|  |  |
| --- | --- |
|  |  |

EPTF CLL NQueue, Function Description

Contents

[1 Introduction 2](#_Toc290380009)

[1.1 Revision history 2](#_Toc290380010)

[1.2 How to Read this Document 2](#_Toc290380011)

[1.3 References 2](#_Toc290380012)

[1.4 Scope 2](#_Toc290380013)

[1.5 Recommended way of reading 3](#_Toc290380014)

[1.6 Typographical conventions 3](#_Toc290380015)

[1.7 Abbreviations 3](#_Toc290380016)

[1.8 Terminology 3](#_Toc290380017)

[2 General Description 4](#_Toc290380018)

[2.1 NQueues for resource allocation 5](#_Toc290380019)

[2.2 NQueues for data sorting 6](#_Toc290380020)

[3 Functional Interface 6](#_Toc290380021)

[3.1 Naming Conventions 6](#_Toc290380022)

[3.2 Public Functions 7](#_Toc290380023)

[3.2.1 Initialization 7](#_Toc290380024)

[3.2.2 Creating an empty queue 7](#_Toc290380025)

[3.2.3 Deleting a queue 7](#_Toc290380026)

[3.2.4 Creating a chain 7](#_Toc290380027)

[3.2.5 Creating items 7](#_Toc290380028)

[3.2.6 Moving an item to head or tail of chain 8](#_Toc290380029)

[3.2.7 Moving an Item after or before another item 8](#_Toc290380030)

[3.2.8 Getting the chain of an item 8](#_Toc290380031)

[3.2.9 Getting the length of a queue 9](#_Toc290380032)

[3.2.10 Getting the length of a chain 9](#_Toc290380033)

[3.2.11 Getting the head item of a chain 9](#_Toc290380034)

[3.2.12 Getting the tail item of a chain 9](#_Toc290380035)

[3.2.13 Iterating through a chain 9](#_Toc290380036)

[3.2.14 Logging functions 9](#_Toc290380037)

[3.2.15 Dumping a queue to a PNG image 10](#_Toc290380038)

[3.3 Summary Table of all public functions for EPTF NQueue 10](#_Toc290380039)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2011-04-12 | PA1 | First draft version | EGBOTAT |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## How to Read this Document

This is the Function Description for the NQueue of the Ericsson Performance Test Framework (TitanSim), Core Load Library (CLL). TitanSim CLL is developed for the TTCN-3 [1] Toolset with TITAN [2]. For more information on the TitanSim CLL please consult the Product Revision Information [3].

## References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. Part 1: Core Language
2. 1/198 17-CRL 113 200 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 109 21-CNL 113 512-2 Uen   
   TitanSim CLL for TTCN-3 toolset with TITAN, Product Revision Information
4. 155 17-CNL 113 512 Uen   
   TitanSim CLL for TTCN-3 toolset with TITAN, Function Specification
5. 6/155 16-CNL 113 512 Uen  
   EPTF CLL Free Busy Queue, Function Description

1. [www.graphviz.org](http://www.graphviz.org)  
   open source graph visualization software

## Scope

This document is to specify the content and functionality of the NQueue feature of the TitanSim CLL.

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of TitanSim CLL [4]. They should get familiar with the list of acronyms and the glossary in Section 1.7 and 1.8, respectively.

## Typographical conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context.

## Abbreviations

API Application Programming Interface

CLL Core Load Library

EPTF Ericsson Load Test Framework, formerly TITAN Load Test Framework

FIFO First In First Out

LIFO Last In First Out

PNG Portable Network Graphics

TitanSim Ericsson Load Test Framework, formerly TITAN Load Test Framework

TTCN-3 Testing and Test Control Notation version 3 [1]

## Terminology

*TitanSim Core (Load) Library(CLL)* is that part of the TitanSim software that is totally project independent. (I.e., which is not protocol-, or application-dependent). The TitanSim CLL is to be supplied and supported by the TCC organization. Any TitanSim CLL development is to be funded centrally by Ericsson

*NQueue* is a data type consisting of an arbitrary number of chains and an arbitrary number of items. Items in the queue are identified by their index, which has a continuous range and is thus useful for creating associated data structures by indexing a list of data with the item index.

