOL* value and not accepting any parameters.

TC.7

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SystemI* structure implements the *get_ticktime* operation returning *RTI_BOOL* value and accepting the following parameters:
    - out: *RTI_INT32* *sec
    - out: *RTI_UINT32* *nanosec

QTS-R-526.1

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.start_timer* operation configures the *OSAPI_System* instance to start invoking the *tick_handler* method parameter on the self *OSAPI_Timer_T* at the frequency returned by the *OSAPI_SystemI.get_timer_resolution* operation, and returns *RTI_TRUE* on success.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.get_timer_resolution* operation can be invoked successfully.
- **Actions:**
  - call the *OSAPI_SystemI.start_timer* method.
- **Check that:**
  - *OSAPI_System* instance is configured to start invoking the *tick_handler* method parameter on the *self OSAPI_Timer_T* at the frequency returned by the *OSAPI_SystemI.get_timer_resolution* operation, and
  - *RTI_TRUE* is returned.

QTS-R-526.1.1

Analysis

The requirement is met when the *OSAPI_SystemI.start_timer* operation does not configure the OSAPI_System instance to start invoking the *tick_handler* method and returns *RTI_FALSE*:

- [**TC.1**] on failure.
- [**TC.2**] when the system has not been initialized.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.start_timer* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.start_timer* operation.
- **Check that:**
  - the *OSAPI_SystemI.start_timer* operation does not configure the OSAPI_System instance to start invoking the *tick_handler* method, and
  - the *OSAPI_SystemI.start_timer* operation returns *RTI_FALSE*.

TC.2

- **Preconditions:**
  - the system has not been initialized.
- **Actions:**
  - call the *OSAPI_SystemI.start_timer* operation.
- **Check that:**
  - the *OSAPI_SystemI.start_timer* operation does not configure the OSAPI_System instance to start invoking the *tick_handler* method, and
  - the *OSAPI_SystemI.start_timer* operation returns *RTI_FALSE*.

QTS-R-526.1.2

Analysis

The requirement is met when:

- [**TC.1**] The Platform Integration calls the *tick_handler* method for each of the *OSAPISYSTEM_MAX_TIMERS* instances at least at the timer resolution rate returned by the *OSAPI_SystemI.get_timer_resolution* operation.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - The Platform Integration calls the *tick_handler* method for each of the *OSAPISYSTEM_MAX_TIMERS* instances at least at the timer resolution rate returned by the *OSAPI_SystemI.get_timer_resolution* operation.

QTS-R-526.2

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_timer_resolution* operation returns a value greater than 0 (zero) for the resolution of the clock used by the system timer.

TC.1

- **Preconditions:**
  - the system has been initialized.
- **Actions:**
  - call the *OSAPI_SystemI.get_timer_resolution* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_timer_resolution* operation returns a value greater than 0 (zero) for the resolution of the clock used by the system timer.

QTS-R-526.3

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_timer_resolution* operation returns 0 if the system has not been initialized.

TC.1

- **Preconditions:**
  - the system has not been initialized.
- **Actions:**
  - call the *OSAPI_SystemI.get_timer_resolution* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_timer_resolution* operation returns 0.

QTS-R-526.4

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_time* operation updates the memory pointed to by the *now* pointer and returns *RTI_TRUE* on success.
- [**TC.2**] The *OSAPI_SystemI.get_time* operation returns the current system time in NtpTime format.

TC.1

- **Preconditions:**
  - a successful call of the *OSAPI_SystemI.get_time* operation can be performed.
- **Actions:**
  - call the *OSAPI_SystemI.get_time* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_time* operation updates the memory pointed to by the *now* pointer, and
  - the *OSAPI_SystemI.get_time* operation returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - a successful call of the *OSAPI_SystemI.get_time* operation can be performed.
- **Actions:**
  - call the *OSAPI_SystemI.get_time* operation.
- **Check that:**
  - the *now* pointer points to the current system time in NtpTime format.

QTS-R-526.5

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_time* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.get_time* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.get_time* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_time* operation returns *RTI_FALSE*.

QTS-R-526.6

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.initialize* operation initializes the system and returns *RTI_TRUE*.

TC.1

- **Preconditions:**
  - It is possible to initialize the system.
- **Actions:**
  - call the *OSAPI_SystemI.initialize* operation.
- **Check that:**
  - the system is intiailized, and
  - the *OSAPI_SystemI.initialize* operation returns *RTI_TRUE*.

QTS-R-526.7

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.initialize* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.initialize* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.initialize* operation.
- **Check that:**
  - the *OSAPI_SystemI.initialize* operation returns *RTI_FALSE*.

QTS-R-526.8

Analysis

The requirement is met when the operation *OSAPI_SystemI.generate_uuid* :

- [**TC.1**] updates the memory pointed to by the *uuid_out* pointer and returns *RTI_TRUE* on success.
- [**TC.2**] provides a system id that is unique in the system and returns it through the *uuid_out* pointer.

TC.1

- **Preconditions:**
  - it is possible to provide a system id.
- **Actions:**
  - call the *OSAPI_SystemI.generate_uuid* operation.
- **Check that:**
  - the memory pointed to by the *uuid_out* pointer gets updated, and
  - the *OSAPI_SystemI.generate_uuid* operation returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - it is possible to provide a system id.
- **Actions:**
  - call the *OSAPI_SystemI.generate_uuid* operation.
- **Check that:**
  - the *uuid_out* pointer points to an unique id in the system.

QTS-R-526.9

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.generate_uuid* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.generate_uuid* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.generate_uuid* operation.
- **Check that:**
  - the *OSAPI_SystemI.generate_uuid* operation returns *RTI_FALSE*.

QTS-R-526.10

Analysis

The requirement is met when the *OSAPI_SystemI.get_hostname* operation, on successful call, returns a hostname and *RTI_TRUE* value when:

- [**TC.1**] the length of the hostname is less than or equal to *OSAPI_SYSTEM_MAX_HOSTNAME* bytes.
- [**TC.2**] the length of the hostname is greater than *OSAPI_SYSTEM_MAX_HOSTNAME* bytes.

TC.1

- **Preconditions:**
  - It is possible to get the hostname, and
    - [**TC.1.1**] The hostname in bytes is less than *OSAPI_SYSTEM_MAX_HOSTNAME* bytes,
    - [**TC.1.2**] The hostname in bytes is equal to *OSAPI_SYSTEM_MAX_HOSTNAME* bytes.
- **Actions:**
  - call the *OSAPI_SystemI.get_hostname* operation.
- **Check that:**
  - the hostname is provided through the *hostname* pointer, and
  - the *OSAPI_SystemI.get_hostname* operation returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - It is possible to get the hostname, and
  - the hostname length is greater than *OSAPI_SYSTEM_MAX_HOSTNAME* bytes.
- **Actions:**
  - call the *OSAPI_SystemI.get_hostname* operation.
- **Check that:**
  - only the *OSAPI_SYSTEM_MAX_HOSTNAME* bytes of the hostname are provided through the *hostname* pointer, and
  - the *OSAPI_SystemI.get_hostname* operation returns *RTI_TRUE*.

QTS-R-526.11

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_hostname* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.get_hostname* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.get_hostname* operation.
- **Check that:**
  - *RTI_FALSE* is returned.

QTS-R-526.13

Analysis

The requirement is met when the *OSAPI_SystemI.get_ticktime* operation on success:

- [**TC.1**] updates the memory pointed to by the *sec* and *nanosec* pointers and returns *RTI_TRUE*.
- [**TC.2**] returns the number of seconds and nanoseconds through the *sec* and *nanosec* pointers in increments of the timer resolution since the system started.

TC.1

- **Preconditions:**
  - it is possible to get the system tick time.
- **Actions:**
  - call the *OSAPI_SystemI.get_ticktime* operation.
- **Check that:**
  - the memory regions pointed to by the *sec* and *nanosec* pointers are updated, and
  - the *OSAPI_SystemI.get_ticktime* operation returns *RTI_TRUE*.

TC.2

- **Preconditions:**
  - it is possible to get the system tick time.
- **Actions:**
  - call the *OSAPI_SystemI.get_ticktime* operation.
- **Check that:**
  - the time returned by the *OSAPI_SystemI.get_ticktime* operation through the *sec* and *nanosec* pointers represents the time since the system started in the increments of the system timer resolution.

QTS-R-526.14

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_SystemI.get_ticktime* operation returns *RTI_FALSE* on failure,
- [**TC.2**] The *OSAPI_SystemI.get_ticktime* operation returns *RTI_FALSE* when the system has not been initialized.

TC.1

- **Preconditions:**
  - the *OSAPI_SystemI.get_ticktime* operation fails.
- **Actions:**
  - call the *OSAPI_SystemI.get_ticktime* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_ticktime* operation returns *RTI_FALSE*.

TC.2

- **Preconditions:**
  - the system has not been initialized.
- **Actions:**
  - call the *OSAPI_SystemI.get_ticktime* operation.
- **Check that:**
  - the *OSAPI_SystemI.get_ticktime* operation returns *RTI_FALSE*.

