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

EPTF CLL Free Busy Queue, Function Description

Contents

[1 Introduction 2](#_Toc320534351)

[1.1 Revision history 2](#_Toc320534352)

[1.2 How to Read this Document 2](#_Toc320534353)

[1.3 References 2](#_Toc320534354)

[1.4 Scope 2](#_Toc320534355)

[1.5 Recommended way of reading 2](#_Toc320534356)

[1.6 Typographical conventions 3](#_Toc320534357)

[1.7 Abbreviations 3](#_Toc320534358)

[1.8 Terminology 3](#_Toc320534359)

[2 General Description 3](#_Toc320534360)

[2.1 Free Busy Queues for resource allocation 4](#_Toc320534361)

[2.2 Free Busy Queues for data sorting 4](#_Toc320534362)

[3 Functional Interface 4](#_Toc320534363)

[3.1 Naming Conventions 4](#_Toc320534364)

[3.2 Public Functions 4](#_Toc320534365)

[3.2.1 Initialization of the Free Busy Queue feature 4](#_Toc320534366)

[3.2.2 Initialization of a Free Busy Queue instance 5](#_Toc320534367)

[3.2.3 Removing a Free Busy Queue instance 5](#_Toc320534368)

[3.2.4 Setting/Getting Data Indices 5](#_Toc320534369)

[3.2.5 Getting or Creating Free Slots 5](#_Toc320534370)

[3.2.6 Moving a Free Slot to the Busy Chain 6](#_Toc320534371)

[3.2.7 Moving a Busy Slot to the Free Chain 6](#_Toc320534372)

[3.2.8 Invalid elements 6](#_Toc320534373)

[3.2.9 Getting the Head Index of the Busy Chain 6](#_Toc320534374)

[3.2.10 Iterating Through the Busy Chain 7](#_Toc320534375)

[3.2.11 Checking the State of an Item 7](#_Toc320534376)

[3.2.12 Getting the Length of a Chain 7](#_Toc320534377)

[3.2.13 Logging a Chain 7](#_Toc320534378)

[3.3 Summary Table of all public functions for EPTF Free Busy Queue 7](#_Toc320534379)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-26 | PA1 | First draft version | EGBOTAT |
| 2007-12-06 | PA2 | Final version after review | EGBOTAT |
| 2012-02-21 | PB1 | Based on NQueue feature | ETHJGI |
|  |  |  |  |

## How to Read this Document

This is the Function Description for the Free Busy Queue 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. TitanSim CLL for TTCN-3 toolset with TITAN, Reference Guide  
   <http://ttcn.ericsson.se/products/libraries.shtml>

## Scope

This document is to specify the content and functionality of the Free Busy Queue 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

CLL Core Load Library

FBQ Free Busy Queue

EPTF Ericsson Load Test Framework, formerly TITAN Load Test Framework

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

# General Description

This document specifies the Free Busy Queue feature of the TitanSim CLL.

The EPTF Free Busy Queue feature makes it possible to

* Manage linked lists (“queues”) over ‘record of’ data structures easily using index-arithmetics
  + Provide efficient iterators over the elements linked into the same list
  + Users can optionally store associated integer attributes with queue elements
* Book-keep the allocated (“busy”) and free status of elements of a ‘record of’ type efficiently
  + Provide efficient methods for moving elements back and forth between “free” and “busy” chains with computational complexity of O(1) (that is, constant), regardless of the length of the queue
  + Maintain up-to-date additional statistics about the number of elements in the “free”, and “busy” chain, respectively

The aim of the EPTF Free Busy Queue feature is to provide dynamic memory allocation for the TTCN-3 language in an efficient way.

## Free Busy Queues for resource allocation

Typically, a Free Busy Queue has an associated data array (record of a user defined type), with the same length as the queue. Each slot of the queue can have either busy or free state, also representing the allocation state of elements in the corresponding data array with the same index. Free and busy slots are chained in a free- and a busy queue respectively inside the Free Busy Queue. 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 Free Busy Queue to resize the data record-of in case the length of the queue increases (it never decreases). That is, when an index is returned from f\_EPTF\_FBQ\_getOrCreateFreeSlot that is bigger than the biggest index of the user data array.