*Chain* is a group of items in the queue. Each item belongs to a chain and can be moved to another chain. A chain is represented as a doubly linked list and can be iterated forward (head to tail) or backward (tail to head). A chain can, for example, be thought of as the state of its items (e.g. free/busy).

# General Description

This document specifies the NQueue feature of the TitanSim CLL.

The EPTF NQueue feature makes it possible to

* Manage linked lists (“chains”) over ‘record of’ datastructures easily using index-arithmetics
  + Provide efficient iterators over the elements linked into the same list
* Book-keep the state of elements of a ‘record of’ type efficiently
  + Provide efficient methods for moving elements back and forth between an arbitrary number of chains with computational complexity of O(1) (that is, constant), regardless of the length of the queue
  + Reorder items within the same chain
  + Maintain up-to-date additional statistics about the number of elements in each chain, respectively
* Implement stacks
  + Stack-like data structures can be implemented with a chain by adding elements at the chain head (or tail) and also removing them from the head (or tail) of the chain.

The NQueue feature can be thought of as a general version of the Free Busy Queue [5] feature and can, in a similar way, be used for dynamic memory allocation for the TTCN-3 language in an efficient way. Furthermore, it can be used to group the items in a ‘record of’ data structure by their state (chain ID) and iterating through these items.

An example of NQueue and an associated user defined data list can be seen on Figure 1.

Queue

Chain 0

**...**

0

1

2

3

4

5

6

7

**...**

n

0

1

2

3

4

5

6

7

n

items

associated data

head

tail

Chain 1

head

tail

Figure 1 An example of NQueue and associated data list

## NQueues for resource allocation

The NQueue feature can be used as a replacement of the Free Busy Queue feature for resource allocation. This section describes one of the possible ways to do this, i.e. items are reused in a FIFO (round-robin) order. It is also possible to do a stack-like implementation with LIFO reuse order.

Typically, an NQueue has an associated data array (record of a user defined type), with the same length as the queue. Two chains in the queue have to be created: one representing the free state and the other the busy state of items in the queue. This also represents the allocation state of elements in the corresponding data array with the same index. Allocating or freeing a new element is usually a matter of moving a slot from one chain to the other.

It is up to the user of the NQueue to resize the data record-of in case the length of the queue increases (it never decreases). This can happen if the free chain has no more elements, so a new item has to be created in the queue.

Allocating a new element can be done by one of the following:

* If the length of the free chain is nonzero, the head of the free chain will be the index of the allocated element. In this case, the item at this index has to be moved to the tail of the chain representing busy elements. In this case, an element of the associated data which was earlier freed is now reused.
* If the length of the free chain is zero, a new item has to be created in the queue. This can be done in the busy chain as a single step, so the item does not have to be moved from free to busy chain. The index of the new item is in this case larger than the valid index range of the associated data list, so a new element will be created there as well.

Freeing an element is done by simply moving the queue item with the same index to the tail of the free chain. This item will then be reused by one of the subsequent element allocations.

Figure 1 can be used as an example, with chain 0 representing free items and chain 1 representing busy/allocated items. In this case, the next allocated item will be item 0. Freeing one of the busy items will result in it being chained after item 5, and becoming the new tail of chain 0.

## NQueues for data sorting

NQueues can be also used for storing the ordering of user elements of user data arrays. If the order of the user data elements are stored as a linked list, reordering of complex data structures can be performed via integer index arithmetic only, that is, without moving the (possibly complex) user data itself.

# Functional Interface

Apart from this description a cross-linked reference guide for the TitanSim CLL Functions can be reached for on-line reading **Error! Reference source not found.**.

The component that wishes to use the NQueue feature has to extend the component type EPTF\_NQueue\_CT defined in module EPTF\_CLL\_NQueue\_Definitions.