QTS-R-526.15

Analysis

The requirement is met when:

- [**TC.1**] The *OSAPI_System_get_native_interface* operation has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the *OSAPI_System_get_native_interface* operation has the following signature:

Operations	Parameters	Type
OSAPI_System_get_native_interface	out: *OSAPI_SystemI* *intf	*void*

QTS-R-526.15.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_System_get_native_interface* operation returns a pointer to an *OSAPI_SystemI* object through the *intf* parameter, that implements the *OSAPI_SystemI* interface.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call the *OSAPI_System_get_native_interface* operation.
- **Check that:**
  - the pointer returned by the *OSAPI_System_get_native_interface* operation through the *intf* parameter, points to an object implementing the *OSAPI_SystemI* interface.

QTS-R-526.15.2

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_System_get_native_interface* operation returns without updating the parameter *intf* if *intf* is 'nil'.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call the *OSAPI_System_get_native_interface* operation providing 'nil' as the *intf* parameter.
- **Check that:**
  - the function returns without updating the *intf* paramteter.

QTS-R-526.16

Analysis

The requirement is met when:

- [**TC.1**] the global variable *OSAPI_System_gv_System* is defined.
- [**TC.2**] the global variable *OSAPI_System_gv_System* is declared as a pointer to a type OSAPI_System.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the global variable *OSAPI_System_gv_System* is defined.

TC.2

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the global variable *OSAPI_System_gv_System* is declared as a pointer to a type OSAPI_System.

QTS-R-526.17

Analysis

The requirement is met when:

- [**TC.1**] the PSL has an instance of the OSAPI_System object.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the PSL has an instance of the OSAPI_System object.

QTS-R-526.18

Analysis

The requirement is met when:

- [**TC.1**] the PSL initializes its instance of the OSAPI_System object with OSAPI_System_INITIALIZER.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the PSL initializes its instance of the OSAPI_System object with OSAPI_System_INITIALIZER.

8.8 Concurrency

QTS-R-527.1

Analysis

The requirement is met when:

- [**TC.1**] The listed PSL operations are reentrant and thread-safe.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the following PSL operations are reentrant and thread safe:
    - *OSAPI_Heap_allocate_buffer*
    - *OSAPI_Mutex_new*
    - *OSAPI_Mutex_T*ake
    - *OSAPI_Mutex_give*
    - *OSAPI_Semaphore_new*
    - *OSAPI_Semaphore_take*
    - *OSAPI_Semaphore_give*
    - *OSAPI_Process_getpid*
    - *OSAPI_Memory_copy*
    - *OSAPI_Memory_zero*
    - *OSAPI_Memory_compare*
    - *OSAPI_Memory_fndchr*
    - *OSAPI_Memory_move*
    - *OSAPI_String_length*
    - *OSAPI_String_cmp*
    - *OSAPI_String_ncmp*
    - *OSAPI_SystemI.get_timer_resolution*
    - *OSAPI_SystemI.get_time*
    - *OSAPI_SystemI.start_timer*

- *OSAPI_SystemI.generate_uuid*
- *OSAPI_SystemI.get_hostname*
- *OSAPI_SystemI.initialize*
- *OSAPI_SystemI.get_ticktime*

8.9 Debug support

QTS-R-528.1

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_get_last_error_code operation has specified signature when debugging is enabled

TC.1

- **Preconditions:**
  - debugging is enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_get_last_error_code has the following signature:

Operations	Parameters	Type
OSAPI_Log_get_last_error_code	in: *void*	*RTI_INT32*

QTS-R-528.1.1

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_get_last_error_code operation returns the last error code tracked by the platform

TC.1

- **Preconditions:**
  - debugging is enabled
  - there are at least two different error codes tracked by the platform
- **Actions:**
  - call OSAPI_Log_get_last_error_code
- **Check that:**
  - OSAPI_Log_get_last_error_code operation returns the last error code tracked by the platform

QTS-R-528.1.2

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_get_last_error_code operation is defined when debugging is enabled.
- [**TC.2**] the OSAPI_Log_get_last_error_code operation is not defined when debugging is not enabled.

TC.1

- **Preconditions:**
  - debugging is enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_get_last_error_code is defined

TC.2

- **Preconditions:**
  - debugging is not enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_get_last_error_code is not defined

QTS-R-528.10

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_set_last_error_code operation has specified signature when debugging is enabled

TC.1

- **Preconditions:**
  - debugging is enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_set_last_error_code has the following signature:

Operations	Parameters	Type
OSAPI_Log_set_last_error_code	in: *RTI_INT32* err	*void*

QTS-R-528.10.1

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_set_last_error_code operation sets the last error code tracked by the platform to the value of the parameter *err*

TC.1

- **Preconditions:**
  - debugging is enabled
  - there are at least two different error codes tracked by the platform
- **Actions:**
  - call OSAPI_Log_set_last_error_code with *err*
- **Check that:**
  - OSAPI_Log_set_last_error_code operation sets the last error code tracked by the platform to the value of the parameter *err*

QTS-R-528.10.2

Analysis

The requirement is met when:

- [**TC.1**] the OSAPI_Log_set_last_error_code operation is defined when debugging is enabled.
- [**TC.2**] the OSAPI_Log_set_last_error_code operation is not defined when debugging is not enabled.

TC.1

- **Preconditions:**
  - debugging is enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_set_last_error_code is defined

TC.2

- **Preconditions:**
  - debugging is not enabled
- **Actions:**
  - N/A
- **Check that:**
  - OSAPI_Log_set_last_error_code is not defined

9 PSL NETIO Datagram (DGRAM) Test Specifications

9.1 DGRAM Transport Plugin API

QTS-R-550.1

Analysis

The requirement is met when:

- [**TC.1**] An instance of the NETIO_DGRAM_InterfaceI structure is created and its member attributes have assigned function pointers that implement interfaces as per the table.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - an instance of the NETIO_DGRAM_InterfaceI structure is created and its member attributes have been assigned function pointers to functions that implement the interfaces as per the table below:

Attribute	Parameters	Type
create_instance	in: *NETIO_Interface_T* *upstream in: *void* *property	*NETIO_Interface_T**
get_interface_list	in: *NETIO_Interface_T* *user_intf inout: *struct NETIO_DGRAM_InterfaceTableEntrySeq* *if_table	*RTI_BOOL*
release_address	in: *NETIO_Interface_T* *self in: *struct NETIO_Address* *src_addr	*RTI_BOOL*
resolve_address	in: *NETIO_Interface_T* *netio_intf in: *const struct NETIO_DGRAM_InterfaceTableEntry* *if_entry in: *const* char *address_string out: *struct NETIO_Address* *address_value out: *RTI_BOOL* *invalid	*RTI_BOOL*
send	in: *NETIO_Interface_T* *self in: *struct NETIO_Interface* *source in: *struct NETIO_Address* *destination in: *NETIO_Packet_T* *packet	*RTI_BOOL*
get_route_table	in: *NETIO_Interface_T* *netio_intf inout: *struct NETIO_Address*Seq *address inout: *struct* NETIO_NetmaskSeq *netmask	*RTI_BOOL*
bind_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source	*RTI_BOOL*

QTS-R-550.1.1

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.create_instance* operation returns a non-NULL pointer on success

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.create_instance* operation succeeds when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.create_instance* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.create_instance* operation returns a non-NULL pointer to an instance of an interface of type *NETIO_Interface_T*

QTS-R-550.1.1.1

Analysis

The requirement is met when:

- [**TC.1**] The PSL does not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *NETIO_DGRAM_InterfaceI.create_instance.*

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the PSL does not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *NETIO_DGRAM_InterfaceI.create_instance.*

QTS-R-550.1.2

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.create_instance* operation returns NULL on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.create_instance* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.create_instance* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.create_instance* operation returns NULL

QTS-R-550.1.3

Analysis

The requirement is met when the *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns the available network interfaces, that meet the specified cirteria, and *RTI_TRUE* on success.