Each slot in the queue has a record of integer field called dataidx, which can optionally be used either to point in a user defined data structure (data indices) or to store data directly (in which case, the associated data record-of may not be needed). This “dataidx” mechanism can also be used for associating the elements of the Free Busy Queue with a user data array of different length (e.g., when used in conjunction with sparse arrays).

## Free Busy Queues for data sorting

Free Busy Queues 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 ‎[5].

## Naming Conventions

All functions have the prefix f\_EPTF\_FBQ.

## Public Functions

### Initialization of the Free Busy Queue feature

Before using the EPTF Free Busy Queue feature the

*f\_EPTF\_FBQ\_initFreeBusyQueue(pl\_selfName)*

function should be called with the name of the component as the parameter. This initializes the Free Busy queue feature so that other API functions can be used.

### Initialization of a Free Busy Queue instance

To initialize a EPTF Free Busy Queue the

*f\_EPTF\_FBQ\_initFreeBusyQueue(pl\_queue)*

function should be called with the queue as the parameter. This initializes the queue to empty. Other Free Busy Queue functions operate on the queue instance created by this function.

The Free Busy Queue feature is capable of creating as many Free Busy queues as needed.

### Removing a Free Busy Queue instance

To delete a Free Busy Queue that is no longer needed can be done by the function

*f\_EPTF\_FBQ\_deleteFreeBusyQueue(pl\_queue)*

This function deletes the queue and the associated data as well.

### Setting/Getting Data Indices

The functions *f\_EPTF\_FBQ\_setQueueDidx(pl\_i, pl\_j, pl\_queue)* and *f\_EPTF\_FBQ\_getQueueDidx(pl\_idx, pl\_queue, pl\_didx)* can be used to set and get data indices of an element in the queue.

### Getting or Creating Free Slots

The function *f\_EPTF\_FBQ\_getOrCreateFreeSlot(pl\_queue)* can be used to get a free slot from a queue. The function resizes the queue if there are no free slots available.

Function *f\_EPTF\_FBQ\_createFreeSlots(pl\_number, pl\_queue)* can be used to create several new slot items at once in the free chain of a queue.

Function *f\_EPTF\_FBQ\_getFreeSlot(pl\_queue)* can be used to get the head of the free chain. This function cannot resize the queue, and returns with a negative index if there are no free slots.

### Moving a Free Slot to the Busy Chain

The following functions can be used to move a slot from the free to the busy chain:

*f\_EPTF\_FBQ\_moveFromFreeHeadToBusy(pl\_idx, pl\_location, pl\_afteridx, pl\_beforeidx, pl\_queue)* can be used to move the head of the free chain to somewhere into the busy chain.

*f\_EPTF\_FBQ\_moveFromFreeHeadToBusyTail(pl\_queue)* can be used to move the head of the free chain to the tail of the busy chain.

*f\_EPTF\_FBQ\_moveFromFreeToBusyTail(pl\_idx, pl\_queue)* can be used to move an arbitrary slot from the free chain to the tail of the busy chain.

### Moving a Busy Slot to the Free Chain

The following functions can be used to move a slot from the busy to the free chain:

*f\_EPTF\_FBQ\_moveFromBusyToFreeHead(pl\_idx, pl\_queue)* moves an item from the busy chain to the head of the free chain.

*f\_EPTF\_FBQ\_moveFromBusyToFreeTail(pl\_idx, pl\_queue)* moves an item from the busy chain to the tail of the free chain.

### Invalid elements

Items in the Free Busy Queue have a third state, called ‘invalid’. Invalid items are not linked into any of the free or busy chains. They are linked into the invalid chain.

The function *f\_EPTF\_FBQ\_moveFromBusyToInvaild(pl\_idx, pl\_queue)* can be used to move an item from the busy chain to invalid state, i.e. to unchain it. It is the responsibility of the user to keep track of such items; and link them eventually back to one of the free or busy chains.

Function *f\_EPTF\_FBQ\_moveFromInvalidToFreeHead(pl\_idx, pl\_queue)* can be used to move an invalid item to the head of the free chain.

### Getting the Head Index of the Busy Chain

The function *f\_EPTF\_FBQ\_getBusyHeadIdx(pl\_idx, pl\_queue)* can be used to get the head element of the busy chain.