The API of the NQueue feature is implemented in external functions. Because of this, the functions declared in module EPTF\_CLL\_NQueue\_Functions do not have a ‘runs on’ clause. However, all these functions have to be handled as if they were defined as ‘runs on EPTF\_NQueue\_CT’. Calling these functions from other functions that do not have a ‘runs on’ attribute or that run on a component type not extending EPTF\_NQueue\_CT is not recommended.

## Naming Conventions

All functions have the prefix f\_EPTF\_NQueue\_.

## Public Functions

### Initialization

Before using the EPTF NQueue feature, the following function has to be called:

*f\_EPTF\_NQueue\_init\_CT(pl\_selfName)*

### Creating an empty queue

The following function creates a new, empty queue:

*f\_EPTF\_NQueue\_createQueue(pl\_name)*

This function returns a queue ID of type EPTF\_NQueue\_QueueId which can later be used to reference the queue. An (optional) charstring name can be specified for the queue, which will be logged in error messages.

### Deleting a queue

The function *f\_EPTF\_NQueue\_deleteQueue(pl\_queue)* can be used to delete a queue and all the resources it uses. This is useful if a queue is used locally in a function. The IDs of deleted queues are stored in a stack and are reused on subsequent queue creations.

All queues are automatically deleted at component cleanup.

### Creating a chain

The following function creates a new chain within an existing queue:

*f\_EPTF\_NQueue\_createChain(pl\_queue)*

This function returns the ID of chain that is of type EPTF\_NQueue\_ChainId which can later be used to reference the chain within the queue.

### Creating items

The following functions can be used to create a single item in a queue at either the head or tail of a chain, of after/before another item:

*f\_EPTF\_NQueue\_createItemAtChainHead(pl\_queue, pl\_chain)*

*f\_EPTF\_NQueue\_createItemAtChainTail(pl\_queue, pl\_chain)*

*f\_EPTF\_NQueue\_createItemAfter(pl\_queue, pl\_afterIdx)*

*f\_EPTF\_NQueue\_createItemBefore(pl\_queue, pl\_beforeIdx)*

These functions return the index of the created item (of type EPTF\_NQueue\_ItemIdx).

The following functions can be used to create more than one items at the same time:

*f\_EPTF\_NQueue\_createItem****s****AtChainHead(pl\_queue, pl\_chain, pl\_count)*

*f\_EPTF\_NQueue\_createItem****s****AtChainTail(pl\_queue, pl\_chain, pl\_count)*

These functions do not have a return value. The created items can be retrieved from the head or tail of the respective chain. These functions are identical to calling their single item creating counterpart in a for loop pl\_count times (i.e. createItemsAtChainHead chains the new items in a reverse order).

### Moving an item to head or tail of chain

The following functions can be used to move an item to the head or tail of the same or another chain:

*f\_EPTF\_NQueue\_moveToHead(pl\_queue, pl\_toChain, pl\_item)*

*f\_EPTF\_NQueue\_moveToTail(pl\_queue, pl\_toChain, pl\_item)*

The parameter *pl\_toChain* is the target chain, and can be the same chain in which the item is before the move.

### Moving an Item after or before another item

The following functions can be used to move an item after or before another item:

f\_EPTF\_NQueue\_moveAfter(pl\_queue, pl\_afterIdx, pl\_item)

f\_EPTF\_NQueue\_moveBefore(pl\_queue, pl\_beforeIdx, pl\_item)

The parameter *pl\_afterIdx* and *pl\_beforeIdx* is the item index after or before which to move the item at *pl\_item* respectively. The target chain of the item is the chain of the item after or before which it is moved to.