The criteria are met when the following logic is met:

A and **B** and **C** and (**D1** or **D2**)

Where each criterion is defined as follows:

A - <*entry.address*> is not within <*entry.multicast_group*>

B - <*entry.multicast_group.netmask.bits*> is not less than 0 and not greater than 128

C - <*entry.multicast_group.netmask.mask*> all array elements are zeroes

D1 - <*entry.locator_kind*> is greater than 0 and less than *NETIO_ADDRESS*_RTPS_RESERVED_LOW

D2 - <*entry.locator_kind*> is greater than *NETIO_ADDRESS*_RTPS_RESERVED_HIGH

test case	A	B	C	D1	D2
TC.1	T	T	T	T	F
TC.2	T	T	T	F	T

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation succeeds when called
  - there are at least two available interfaces in which at least one, but not all, meet the criteria:
    - the interfaces only meet the criteria in the following way:
      - **A** - <*entry.address*> is not within <*entry.multicast_group*>
      - **B** - <*entry.multicast_group.netmask.bits*> is not less than 0 and not greater than 128
      - **C** - <*entry.multicast_group.netmask.mask*> all array elements are zeroes
      - **D1** - <*entry.locator_kind*> is greater than 0 and less than *NETIO_ADDRESS*_RTPS_RESERVED_LOW
      - **!D2** - <*entry.locator_kind*> is NOT greater than *NETIO_ADDRESS*_RTPS_RESERVED_HIGH
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_interface_list* operation
- **Check that:**
  - the available network interfaces, that meet the criteria, for the NETIO_DGRAM interface passed in *user_intf* are returned through the *if_table* parameter
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_TRUE*

TC.2

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation succeeds when called
  - there are at least two available interfaces in which at least one, but not all, meet the criteria:
    - the interfaces only meet the criteria in the following way:
      - **A** - <*entry.address*> is not within <*entry.multicast_group*>
      - **B** - <*entry.multicast_group.netmask.bits*> is not less than 0 and not greater than 128
      - **C** - <*entry.multicast_group.netmask.mask*> all array elements are zeroes
      - **!D1** - <*entry.locator_kind*> is NOT greater than 0 or NOT less than *NETIO_ADDRESS*_RTPS_RESERVED_LOW
      - **D2** - <*entry.locator_kind*> is greater than *NETIO_ADDRESS*_RTPS_RESERVED_HIGH
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_interface_list* operation
- **Check that:**
  - the available network interfaces, that meet the criteria, for the NETIO_DGRAM interface passed in *user_intf* are returned through the *if_table* parameter
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_TRUE*

QTS-R-550.1.3.1

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_TRUE* without updating the *if_table* sequence when there are no available interfaces.

TC.1

- **Preconditions:**
  - there are no interfaces available
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_interface_list* operation
- **Check that:**
  - the *if_table* sequence is not updated
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_TRUE*

QTS-R-550.1.4

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_interface_list* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.get_interface_list* operation returns *RTI_FALSE*

QTS-R-550.1.5

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.release_address* operation causes interface to stop listening to messages from the address and returns *RTI_TRUE* on success.

TC.1

- **Preconditions:**
  - the NETIO_DGRAM interface specified by *self* has been listening to messages from the address *src_addr*
  - the *NETIO_DGRAM_InterfaceI.release_address* operation succeeds when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.release_address* operation
- **Check that:**
  - the NETIO_DGRAM interface specified by *self* stopped listening to messages from the address *src_addr*
  - the *NETIO_DGRAM_InterfaceI.release_address* operation returns *RTI_TRUE*

QTS-R-550.1.6

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.release_address* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.release_address* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.release_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.release_address* operation returns *RTI_FALSE*

QTS-R-550.1.7

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.resolve_address* operation converts an address that is a valid address for the NETIO_DGRAM interface and returns *RTI_TRUE* on successful conversion.

TC.1

- **Preconditions:**
  - the *address_string* is a valid address for the NETIO_DGRAM interface specified by *netio_intf*
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation converts address successfully when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.resolve_address* operation
- **Check that:**
  - the *address_string* is converted to an *address_value* using the locator kind information stored in the interface table *if_entry*
  - the value pointed by *invalid* is set to *RTI_FALSE*
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation returns *RTI_TRUE*

QTS-R-550.1.7.1

Analysis

The requirement is met when:

- [**TC.1**] The NETIO_DGRAM_InterfaceI.resolve_address operation converts the *address_string* from dotted notation "a.b.c.d" to *address_value.value.address.octet* when the locator_kind of the interface entry *if_entry* is *NETIO_ADDRESS*_KIND_UDPv4 and the *address_string* is an IPv4 address