### Iterating Through the Busy Chain

The function *f\_EPTF\_FBQ\_getFwdBusyItemIdx(pl\_idx, pl\_queue)* can be used to iterate forward, *f\_EPTF\_FBQ\_getBwdBusyItemIdx(pl\_idx, pl\_queue)* can be used to iterate backward through the busy chain from a given index.

### Checking the State of an Item

The following functions can be used to check if an item is in a given state:

*f\_EPTF\_FBQ\_itemIsBusy(pl\_idx, pl\_queue)* returns true if the item is in the busy chain,

*f\_EPTF\_FBQ\_itemIsFree(pl\_idx, pl\_queue)* returns true if the item is in the free chain,

*f\_EPTF\_FBQ\_itemIsInvalid(pl\_idx, pl\_queue)* returns true if the item is invalid.

### Getting the Length of a Chain

The functions *f\_EPTF\_FBQ\_getLengthOfBusyChain(pl\_queue)* and *f\_EPTF\_FBQ\_getLengthOfFreeChain(pl\_queue)* can be used to get the length of the busy and the free chain respectively. The number of all items (including the invalid ones) can be determined by the function *f\_EPTF\_FBQ\_getLengthOfQueue(pl\_queue)*.

### Logging a Chain

The function *f\_EPTF\_FBQ\_logChain(pl\_chain, pl\_queue)* can be used to log one of the chains (for debugging).

## Summary Table of all public functions for EPTF Free Busy Queue

Table 1. Summary of Free Busy Queue functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_FBQ\_init\_CT | Initializes the FBQ feature |
| f\_EPTF\_FBQ\_initFreeBusyQueue | Initializes a queue to empty |
| f\_EPTF\_FBQ\_deleteFreeBusyQueue | Deletes the queue and the associated data |
| f\_EPTF\_FBQ\_setQueueDidx | Sets the data indices of an item |
| f\_EPTF\_FBQ\_getQueueDidx | Gets the data indices of an item |
| f\_EPTF\_FBQ\_getOrCreateFreeSlot | Returns an index to an existing or a newly created free item |
| f\_EPTF\_FBQ\_createFreeSlots | Creates free slots |
| f\_EPTF\_FBQ\_getFreeSlot | Returns the head of the free chain |
| f\_EPTF\_FBQ\_moveFromFreeHeadToBusy | Moves the head of the free chain to somewhere in the busy chain |
| f\_EPTF\_FBQ\_moveFromFreeHeadToBusyTail | Moves the head of the free chain to the tail of the bust chain |
| f\_EPTF\_FBQ\_moveFromFreeToBusyTail | Moves an element from the free chain to the tail of the busy chain |
| f\_EPTF\_FBQ\_moveFromBusyToFreeHead | Moves an item from the busy chain to the head of the free chain |
| f\_EPTF\_FBQ\_moveFromBusyToFreeTail | Moves a busy slot to the tail of the free chain |
| f\_EPTF\_FBQ\_moveFromBusyToInvalid | Unchains a busy slot |
| f\_EPTF\_FBQ\_moveFromInvalidToFreeHead | Moves an invalid slot to the head of the free chain |
| f\_EPTF\_FBQ\_getBusyHeadIdx | Returns the head of the busy chain |
| f\_EPTF\_FBQ\_getFwdBusyItemIdx | Gets the next item in the busy chain from a given index |
| f\_EPTF\_FBQ\_getBwdBusyItemIdx | Gets the previous item in the busy chain from a given index |
| f\_EPTF\_FBQ\_itemIsBusy | Returns true if the item is in the busy chain |
| f\_EPTF\_FBQ\_itemIsFree | Returns true if the item is in the free chain |
| f\_EPTF\_FBQ\_itemIsInvalid | Returns true if the item is in none of the chains |
| f\_EPTF\_FBQ\_getLengthOfBusyChain | Returns the length of the busy chain |
| f\_EPTF\_FBQ\_getLengthOfFreeChain | Returns the length of the free chain |
| f\_EPTF\_FBQ\_getLengthOfQueue | Returns the number of all items in the queue |
| f\_EPTF\_FBQ\_logChain | Logs a chain for debugging |