### Getting the chain of an item

The following function returns the ID of the chain in which the item is linked:

*f\_EPTF\_NQueue\_getChainOfItem(pl\_queue, pl\_item)*

### Getting the length of a queue

The following function returns the length of (i.e. number of elements in) a queue:

*f\_EPTF\_NQueue\_getLengthOfQueue(pl\_queue)*

### Getting the length of a chain

The following function returns the length of a chain:

*f\_EPTF\_NQueue\_getLengthOfChain(pl\_queue, pl\_chain)*

### Getting the head item of a chain

The following function returns the index of the head (first) item in a chain or -1 if the chain is empty:

*f\_EPTF\_NQueue\_getHeadOfChain(pl\_queue, pl\_chain)*

### Getting the tail item of a chain

The following function returns the index of the tail (last) item in a chain or -1 if the chain is empty:

*f\_EPTF\_NQueue\_getTailOfChain(pl\_queue, pl\_chain)*

### Iterating through a chain

The function *f\_EPTF\_NQueue\_getNextItemIdx(pl\_queue, pl\_item)* can be used to iterate forward, *f\_EPTF\_NQueue\_getPrevItemIdx(pl\_queue, pl\_item)* can be used to iterate backward through the busy chain from a given index. These functions modify the inout parameter *pl\_item* in case there is a next or previous item, and return true. If there is no more items in the chain, they return false.

### Logging functions

Following functions can be used to log information about the queue for debugging:

*f\_EPTF\_NQueue\_logQueue(pl\_queue)*

*f\_EPTF\_NQueue\_logChain(pl\_queue, pl\_chain)*

*f\_EPTF\_NQueue\_logChainFields(pl\_queue, pl\_chain)*

### Dumping a queue to a PNG image

The function *f\_EPTF\_NQueue\_dumpToPng(pl\_queue, pl\_name)* can be used to save a queue as a digraph file and generate a PNG image using the tool ‘dot’ (which has to be installed to use this function, see [6]). This is for testing purposes, the number of items in the queue should not be too large due to the limitations of ‘dot’.

## Summary Table of all public functions for EPTF NQueue

Table 1. Summary of NQueue functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_NQueue\_init\_CT | Initializes the NQueue feature |
| f\_EPTF\_NQueue\_createQueue | Creates a new queue |
| f\_EPTF\_NQueue\_deleteQueue | Deletes a queue |
| f\_EPTF\_NQueue\_createChain | Creates a chain in a queue |
| f\_EPTF\_NQueue\_createItemAtChainHead | Creates one item at chain head |
| f\_EPTF\_NQueue\_createItemAtChainTail | Creates one item at chain tail |
| f\_EPTF\_NQueue\_createItemAfter | Creates one item after another |
| f\_EPTF\_NQueue\_createItemBefore | Creates one item before another |
| f\_EPTF\_NQueue\_createItemsAtChainHead | Creates items at chain head |
| f\_EPTF\_NQueue\_createItemsAtChainTail | Creates items at chain tail |
| f\_EPTF\_NQueue\_moveToHead | Moves an item to chain head |
| f\_EPTF\_NQueue\_moveToTail | Moves an item to chain tail |
| f\_EPTF\_NQueue\_moveAfter | Moves an item after another |
| f\_EPTF\_NQueue\_moveBefore | Moves an item before another |
| f\_EPTF\_NQueue\_getChainOfItem | Returns the chain of the item |
| f\_EPTF\_NQueue\_getLengthOfQueue | Returns the length of the queue |
| f\_EPTF\_NQueue\_getLengthOfChain | Returns the length of a chain |
| f\_EPTF\_NQueue\_getHeadOfChain | Returns the head of a chain |
| f\_EPTF\_NQueue\_getTailOfChain | Returns the tail of a chain |
| f\_EPTF\_NQueue\_getNextItemIdx | Forward iterator |
| f\_EPTF\_NQueue\_getPrevItemIdx | Backward iterator |
| f\_EPTF\_NQueue\_logQueue | Logs the queue |
| f\_EPTF\_NQueue\_logChain | Logs a chain with items |
| f\_EPTF\_NQueue\_logChainFields | Logs the chain without items |
| f\_EPTF\_NQueue\_dumpToPng | Dumps the queue to PNG |