TC.1

- **Preconditions:**
  - the locator_kind of the interface entry *if_entry* is *NETIO_ADDRESS*_KIND_UDPv4
  - the *address_string* is an IPv4 address
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.resolve_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation converts the *address_string* from the dotted notation "a.b.c.d" to *address_value.value.address.octet* in the following form in network order:

```
Byte    |0    |1    |2    |3    |4    |5    |6    |7    |8    |9    |10   |11   |12   |13   |14   |15   |16
        +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
        |  a  |  b  |  c  |  d  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
        +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

QTS-R-550.1.8

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.resolve_address* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.resolve_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation returns *RTI_FALSE*

QTS-R-550.1.8.1

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.resolve_address* operation sets the value pointed by *invalid* to *RTI_TRUE* on a failure caused by *address_string* being an invalid address for *netio_intf*.
- [**TC.2**] The *NETIO_DGRAM_InterfaceI.resolve_address* operation sets the value pointed by *invalid* to *RTI_FALSE* on a failure caused by other reason than *address_string* being an invalid address for *netio_intf*.

TC.1

- **Preconditions:**
  - the *address_string* is an invalid address for the NETIO_DGRAM interface specified by *netio_intf*
  - when called, the *NETIO_DGRAM_InterfaceI.resolve_address* operation fails due to *address_string* being invalid address for the *netio_intf*
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.resolve_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation sets the value pointed by *invalid* to *RTI_TRUE*

TC.2

- **Preconditions:**
  - the *address_string* is a valid address for the NETIO_DGRAM interface specified by *netio_intf*
  - when called, the *NETIO_DGRAM_InterfaceI.resolve_address* operation fails due to other reason than *address_string* being invalid address for the *netio_intf*
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.resolve_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.resolve_address* operation sets the value pointed by *invalid* to *RTI_FALSE*

QTS-R-550.1.9

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.send* operation sends the packet from *source* to the *destination* using *self* and returns *RTI_TRUE* on success.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.send* operation succeeds when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.send* operation
- **Check that:**
  - the *packet* is sent from *source* to the *destination* using the *NETIO_Interface_T* object *self*
  - the *NETIO_DGRAM_InterfaceI.send* operation returns *RTI_TRUE*

QTS-R-550.1.10

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.send* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.send* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.send* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.send* operation returns *RTI_FALSE*

QTS-R-550.1.11

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.get_route_table* operation retrieves a sequence of all addresses and a sequence of all netmasks from the routing table and returns *RTI_TRUE* on success.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.get_route_table* operation succeeds when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_route_table* operation
- **Check that:**
  - a sequence of all addresses and a sequence of all netmasks from the routing table, that the NETIO_DGRAM interface can send to, are retrieved as *address* and *netmask* respectively
  - the *NETIO_DGRAM_InterfaceI.get_route_table* operation returns *RTI_TRUE*

QTS-R-550.1.12

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.get_route_table* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.get_route_table* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.get_route_table* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.get_route_table* operation returns *RTI_FALSE*

QTS-R-550.1.13

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.bind_address* operation causes interface to listen for messages from the addresses and returns *RTI_TRUE* on success.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.bind_address* operation succeeds when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.bind_address* operation
- **Check that:**
  - the interface specified by *user_intf* listens for messages from the address specified by *source*
  - the *NETIO_DGRAM_InterfaceI.bind_address* operation returns *RTI_TRUE*

QTS-R-550.1.14

Analysis

The requirement is met when:

- [**TC.1**] The *NETIO_DGRAM_InterfaceI.bind_address* operation returns *RTI_FALSE* on failure.

TC.1

- **Preconditions:**
  - the *NETIO_DGRAM_InterfaceI.bind_address* operation fails when called
- **Actions:**
  - call the *NETIO_DGRAM_InterfaceI.bind_address* operation
- **Check that:**
  - the *NETIO_DGRAM_InterfaceI.bind_address* operation returns *RTI_FALSE*

QTS-R-550.1.15

Analysis

The requirement is met when:

- [**TC.1**] The Platform Integrator uses *NETIO_DGRAM_Interface_upstream_receive* operation to forward a *NETIO_Packet_T packet* and a *NETIO_Address source* to the upstream interface specified by the *netio_intf* parameter upon reception of data from the platform's network stack.

TC.1

- **Preconditions:**
  - the platform received data on the network stack
- **Actions:**
  - N/A
- **Check that:**
  - the Platform Integrator uses the *NETIO_DGRAM_Interface_upstream_receive* operation to forward a *NETIO_Packet_T packet* that contains payload data received from the network and a *NETIO_Address source* to the upstream interface specified by the *netio_intf* parameter

9.2 Concurrency

QTS-R-551.1

Analysis

The requirement is met when:

- [**TC.1**] The listed functions are reentrant and thread-safe.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the listed functions are reentrant and thread-safe:
    - *NETIO_DGRAM_InterfaceI.create_instance*
    - *NETIO_DGRAM_InterfaceI.get_interface_list*
    - *NETIO_DGRAM_InterfaceI.release_address*
    - *NETIO_DGRAM_InterfaceI.resolve_address*
    - *NETIO_DGRAM_InterfaceI.send*
    - *NETIO_DGRAM_InterfaceI.get_route_table*
    - *NETIO_DGRAM_InterfaceI.bind_address*

10 PSL Data Types Test Specifications

QTS-R-560.1

Analysis

The requirement is met when the following data types have their minimum alignment met, and are accessed atomically where noted:

- [**TC.1**] RTI_INT8, minimum alignment 8 bit, atomic access.
- [**TC.2**] RTI_UINT8, minimum alignment 8 bit, atomic access.
- [**TC.3**] RTI_INT16, minimum alignment 16 bit, atomic access.
- [**TC.4**] RTI_UINT16, minimum alignment 16 bit, atomic access.
- [**TC.5**] *RTI_INT32*, minimum alignment 32 bit, atomic access.
- [**TC.6**] *RTI_UINT32*, minimum alignment 32 bit, atomic access.
- [**TC.7**] RTI_INT64, minimum alignment 32 bit.
- [**TC.8**] RTI_UINT64, minimum alignment 32 bit.
- [**TC.9**] RTI_FLOAT32, minimum alignment 32 bit.
- [**TC.10**] RTI_DOUBLE64, minimum alignment 32 bit.
- [**TC.11**] RTI_DOUBLE128, minimum alignment 32 bit.
- [**TC.12**] *RTI_BOOL*, minimum alignment 32 bit, atomic access.

- [**TC.13**] RTI_SIZE_T, minimum alignment 32 bit, atomic access.
- [**TC.14**] OSAPI_Alignment_T, minimum alignment 32 bit, atomic access.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_INT8 data type:
    - has minimum required alignment (in bits): 8,
    - is accessed atomically.

TC.2

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_UINT8 data type:
    - has minimum required alignment (in bits): 8,
    - is accessed atomically.

TC.3

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_INT16 data type:
    - has minimum required alignment (in bits): 16,
    - is accessed atomically.

TC.4

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_UINT16 data type:
    - has minimum required alignment (in bits): 16,
    - is accessed atomically.

TC.5

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the *RTI_INT32* data type:
    - has minimum required alignment (in bits): 32,
    - is accessed atomically.

TC.6

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the *RTI_UINT32* data type:
    - has minimum required alignment (in bits): 32,
    - is accessed atomically.

TC.7

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_INT64 data type has minimum required alignment (in bits): 32.

TC.8

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_UINT64 data type has minimum required alignment (in bits): 32.

TC.9

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_FLOAT32 data type has minimum required alignment (in bits): 32.

TC.10

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_DOUBLE64 data type has minimum required alignment (in bits): 32.

TC.11

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_DOUBLE128 data type has minimum required alignment (in bits): 32.

TC.12

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the *RTI_BOOL* data type:
    - has minimum required alignment (in bits): 32,
    - is accessed atomically.

TC.13

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the RTI_SIZE_T data type:
    - has minimum required alignment (in bits): 32,
    - is accessed atomically.

TC.14

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the OSAPI_Alignment_T data type:
    - has minimum required alignment (in bits): 32,
    - is accessed atomically.

11 PSL Type Plugin Test Specifications

QTS-R-700.0

Analysis

The requirement is met when:

- [**TC.1**] an instance of the NDDS_Type_Plugin structure is created with the specified member attributes assigned function pointers to functions that implement the interfaces described as per the table

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - an instance of the NDDS_Type_Plugin structure is created with the following member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Attribute	Parameters	Type
serialize_data	in: *struct CDR_Stream_t* *stream in: *const void* *sample in: *void* *param	*RTI_BOOL*
deserialize_data	in: *struct CDR_Stream_t* *stream out: *void* *sample in: *void* *param	*RTI_BOOL*
serialize_key	in: *struct CDR_Stream_t* *stream in: *const void* *sample in: *void* *param	*RTI_BOOL*
deserialize_key	in: *struct CDR_Stream_t* *stream out: *void* *sample in: *void* *param	*RTI_BOOL*
get_serialized_sample_max_size	in: *struct NDDS_Type_Plugin* *plugin in: *RTI_UINT32* current_alignment in: *void* *param	*RTI_UINT32*
get_serialized_key_max_size	in: *struct NDDS_Type_Plugin* *plugin in: *RTI_UINT32* current_alignment in: *void* *param	*RTI_UINT32*
create_sample	in: *struct NDDS_Type_Plugin* *plugin out: *void* **sample in: *void* *param	*RTI_BOOL*
delete_sample	in: *struct NDDS_Type_Plugin* *plugin in: *void* *sample in: *void* *param	*RTI_BOOL*
copy_sample	in: *struct NDDS_Type_Plugin* *plugin out: *void* *dst in: *void* *src	*RTI_BOOL*

	in: *void* *param	
get_key_kind	in: *struct NDDS_Type_Plugin* *plugin in: *void* *param	*NDDS_TypePluginKeyKind*
instance_to_keyhash	in: *struct NDDS_Type_Plugin* *plugin in: *struct CDR_Stream_t* *stream out: *DDS_KeyHash_t* *key_hash in: *const void* *instance in: *void* *param	*RTI_BOOL*
create_typed_datareader	in: *void* *reader	*void**
create_typed_datawriter	in: *void* *writer	*void**
delete_typed_datareader	in: *void* *wrapper	*void*
delete_typed_datawriter	in: *void* *wrapper	*void*

11.1.1 serialize_data

QTS-R-700.1

Analysis

The requirement is met when:

- [**TC.1**] serialize_data operation serializes a sample into a Stream_t instance.

TC.1

- **Preconditions:**
  - A created sample.
- **Action:**
  - Call the serialize_data operation on a created sample.
- **Check that:**
  - The given sample is serialized into a Stream_t instance by using serialization rules defined by the CDR specification.

11.1.2 deserialize_data

QTS-R-700.2

Analysis

The requirement is met when:

- [**TC.1**] deserialize_data operation deserializes a sample from a Stream_t instance.

TC.1

- **Preconditions:**
  - A serialized sample to a Stream_t instance.
  - The instance contains data serialized by using rules defined in the CDR specification.
- **Action:**
  - Call the deserialize_data operation on a sample from a Stream_t instance.
- **Check that:**
  - A sample is deserialized from a Stream_t instance.

11.1.3 get_serialized_sample_max_size

QTS-R-700.3

Analysis
The requirement is met when:

- [**TC.1**] get_serialized_sample_max_size operation retrieves the maximum size (in bytes) of a sample when serialized into network representation, using serialization rules defined by the CDR specification.

TC.1

- **Preconditions:**
  - N/A
- **Action:**
  - Call the get_serialized_sample_max_size operation.
- **Check that:**
  - The maximum size (in bytes) of a serialized sample is returned.

11.1.4 create_sample

QTS-R-700.4

Analysis

The requirement is met when:

- [**TC.1**] create_sample operation retrieves a sample.

TC.1

- **Preconditions**:
  - N/A
- **Action:**
  - Call the create_sample operation.
- **Check that:**
  - A sample is retrieved.

11.1.5 copy_sample

QTS-R-700.5

Analysis

The requirement is met when:

- [**TC.1**] copy_sample operation performs a deep copy of all attributes from one sample to another.

TC.1

- **Preconditions:**
  - A created sample.
- **Action:**
  - Call the copy_sample operation on a created sample.
- **Check that:**
  - A deep copy of all attributes from one sample to another is performed.

11.1.6 serialize_key

QTS-R-700.6

Analysis

The requirement is met when:

- [**TC.1**] serialize_key operation serializes a key provided as sample into a Stream_t instance.

TC.1

- **Preconditions:**
  - A key provided as sample.
- **Action:**
  - Call the serialize_key operation on a created key provided as sample.
- **Check that:**
  - The given key provided as sample is serialized into a Stream_t instance by using serialization rules defined by the CDR specification.

11.1.7 deserialize_key

QTS-R-700.7

Analysis

The requirement is met when:

- [**TC.1**] deserialize_key operation deserializes a key from a Stream_t instance.

TC.1

- **Preconditions:**
  - A serialized key provided as a sample to a Stream_t instance.
  - The instance contains data serialized by using rules defined in the CDR specification.
- **Action:**
  - Call the deserialize_key operation on a serialized key provided as a sample from a Stream_t instance.
- **Check that:**
  - A key provided as a sample is deserialized from a Stream_t instance.

11.1.8 get_serialized_key_max_size

QTS-R-700.8

Analysis

The requirement is met when:

- [**TC.1**] get_serialized_key_max_size operation retrieves the maximum size (in bytes) of a key when serialized into network representation, using serialization rules defined by the CDR specification.

TC.1

- **Preconditions:**
  - N/A
- **Action:**
  - Call the get_serialized_key_max_size operation.
- **Check that:**
  - The maximum size (in bytes) of a serialized key provided as a sample is returned.

12 Zero Copy Transfer Test Specifications

12.1 Zero Copy PSL OSAPI

12.1.1 SHMEM Segment

QTS-R-601.1

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_exists* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_exists* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_exists	in: *const char* *name	*RTI_BOOL*

QTS-R-601.1.1

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_SharedMemorySegment_exists* returns *RTI_TRUE* when the *OSAPI_SharedMemorySegment* referenced by the *name* parameter exists.

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* referenced by the *name* parameter exists
- **Actions:**
  - call *OSAPI_SharedMemorySegment_exists* with *name*
- **Check that:**
  - *OSAPI_SharedMemorySegment_exists* returns *RTI_TRUE*

QTS-R-601.1.2

Analysis

The requirement is met when the operation *OSAPI_SharedMemorySegment_exists* returns *RTI_FALSE* when:

- [**TC.1**] the *OSAPI_SharedMemorySegment* referenced by the *name* parameter does not exist
- [**TC.2**] the existence of the *OSAPI_SharedMemorySegment* referenced by the *name* parameter cannot be determined

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* referenced by the *name* parameter does not exist
- **Actions:**
  - call *OSAPI_SharedMemorySegment_exists* with *name*
- **Check that:**
  - *OSAPI_SharedMemorySegment_exists* returns *RTI_FALSE*

TC.2

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* referenced by the *name* parameter exists
  - the existence of the *OSAPI_SharedMemorySegment* referenced by the *name* parameter cannot be determined
- **Actions:**
  - call *OSAPI_SharedMemorySegment_exists* with *name*
- **Check that:**
  - *OSAPI_SharedMemorySegment_exists* returns *RTI_FALSE*

QTS-R-601.2

Analysis
The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_get_max_size* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_get_max_size* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_get_max_size	in: *void*	RTI_UINT64

QTS-R-601.2.1

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_get_max_size* operation returns the maximum number of bytes an *OSAPI_SharedMemorySegment* can hold.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call *OSAPI_SharedMemorySegment_get_max_size*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_get_max_size* returns the maximum number of bytes an *OSAPI_SharedMemorySegment* can hold

QTS-R-601.3

Analysis
The requirement is met when:

- [**TC.1**] *Operation OSAPI_SharedMemorySegment_is_owner_alive* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_is_owner_alive* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_is_owner_alive	in: *struct OSAPI_SharedMemorySegmentHandle* *handle out: *RTI_BOOL* *alive	*RTI_BOOL*

QTS-R-601.3.1

Analysis
The requirement is met when the *OSAPI_SharedMemorySegment_is_owner_alive* operation, upon success, returns *RTI_TRUE* and :

- [**TC.1**] sets *alive* to *RTI_TRUE* when the owner of the *OSAPI_SharedMemorySegment* referenced by its *handle* parameter is alive
- [**TC.2**] sets *alive* to *RTI_FALSE* when the owner of the *OSAPI_SharedMemorySegment* referenced by its *handle* parameter is not alive

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_is_owner_alive* succeeds when called
  - the owner of the *OSAPI_SharedMemorySegment* referenced by its *handle* parameter is alive
- **Actions:**
  - call *OSAPI_SharedMemorySegment_is_owner_alive* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_is_owner_alive* returns *RTI_TRUE*
  - parameter *alive* is set to *RTI_TRUE*

TC.2

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_is_owner_alive* succeeds when called
  - the owner of the *OSAPI_SharedMemorySegment* referenced by its *handle* parameter is not alive
- **Actions:**
  - call *OSAPI_SharedMemorySegment_is_owner_alive* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_is_owner_alive* returns *RTI_TRUE*
  - parameter *alive* is set to *RTI_FALSE*

QTS-R-601.3.2

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_is_owner_alive* operation returns *RTI_FALSE* when the liveliness of the *OSAPI_SharedMemorySegment* owner referenced by the *handle* parameter cannot be determined.

TC.1

- **Preconditions:**
  - the liveliness of the *OSAPI_SharedMemorySegment* owner referenced by the *handle* parameter cannot be determined when the operation *OSAPI_SharedMemorySegment_is_owner_alive* is called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_is_owner_alive* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_is_owner_alive* returns *RTI_FALSE*

QTS-R-601.4

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegmentHandle_get_size* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegmentHandle_get_size* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegmentHandle_get_size	in: *OSAPI_SHMEM_MODE* mode	*RTI_SIZE_T*

QTS-R-601.4.1

Analysis

The requirement is met when the *OSAPI_SharedMemorySegmentHandle_get_size* operation returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* of the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter when:

- [**TC.1**] the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter is not divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- [**TC.2**] the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter is divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*

TC.1

- **Preconditions:**
  - the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter is not divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- **Actions:**
  - call *OSAPI_SharedMemorySegmentHandle_get_size* with *mode*
- **Check that:**
  - the *OSAPI_SharedMemorySegmentHandle_get_size* returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is higher than the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter.

TC.2

- **Preconditions:**
    - the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter is divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- **Actions:**
    - call *OSAPI_SharedMemorySegmentHandle_get_size* with *mode*
- **Check that:**
    - the *OSAPI_SharedMemorySegmentHandle_get_size* returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is equal to the number of bytes of an *OSAPI_SharedMemorySegmentHandle* for the mode specified by the *mode* parameter.

QTS-R-601.5

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegmentHeader_get_size* has the specified signature.

TC.1

- **Preconditions:**
    - N/A
- **Actions:**
    - N/A
- **Check that:**
    - *OSAPI_SharedMemorySegmentHeader_get_size* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegmentHeader_get_size	in: *OSAPI_SHMEM_MODE* mode	*RTI_SIZE_T*

QTS-R-601.5.1

Analysis

The requirement is met when the *OSAPI_SharedMemorySegmentHeader_get_size* operation returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* of the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter, when:

- [**TC.1**] the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter is not divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- [**TC.2**] the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter is divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*

TC.1

- **Preconditions:**
  - the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter is not divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- **Actions:**
  - call *OSAPI_SharedMemorySegmentHeader_get_size* with *mode*
- **Check that:**
  - the *OSAPI_SharedMemorySegmentHeader_get_size* returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is higher than the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter.

TC.2

- **Preconditions:**
  - the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter is divisible by *OSAPI_SHARED_MEMORY_ALIGNMENT*
- **Actions:**
  - call *OSAPI_SharedMemorySegmentHeader_get_size* with *mode*
- **Check that:**
  - the *OSAPI_SharedMemorySegmentHeader_get_size* returns the minimum multiple of *OSAPI_SHARED_MEMORY_ALIGNMENT* that is equal to the number of bytes of an *OSAPI_SharedMemorySegmentHeader* for the mode specified by the *mode* parameter.

QTS-R-601.6

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_create_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_create_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_create_impl	inout: *struct OSAPI_SharedMemorySegmentHandle* *handle in: *const char* *name in: *RTI_UINT64* size in: *OSAPI_SHMEM_MODE* mode out: *RTI_BOOL* *exists	*RTI_BOOL*

QTS-R-601.6.1

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_create_impl* creates a new initialized *OSAPI_SharedMemorySegment*, sets parameters and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_create_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_create_impl* with *handle, name, size, mode* and *exists*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_create_impl*:
    - creates a new initialized *OSAPI_SharedMemorySegment* that is:
      - accessible to other processes,
      - named according to the name specified by the *name* parameter,
