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

EPTF CLL Ring Buffer Support, Function Description

Contents

[1 Introduction 2](#_Toc211150535)

[1.1 Revision history 2](#_Toc211150536)

[1.2 How to Read this Document 2](#_Toc211150537)

[1.3 References 2](#_Toc211150538)

[1.4 Scope 2](#_Toc211150539)

[1.5 Recommended way of reading 2](#_Toc211150540)

[1.6 Typographical conventions 3](#_Toc211150541)

[1.7 Abbreviations 3](#_Toc211150542)

[1.8 Terminology 3](#_Toc211150543)

[2 General Description 3](#_Toc211150544)

[3 Functional Interface 4](#_Toc211150545)

[3.1 Naming Conventions 4](#_Toc211150546)

[3.2 Public Functions 4](#_Toc211150547)

[3.2.1 Initialization 4](#_Toc211150548)

[3.2.2 Insert and Remove Elements 4](#_Toc211150549)

[3.2.3 Check Buffer Size 5](#_Toc211150550)

[3.2.4 Access to Elements 5](#_Toc211150551)

[3.2.5 Convenience Functions 5](#_Toc211150552)

[3.3 Summary Table of all public functions for EPTF Ring Buffer Support 5](#_Toc211150553)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-09 | PA1 | First draft version | EISTFAL |
| 2007-11-21 | PA2 | Updated after eview | EISTFAL |
|  |  |  |  |
|  |  |  |  |

## How to Read this Document

This is the Function Description for the Ring Buffer Support 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 Ring Buffer Support 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 ‎1.5 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

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 **Error! Reference source not found.**

## 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

TitanSim Ring Buffer component provides a general ring buffer (or circular buffer) implementation. Ring buffer is a [data structure](http://en.wikipedia.org/wiki/Data_structure) that uses a single, fixed-size [buffer](http://en.wikipedia.org/wiki/Buffer_%28computer_science%29) as if it were connected end-to-end. This structure can be easily used for buffering [data streams](http://en.wikipedia.org/wiki/Data_stream).

TitanSim Ring Buffer is a sequence of elements with maximized size, called capacity. It allows random access to elements and provides constant time insertion and removal at both front-end and back-end of the sequence. If you insert a new element to a full buffer (i.e. number of elements is equal with buffer capacity), the element at the other end is being overwritten.

TitanSim Ring Buffer component provides ring buffer implementation both for integer and generic types. Generic (type independent) ring buffer support is implemented through macro-preprocessor of C++ compiler.

# 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 for managing a ring buffer built from integer values have the prefix f\_EPTF\_RB\_. Functions to manage a generic ring buffer (which stores values of a user defined type) have the prefix <USER\_TYPE>RingBuffer, where <USER\_TYPE> is the name of the user defined type.

## Public Functions

### Initialization

f\_EPTF\_RB\_init() function shall be used to initialize a new ring buffer.

<USER\_TYPE>RingBuffer\_init() function initializes a new generic ring buffer instance.

The capacity of the new ring buffer can be given by the user, or the default value is used defined by c\_EPTF\_RB\_default\_capacity constant.

### Insert and Remove Elements

New elements can be inserted and removed with constant time at both front-end and back-end of the buffer.

f\_EPTF\_RB\_push\_back() inserts new element at the back-end of an integer ring buffer. <USER\_TYPE>RingBuffer\_push\_back() shall be used for same purpose in case of generic ring buffers.

f\_EPTF\_RB\_push\_front() and <USER\_TYPE>RingBuffer\_push\_front() functions are used to insert new element at front-end of ring buffer.

Elements can be removed from front-end by f\_EPTF\_RB\_pop\_front() and <USER\_TYPE>RingBuffer\_pop\_front() functions.

For removing element from back-end of ring buffer f\_EPTF\_RB\_pop\_back() and <USER\_TYPE>RingBuffer\_pop\_back() functions shall be used.

### Check Buffer Size

f\_EPTF\_RB\_size() and <USER\_TYPE>RingBuffer\_size() functions return the number of the stored elements.

f\_EPTF\_RB\_empty() and <USER\_TYPE>RingBuffer\_empty() functions return true, if the given ring buffer is empty.

f\_EPTF\_RB\_capacity() and <USER\_TYPE>RingBuffer\_capacity() functions return the largest possible size of the ring buffer. This value is set by the initialization of the ring buffer.

### Access to Elements

f\_EPTF\_RB\_front() and <USER\_TYPE>RingBuffer\_front() functions return the first (eldest) element of the ring buffer.

f\_EPTF\_RB\_back() and <USER\_TYPE>RingBuffer\_back() functions return the last (latest) element of the ring buffer.

f\_EPTF\_RB\_get() and <USER\_TYPE>RingBuffer\_get() functions return the element at a specific index.

**Note that these functions above do not perform any check before accessing to an elements, i.e. the user should check the boundaries of ring buffer before using them! Calling front() or back() on an empty ring buffer, or using get() with invalid index results undefined behaviour!**

Safe access is implemented by f\_EPTF\_RB\_at() and <USER\_TYPE>RingBuffer\_at() functions. They check the validity of the given index and return false, if element is not stored at the specific index.

### Convenience Functions

The content of a ring buffer can be dumped into a simple data array from front-end to back-end. For this purpose f\_EPTF\_RB\_dump() and <USER\_TYPE>RingBuffer\_dump() functions are implemented.

## Summary Table of all public functions for EPTF Ring Buffer Support

Table 1. Summary of Ring Buffer Support functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_RB\_init | Initialize ring buffer of integers |
| <USER\_TYPE>RingBuffer\_init | Initialize generic ring buffer |
| f\_EPTF\_RB\_size | Returns the number of stored element |
| <USER\_TYPE>RingBuffer\_size | Returns the number of stored element |
| f\_EPTF\_RB\_capacity | Returns the largest possible size |
| <USER\_TYPE>RingBuffer\_capacity | Returns the largest possible size |
| f\_EPTF\_RB\_empty | Checks buffer is empty |
| <USER\_TYPE>RingBuffer\_empty | Checks buffer is empty |
| f\_EPTF\_RB\_push\_back | Insert a new element at the back-end |
| <USER\_TYPE>RingBuffer\_push\_back | Insert a new element at the back-end |
| f\_EPTF\_RB\_push\_front | Insert a new element at the front-end |
| <USER\_TYPE>RingBuffer\_push\_front | Insert a new element at the front-end |
| f\_EPTF\_RB\_pop\_front | Remove element at the front-end |
| <USER\_TYPE>RingBuffer\_pop\_front | Remove element at the front-end |
| f\_EPTF\_RB\_pop\_back | Remove element at the back-end |
| <USER\_TYPE>RingBuffer\_pop\_back | Remove element at the back-end |
| f\_EPTF\_RB\_front | Return element at the front-end (unchecked access!) |
| <USER\_TYPE>RingBuffer\_front | Return element at the front-end (unchecked access!) |
| f\_EPTF\_RB\_back | Return element a the back-end (unchecked access!) |
| <USER\_TYPE>RingBuffer\_back | Return element at the back-end (unchecked access!) |
| f\_EPTF\_RB\_get | Return element at a specific index (unchecked access!) |
| <USER\_TYPE>RingBuffer\_get | Return element at a specific index (unchecked access!) |
| f\_EPTF\_RB\_at | Return element at a specific index (boundaries checked) |
| <USER\_TYPE>RingBuffer\_at | Return element at a specific index (boundaries checked) |
| f\_EPTF\_RB\_dump | Dumps content to a simple array |
| <USER\_TYPE>RingBuffer\_dump | Dumps content to a simple array |