      - of a size that is equal to or greater than the size specified by the *size* parameter,
      - with a mode specified by the *mode* parameter
    - populates the *handle* with the *OSAPI_SharedMemorySegment* information
    - sets the *exists* parameter to *RTI_FALSE*
    - returns *RTI_TRUE*

QTS-R-601.6.2

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_create_impl* operation sets the *exists* parameter to *RTI_TRUE* and returns *RTI_FALSE*, when there is already an *OSAPI_SharedMemorySegment* in the system identified by the name specified by the *name* parameter

TC.1

- **Preconditions:**
    - there is already an *OSAPI_SharedMemorySegment* in the system identified by the name specified by the *name* parameter
- **Actions:**
    - call *OSAPI_SharedMemorySegment_create_impl* with *name* and *exists*
- **Check that:**
    - the *OSAPI_SharedMemorySegment_create_impl*:
        - sets the *exists* parameter to *RTI_TRUE*
        - returns *RTI_FALSE*

QTS-R-601.6.3

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_create_impl* operation returns *RTI_FALSE* when it cannot create the new *OSAPI_SharedMemorySegment*

TC.1

- **Preconditions:**
    - *OSAPI_SharedMemorySegment_create_impl* operation cannot create the new *OSAPI_SharedMemorySegment* when called
- **Actions:**
    - call *OSAPI_SharedMemorySegment_create_impl*
- **Check that:**
    - the *OSAPI_SharedMemorySegment_create_impl* returns *RTI_FALSE*

QTS-R-601.7

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_delete_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_delete_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_delete_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

QTS-R-601.7.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_delete_impl* deletes the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter, and returns *RTI_TRUE*.

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_delete_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_delete_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_delete_impl*:
    - deletes the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter
    - returns *RTI_TRUE*

QTS-R-601.7.2

Analysis

The requirement is met when:

- **[TC.1]** the *OSAPI_SharedMemorySegment_delete_impl* returns *RTI_FALSE* when it cannot delete the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment_delete_impl* operation cannot delete the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_delete_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_delete_impl* returns *RTI_FALSE*

QTS-R-601.8

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_SharedMemorySegment_attach_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_attach_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_attach_impl	inout: *struct OSAPI_SharedMemorySegmentHandle* *handle in: *const char* *name, in: *OSAPI_SHMEM_MODE* mode	*RTI_BOOL*

QTS-R-601.8.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_attach_impl* attaches the *OSAPI_SharedMemorySegmentHandle* specified by the *handle* parameter, in the mode specified by the *mode* parameter, to an existing *OSAPI_SharedMemorySegment* with a name equal to the *name* parameter, and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_attach_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_attach_impl* with *handle, mode* and *name*
- **Check that:**
  - the OSAPI_SharedMemorySegment_attach_impl :
    - attaches the *OSAPI_SharedMemorySegmentHandle* specified by the *handle* parameter, in the mode specified by the *mode* parameter, to an existing *OSAPI_SharedMemorySegment* with a name equal to the *name* parameter
    - returns *RTI_TRUE*

QTS-R-601.8.2

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_attach_impl* returns *RTI_FALSE* when it *OSAPI_SharedMemorySegment_attach_impl* operation cannot attach, in the mode specified by the *mode* parameter, to an OSAPI_SharedMemorySegment with a name equal to the *name* parameter

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment_attach_impl* operation cannot attach, in the mode specified by the *mode* parameter, to an OSAPI_SharedMemorySegment with a name equal to the *name* parameter when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_attach_impl* with *handle, mode* and *name*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_attach_impl* returns *RTI_FALSE*

QTS-R-601.9

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_detach_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_detach_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_detach_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

QTS-R-601.9.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_detach_impl* detaches the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter, and returns *RTI_TRUE*.

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_detach_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_detach_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_detach_impl*:
    - detaches the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter
    - returns *RTI_TRUE*

QTS-R-601.9.2

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_detach_impl* returns *RTI_FALSE* when it cannot detach the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment_detach_impl* operation cannot detach the *OSAPI_SharedMemorySegment* referenced by the handle specified by the *handle* parameter when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_detach_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_detach_impl* returns *RTI_FALSE*

QTS-R-601.10

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_lock_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_lock_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_lock_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle out: *OSAPI_SHMEM_STATUS* *status_out	*RTI_BOOL*

QTS-R-601.10.1

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_lock_impl* operation locks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter when the *OSAPI_SharedMemorySegment* is not locked yet.

TC.1

- **Preconditions:**
    - the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter is not locked yet
- **Actions:**
    - call *OSAPI_SharedMemorySegment_lock_impl* with *handle*
- **Check that:**
    - the *OSAPI_SharedMemorySegment_lock_impl* locks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter

QTS-R-601.10.2

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_lock_impl* operation blocks the calling execution context until the lock is released and then locks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter when the *OSAPI_SharedMemorySegment* is already locked

TC.1

- **Preconditions:**
    - the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter is already locked
- **Actions:**
    - call *OSAPI_SharedMemorySegment_lock_impl* with *handle*
- **Check that:**
    - the *OSAPI_SharedMemorySegment_lock_impl* blocks the calling execution context until the lock is released and then locks the *OSAPI_SharedMemorySegment*

QTS-R-601.10.3

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_lock_impl* operation returns OSAPI *SHMEM_STATUS_OK* through *status_out* parameter and returns *RTI_TRUE* when it succeeds

TC.1

- **Preconditions:**
    - *OSAPI_SharedMemorySegment_lock_impl* succeeds when called

  Actions:
    - call *OSAPI_SharedMemorySegment_lock_impl* with *status_out*
- **Check that:**
    - the *OSAPI_SharedMemorySegment_lock_impl*:
        - returns OSAPI_SHMEM_STATUS_OK through *status_out* parameter
        - returns *RTI_TRUE*

QTS-R-601.10.4

Analysis
The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_lock_impl* operation returns OSAPI *OSAPI_SHMEM_STATUS_OWNER_DEAD* through *status_out* parameter when the previous owner of the *OSAPI_SharedMemorySegment* terminated while holding a lock on the *OSAPI_SharedMemorySegment*

TC.1

- **Preconditions:**
    - the owner of the *OSAPI_SharedMemorySegment* terminated while holding a lock on the *OSAPI_SharedMemorySegment*

  Actions:
    - call *OSAPI_SharedMemorySegment_lock_impl* with *status_out* on a *OSAPI_SharedMemorySegment* that was held by the owner that was terminated
- **Check that:**
    - the *OSAPI_SharedMemorySegment_lock_impl*:
        - returns *OSAPI_SHMEM_STATUS_OWNER_DEAD* through *status_out* parameter

QTS-R-601.10.5

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_lock_impl* operation returns *RTI_FALSE* when the *OSAPI_SharedMemorySegment* cannot be locked

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* cannot be locked
- **Actions:**
  - call *OSAPI_SharedMemorySegment_lock_impl*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_lock_impl* returns *RTI_FALSE*

QTS-R-601.11

Analysis

The requirement is met when:

- [**TC.1**] Operation *OSAPI_SharedMemorySegment_unlock_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_unlock_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_unlock_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

QTS-R-601.11.1

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_unlock_impl* unlocks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_unlock_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_unlock_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_unlock_impl*:
    - unlocks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter
    - returns *RTI_TRUE*

QTS-R-601.11.2

Analysis

The requirement is met when:

- [**TC.1**] the *OSAPI_SharedMemorySegment_unlock_impl* operation returns *RTI_FALSE* when the *OSAPI_SharedMemorySegment* cannot be unlocked

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* cannot be unlocked

  Actions:
  - call *OSAPI_SharedMemorySegment_unlock_impl*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_unlock_impl* returns *RTI_FALSE*

QTS-R-601.12

Analysis

The requirement is met when:

- **[TC.1]** Operation *OSAPI_SharedMemorySegment_mark_consistent_impl* has the specified signature.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - *OSAPI_SharedMemorySegment_mark_consistent_impl* has the following signature:

Operations	Parameters	Type
OSAPI_SharedMemorySegment_mark_consistent_impl	in: *struct OSAPI_SharedMemorySegmentHandle* *handle	*RTI_BOOL*

QTS-R-601.12.1

Analysis

The requirement is met when:

- **[TC.1]** the *OSAPI_SharedMemorySegment_mark_consistent_impl* marks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter as consistent and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - *OSAPI_SharedMemorySegment_mark_consistent_impl* succeeds when called
- **Actions:**
  - call *OSAPI_SharedMemorySegment_mark_consistent_impl* with *handle*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_mark_consistent_impl*:
    - marks the *OSAPI_SharedMemorySegment* referenced by its handle specified by the *handle* parameter as consistent
    - returns *RTI_TRUE*

QTS-R-601.12.2

Analysis

The requirement is met when the *OSAPI_SharedMemorySegment_mark_consistent_impl* operation returns *RTI_FALSE* when:

- [**TC.1**] the *OSAPI_SharedMemorySegment* cannot be marked as consistent
- [**TC.2**] the support for robust locking is not enabled

TC.1

- **Preconditions:**
  - the *OSAPI_SharedMemorySegment* cannot be marked as consistent
  - the support for robust locking is enabled
- **Actions:**
  - call *OSAPI_SharedMemorySegment_mark_consistent_impl*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_mark_consistent_impl* returns *RTI_FALSE*

TC.2

- **Preconditions:**
  - the support for robust locking is not enabled

  Actions:
  - call *OSAPI_SharedMemorySegment_mark_consistent_impl*
- **Check that:**
  - the *OSAPI_SharedMemorySegment_mark_consistent_impl* returns *RTI_FALSE*

12.1.2 Type Plugin

QTS-R-604.0

Analysis

The requirement is met when:

- [**TC.1**] an instance of the *NDDS_Type_Plugin* structure for Zero Copy Transport is created with the specified additional member attributes assigned function pointers to functions that implement the interfaces described as per the table

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - an instance of the *NDDS_Type_Plugin* structure for Zero Copy Transport is created with the following additional member attributes assigned function pointers to functions that implement the interfaces described in the table below:

Attribute	Parameters	Type
get_loan	in: *void* *managed_pool out: *void*** sample_out	*ReturnCode_t*
discard_loan	in: *void* *managed_pool in: *RTI_UINT32* sample_id	*RTI_BOOL*
create_managed_pool	in: *struct NDDS_Type_Plugin* *plugin in: *void* *datawriter in: *void* *param out: *void* **managed_pool_out	*RTI_BOOL*
delete_managed_pool	in: *void* *managed_pool	*void*
get_sample_id	in: *void* *managed_pool in: *const void* *user_data out: *RTI_UINT32* *sample_id_out	*RTI_BOOL*
needs_opaque_samples	in: *void* *reader in: *void* *param	*RTI_BOOL*

QTS-R-604.1

Analysis

The requirement is met when:

- **[TC.1]** The *get_loan* operation requests a loan of a sample from a managed pool of samples.

TC.1

- **Preconditions:**
  - N/A
- **Action:**
  - Call the *get_loan* operation.
- **Check that:**
  - The *get_loan* operation requests a loan of a sample from a managed pool of sample.

QTS-R-604.2

Analysis

The requirement is met when:

- **[TC.1]** The *discard_loan* operation returns a loaned sample previously retrieved by *get_loan* operation to a managed pool of samples.

TC.1

- **Preconditions:**
  - A sample that was retrieved by the *get_loan* operation.
- **Action:**
  - Call the *discard_loan* operation with the sample ID of the loaned sample.
- **Check that:**
  - The *discard_loan* operation returns the loaned sample to the managed pool of samples.

QTS-R-604.3

Analysis

The requirement is met when:

- [**TC.1**] The *create_managed_pool* operation creates a managed pool of samples if it is needed.

TC.1

- **Preconditions:**
  - The creation of a managed pool of samples is needed.
- **Action:**
  - Call the *create_managed_pool* operation.
- **Check that:**
  - The *create_managed_pool* operation creates a managed pool of samples.

QTS-R-604.4

Analysis

The requirement is met when:

- [**TC.1**] The *get_sample_id* operation looks up the unique ID of a loaned sample given its user data memory region.

TC.1

- **Preconditions:**
  - A loaned sample.
- **Action:**
  - Call the *get_sample_id* operation with the address of user data of the loaned sample.
- **Check that:**
  - The *get_sample_id* operation looks up the unique ID of a loaned sample.

QTS-R-604.5

Analysis

The requirement is met when:

- [**TC.1**] The needs_opaque_samples operation checks whether the additional opaque samples are needed.

TC.1

- **Preconditions:**
  - N/A
- **Action:**
  - Call the needs_opaque_samples operation.
- **Check that:**
  - The needs_opaque_samples operation checks whether the additional opaque samples are needed.

12.1.3 Concurrency

QTS-R-605.1

Analysis

The requirement is met when:

- [**TC.1**] the listed PSL operations are reentrant and thread-safe

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - The following PSL operations are reentrant and thread-safe:
    - *OSAPI_SharedMemorySegment_exists*
    - *OSAPI_SharedMemorySegment_get_max_size*
    - *OSAPI_SharedMemorySegment_is_owner_alive*
    - *OSAPI_SharedMemorySegmentHandle_get_size*
    - *OSAPI_SharedMemorySegmentHeader_get_size*
    - *OSAPI_SharedMemorySegment_create_impl*
    - *OSAPI_SharedMemorySegment_delete_impl*
    - *OSAPI_SharedMemorySegment_attach_impl*
    - *OSAPI_SharedMemorySegment_detach_impl*
    - *OSAPI_SharedMemorySegment_lock_impl*
    - *OSAPI_SharedMemorySegment_unlock_impl*
    - *OSAPI_SharedMemorySegment_mark_consistent_impl*

12.2 Zero Copy Notification Transport

12.2.1 Notification transport plugin API

QTS-R-602.1

Analysis

The requirement is met when:

- [**TC.1**] an instance of the ZCOPY_NotifUserInterfaceI structure is created and its member attributes have assigned function pointers that implement interfaces as per the table

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - an instance of the ZCOPY_NotifUserInterfaceI structure is created and its member attributes have been assigned function pointers to functions that implement the interfaces as per the table below:

Attribute	Parameters	Type
create_instance	in: *NETIO_Interface_T* *upstream in: *void* *property	*NETIO_Interface_T**
reserve_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr out: *void*** port_entry_out	*RTI_BOOL*
release_address	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *void** port_entry	*RTI_BOOL*
resolve_address	in: *NETIO_Interface_T* *user_intf in: *const* char *address_string out: *struct NETIO_Address* *address_value out: *RTI_BOOL* *invalid	*RTI_BOOL*
send	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination in: *void** route_entry	*RTI_BOOL*
get_route_table	in: *NETIO_Interface_T* *netio_intf inout: *struct NETIO_Address*Seq *address inout: *struct* NETIO_NetmaskSeq *netmask	*RTI_BOOL*

Attribute	Parameters	Type
bind	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *struct NETIO_Address* *dst_addr in: *void* *port_entry out: *void* **bind_entry_out	*RTI_BOOL*
unbind	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *src_addr in: *struct NETIO_Address* *dst_addr in: *void* *port_entry in: *void* *bind_entry	*RTI_BOOL*
add_route	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination out: *void*** route_entry_out	*RTI_BOOL*
delete_route	in: *NETIO_Interface_T* *user_intf in: *struct NETIO_Address* *source in: *struct NETIO_Address* *destination in: *void** route_entry	*RTI_BOOL*
notify_recv_port	in: *NETIO_Interface_T* *user_intf in: *void** port_entry	*RTI_BOOL*

12.2.1.1 create_instance

QTS-R-602.1.1

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.create_instance* operation returns a non-NULL pointer on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.create_instance* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.create_instance* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.create_instance* operation returns a non-NULL pointer to an instance of an interface of type *NETIO_Interface_T*

QTS-R-602.1.2

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.create_instance* operation returns NULL on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.create_instance* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.create_instance* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.create_instance* operation returns NULL

QTS-R-602.1.3

Analysis

The requirement is met when:

- [**TC.1**] the PSL does not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *ZCOPY_NotifUserInterfaceI.create_instance.*

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - the PSL does not use *upstream NETIO_Interface_T* parameter to call RCC APIs in the implementation of *ZCOPY_NotifUserInterfaceI.create_instance.*

12.2.1.2 reserve_address

QTS-R-602.1.11

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.reserve_address* operation causes the interface specified by *user_intf* to configure resources for listening to messages on address *src_add* and returns a pointer to a port entry specified by *port_entry_out* and returns *RTI_TRUE* on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.reserve_address* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.reserve_address* operation with *user_intf* and *src_addr*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.reserve_address* operation:
    - causes the interface specified by *user_intf* to configure resources for listening to messages on address *src_add*
    - returns a pointer to a port entry specified by *port_entry_out*
    - returns *RTI_TRUE*

QTS-R-602.1.12

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.reserve_address* operation returns NULL through the port entry specified by *port_entry_out* and returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.reserve_address* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.reserve_address* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.reserve_address* operation returns:
    - NULL through the port entry specified by *port_entry_out*
    - *RTI_FALSE*

12.2.1.3 release_address

QTS-R-602.1.21

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.release_address* operation causes the interface specified by *user_intf* to release resources associated with the port entry specified by *port_entry* used for listening to messages on address *src_addr* and returns *RTI_TRUE* on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.release_address* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.release_address* operation with *user_intf*, *port_entry* and *src_addr*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.release_address* operation:
    - causes the interface specified by *user_intf* to release resources associated with the port entry specified by *port_entry* used for listening to messages on address *src_addr*
    - returns *RTI_TRUE*

QTS-R-602.1.22

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.release_address* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.release_address* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.release_address* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.release_address* operation returns *RTI_FALSE*

12.2.1.4 resolve_address

QTS-R-602.1.32

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.resolve_address* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.resolve_address* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.resolve_address* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.resolve_address* operation returns *RTI_FALSE*

QTS-R-602.1.31

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.resolve_address* operation sets *address_value* to a valid address for the Notification Mechanism interface specified by *user_intf,* sets the value pointed to by *invalid* to *RTI_FALSE*, and returns *RTI_TRUE* on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.resolve_address* operation converts address successfully when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.resolve_address* operation with *user_intf, address_value* and *invalid*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.resolve_address* operation:
    - sets *address_value* to a valid address for the Notification Mechanism interface specified by *user_intf*
    - sets the value pointed by *invalid* to *RTI_FALSE*
    - returns *RTI_TRUE*

12.2.1.5 send

QTS-R-602.1.41

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.send* operation causes the interface specified by *user_intf* to send a notification from the source specified by *source* to the destination specified by *destination* using the route entry specified by *route_entry* and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.send* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.send* operation with *user_intf, source, destination* and *route_entry*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.send* operation:
    - cause the interface specified by *user_intf* to send a notification from the source specified by *source* to the destination specified by *destination* using the route entry specified by *route_entry*
    - return *RTI_TRUE*

QTS-R-602.1.42

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.send* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.send* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.send* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.send* operation returns *RTI_FALSE*

12.2.1.6 get_route_table

QTS-R-602.1.51

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.get_route_table* operation retrieves a sequence of all addresses as *address* and a sequence of all netmasks as *netmask* from the routing table to which the Notification Mechanism interface specified by *netio_intf* can send to, and returns *RTI_TRUE* on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.get_route_table* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.get_route_table* operation with *address, netmask* and *netio_intf*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.get_route_table* operation:
    - retrieves a sequence of all addresses as *address* and a sequence of all netmasks as *netmask* from the routing table to which the Notification Mechanism interface specified by *netio_intf* can send to
    - returns *RTI_TRUE*

QTS-R-602.1.52

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.get_route_table* operation does not modify the *address* and *netmask* sequences and returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.get_route_table* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.get_route_table* operation with *address* and *netmask*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.get_route_table* operation:
    - does not modify the *address* and *netmask* sequences
    - returns *RTI_FALSE*

12.2.1.7 bind

QTS-R-602.1.61

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.bind* operation causes the implementation to listen for notification messages and returns *RTI_TRUE* on success

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.bind* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.bind* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.bind* operation:
    - causes the implementation to listen for notification messages
    - returns *RTI_TRUE*

QTS-R-602.1.62

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.bind* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.bind* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.bind* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.bind* operation returns *RTI_FALSE*

12.2.1.8 unbind

QTS-R-602.1.71

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.unbind* operation returns *RTI_TRUE*.

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.unbind* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.unbind* operation returns *RTI_TRUE*

12.2.1.9 add_route

QTS-R-602.1.81

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.add_route* operation causes the interface specified by *user_intf* to add a route from the source specified by *source* to the destination specified by *destination*, returns a pointer to a route entry specified by *route_entry_out* and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.add_route* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.add_route* operation with *user_intf, source*, *destination* and *route_entry_out*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.add_route* operation:
    - causes the interface specified by *user_intf* to add a route from the source specified by *source* to the destination specified by *destination*
    - returns a pointer to a route entry specified by *route_entry_out*
    - returns *RTI_TRUE*

QTS-R-602.1.82

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.add_route* operation returns NULL through the port entry specified by *route_entry_out* and returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.add_route* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.add_route* operation with *route_entry_out*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.add_route* operation:
    - returns NULL through the port entry specified by *route_entry_out*
    - returns *RTI_FALSE*

12.2.1.10 delete_route

QTS-R-602.1.91

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.delete_route* operation causes the interface specified by *user_intf* to remove the *route_entry* from the source specified by *source* and the destination specified by *destination* and return *RTI_TRUE*

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.delete_route* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.delete_route* operation with *user_intf*, *route_entry*, *source* and *destination*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.delete_route* operation:
    - causes the interface specified by *user_intf* to remove the *route_entry* from the source specified by *source* and the destination specified by *destination*
    - returns *RTI_TRUE*

QTS-R-602.1.92

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.delete_route* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.delete_route* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.delete_route* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.delete_route* operation returns *RTI_FALSE*

12.2.1.11 notif_recv_port

QTS-R-602.1.101

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation causes the interface specified by *user_intf* to notify the receive port specified by *port_entry* and returns *RTI_TRUE*

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation succeeds when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation with *user_intf* and *port_entry*
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation:
    - causes the interface specified by *user_intf* to notify the receive port specified by *port_entry*
    - returns *RTI_TRUE*

QTS-R-602.1.102

Analysis

The requirement is met when:

- [**TC.1**] the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation returns *RTI_FALSE* on failure

TC.1

- **Preconditions:**
  - the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation fails when called
- **Actions:**
  - call the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation
- **Check that:**
  - the *ZCOPY_NotifUserInterfaceI.notify_recv_port* operation returns *RTI_FALSE*

12.2.2 Concurrency

QTS-R-606.1

Analysis

The requirement is met when:

- [**TC.1**] the listed *ZCOPY_NotifUserInterfaceI* operations are reentrant and thread-safe

TC.1

- **Preconditions:**
  - N/A
- **Actions:**
  - N/A
- **Check that:**
  - The following *ZCOPY_NotifUserInterfaceI* operations are reentrant and thread-safe:
    - *ZCOPY_NotifUserInterfaceI.create_instance*
    - *ZCOPY_NotifUserInterfaceI.reserve_address*
    - *ZCOPY_NotifUserInterfaceI.release_address*
    - *ZCOPY_NotifUserInterfaceI.resolve_address*
    - *ZCOPY_NotifUserInterfaceI.send*
    - *ZCOPY_NotifUserInterfaceI.get_route_table*
    - *ZCOPY_NotifUserInterfaceI.bind*
    - *ZCOPY_NotifUserInterfaceI.unbind*
    - *ZCOPY_NotifUserInterfaceI.add_route*
    - *ZCOPY_NotifUserInterfaceI.delete_route*
    - *ZCOPY_NotifUserInterfaceI.notify_recv_port*

13 PSL Platform Notes and Build Instructions

This section describes the steps required to build the Platform Support Library (PSL) and integrate it with the Platform Independent Library (PIL). These steps include which options are used to build the PIL and how to configure the Platform, including:

- How the PIL is built
- How the PSL is built
- How the libraries are built
- Which compiler, version, and tools are used
- Lists of compilation and linking flags
- Calling conventions used

13.1 Instructions for Using RCC for ARMv8 Architecture

RTI Connext Cert 2.4.15 for ARMv8 is available as a static Platform Independent Library (PIL). This section describes how to build and link a Platform Support Library (PSL) with the static PIL.

13.1.1 The Platform Independent Library

The *RTI Connext Cert* Platform Independent Library (RCC PIL) is delivered as a static library and is built using the QNX qcc 8.3.0 compiler with the following options:

C compiler option	Description
`-Vgcc/8.3.0,gcc_ntoaarch64le`	Specifies the compiler name, version number, and the target name as gcc, 8.3.0, and gcc_ntoaarch64le, respectively.
`-std=c99`	Specifies the language standard to C99.
`-Winit-self`	Warns about uninitialized variables which are initialized with themselves.
`-fstrict-aliasing`	Allows the compiler to assume the strictest aliasing rules applicable to the language being compiled.
`-Wmissing-declarations`	Warns if a global function is defined without a previous declaration.
`-Wall`	Enables all the warnings about constructions that some users consider questionable, and that are easy to a*void* (or modify to prevent the warning), even in conjunction with macros.
`-Wextra`	Prints extra warning messages for some events.
`-Wpedantic`	Issues all the warnings demanded by strict ISO C and ISO C++.
`-Wshadow`	Warns when a local variable shadows another local variable, parameter, or global variable, or when a built-in function is shadowed.
`-Wcast-align`	Warns when a pointer is cast such that the required alignment of the target is increased.
`-Wunused`	Warns when a variable, function, parameter, label, or value is unused.
`-Wconversion`	Warns if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype.
`-Wsign-conversion`	Warns about implicit conversions that may change the sign of an integer value, like assigning a signed integer expression to an unsigned integer variable.
`-Wlogical-op`	Warns about suspicious uses of logical operators in expressions.
`-Wdouble-promotion`	Gives a warning when a value of type float is implicitly promoted to double.
`-DNDEBUG`	Excludes debug information.

13.1.2 The Platform Support Library

The *RTI Connext Cert* Platform Support Library (RCC PSL) must be built to be binary compatible with the RCC Platform Independent Library (RCC PIL). This means that the same compiler preprocessor options used to build the RCC PIL must be used to build the RCC PSL.

In addition, the following compiler preprocessor options must be used:

C compiler option	Description
-DRTI_CERT	Enable the RCC CERT profile. Failure to specify this option will result in undefined behavior.
-DRTI_PSL	Enable support to build an RCC PSL. All OSAPI implementations must be compiled with this option. NOTE: An integration with NETIO_DGRAM is considered to be a DDS application and should use the -DRTI_PIL option instead.
-DOSAPI_CC_DEF_H=<name file with compiler definitions>	If QCC is used, then replace this option with -DOSAPI_CC_DEF_H=osapi/osapi_cc_qcc.h This assumes that that the compiler include path includes <installation directory>/rti_me_cert.1.0/include/rti_me
-DRTI_LITTLE_ENDIAN	Enables support for a little-endian CPU.
-DNDEBUG	This option must be specified in order to be able to build an RCC PSL that is compatible with the RCC PIL.
-I<install directory>/rti_me_cert.1.0/include/rti_me	Location of header files required to build RCC applications and platform support libraries.

13.1.3 Compiling and Linking

13.1.3.1 Compiling the application

Use the following compiler definitions when compiling the code that links with the RCC PIL and RCC PSL:

C compiler option	Description
-DRTI_CERT	Enables the RCC CERT profile. Failure to specify this option will result in undefined behavior.
-DRTI_PIL	Enables support for compiling an application that uses the RCC. NOTE: An integration with NETIO_DGRAM is considered to be a DDS application and should use the -DRTI_PIL option.

C compiler option	Description
-DOSAPI_CC_DEF_H=<name file with compiler definitions>	If QCC is used, then replace this option with -DOSAPI_CC_DEF_H=osapi/osapi_cc_qcc.h. This is the same file that is used to build the RCC PIL. This assumes that the compiler include path includes <install directory>/rti_me_cert.1.0/include/rti_me.
-DRTI_LITTLE_ENDIAN	Enables support for a little endian CPU.
-DNDEBUG	This option must be specified in order for an application to link with the RCC PSL and RCC PIL.
-I<install directory>/rti_me_cert.1.0/include/rti_me	Location of header files required to build RCC applications and platform support libraries.

13.1.3.2 Linking a DDS application with an RCC PIL and RCC PSL

The following artifacts are assumed to be available for linking:

Artifact	Description
ddsapp	A DDS application library.
rcc_netio	An RCC NETIO_DGRAM integration. **Note:** It is recommended to not implement an RCC NETIO_DGRAM integration and RCC OSAPI integration in the same library, as that will require a two-pass linking procedure.
rcc_psl	An RCC OSAPI implementation.

When linking, the libraries must be listed in the following order:

```
ddsapp rcc_netio rti_mez rcc_psl
```

14 Document Change Log

Version	Date	Name	Description
2.1	12/18/2024	Brian Sweet	• Updated QTS-R-526.10.
2.0	12/9/2024	Brian Sweet	• Added Zero Copy Transfer requirements and Test Specifications. • Major formatting changes.
1.1	10/06/2023	Brian Sweet	• Updated PSL Platform Notes and Build Instructions.
1.0	09/08/2023	Brian Sweet	• First release version. • Updated requirements and test specs.