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

EPTF CLL Red Black Tree, Function Description

Contents

[1 Introduction 2](#_Toc310943698)

[1.1 Revision history 2](#_Toc310943699)

[1.2 How to Read this Document 2](#_Toc310943700)

[1.3 References 2](#_Toc310943701)

[1.4 Scope 2](#_Toc310943702)

[1.5 Recommended way of reading 3](#_Toc310943703)

[1.6 Typographical conventions 3](#_Toc310943704)

[1.7 Abbreviations 3](#_Toc310943705)

[1.8 Terminology 3](#_Toc310943706)

[2 General Description 4](#_Toc310943707)

[3 Functional Interface 5](#_Toc310943708)

[3.1 Initialization 6](#_Toc310943709)

[3.2 Creating a Tree 6](#_Toc310943710)

[3.3 Deleting a Tree 6](#_Toc310943711)

[3.4 Inserting an Item 6](#_Toc310943712)

[3.5 Removing an Item 7](#_Toc310943713)

[3.6 Getting the Smallest Item 7](#_Toc310943714)

[3.7 Getting the Key of an Item 7](#_Toc310943715)

[3.8 Finding an Item by Key 7](#_Toc310943716)

[3.9 Other Getter Functions 8](#_Toc310943717)

[3.10 Iterating through Items 8](#_Toc310943718)

[3.11 Sorting Items 8](#_Toc310943719)

[3.12 Summary Table of all public functions for EPTF Red Black Tree 8](#_Toc310943720)

[4 Obsolete Functional Interface 9](#_Toc310943721)

[4.1 Naming Conventions 10](#_Toc310943722)

[4.2 Public Functions 10](#_Toc310943723)

[4.2.1 Initialization 10](#_Toc310943724)

[4.2.2 Creating a New Node 10](#_Toc310943725)

[4.2.3 Removing a Node 10](#_Toc310943726)

[4.2.4 Getting Elements from the Tree 10](#_Toc310943727)

[4.3 Summary Table of obsolete functions for EPTF Red Black Tree 11](#_Toc310943728)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-27 | PA1 | First draft version | EGBOTAT |
| 2007-12-06 | PA2 | Final version after review | EGBOTAT |
| 2011-09-27 | PB1 | Added refactored Red Black Tree | EGBOTAT |
|  |  |  |  |

## How to Read this Document

This is the Function Description for the Red Black Tree 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>
6. 6/155 16-CNL 113 512 Uen  
   EPTF Free Busy Queue, Function Description
7. Red-black tree, wikipedia  
   <http://en.wikipedia.org/wiki/Red-black_tree>

## Scope

This document is to specify the content and functionality of the Red Black Tree 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

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

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

*Item* An item in the red black tree. Can be a node, a chain item or a free item. It is uniquely identified by its index. An EPTF red black tree can contain multiple items with the same key.

*Node Item* A node of the red black tree. If there are more than one items in the tree with the same key, subsequently added items are stored in a linked list, with the node item being the head of the list.

*Chain Item* An item that is stored in a linked list and is not the head of the list (i.e. the node itself). Its key equals the key of the node item it is chained after.

*Free Item* An item that was previously removed from the tree, it will be reused in a subsequent insert operation.

*Key* Thekey is a non-unique identifier of an item in the tree. Each item in the tree is added with a specific key and the first item added with a given key can be searched for. Items in a tree can be sorted by their keys (and addition order in case of multiple items with the same key).

# General Description

This document specifies the Red Black Tree feature of the TitanSim CLL.

The EPTF Red Black Tree feature is a self-balancing binary search tree, for details see [7]. It can be used to store indexes into an associated array, similar to the EPTF Free Busy Queue ‎[6].

Each node in the EPTF Red Black Tree is a linked list. This makes it possible to store more than one item with the same key, in the order of the insertion. Each item in the tree (or the lists) has a unique ID/index, which is allocated incrementally with each item-insertion into the tree. Removed items are reused in subsequent insert operations.

Inserting new item in the Red Black Tree is slightly slower, but searching for a specific item is faster than with the Free Busy Queue with a large number of items. Furthermore, the Red Black Tree can be used to sort items or to iterate through items in order.

root

smallest

free head

free count

items

0

1

2

3

4

5

6

7

8

9

10

11

associated array

0

1

2

3

4

5

6

7

8

9

10

11

**...**

Red Black Tree

free item

node item

chain item

Figure 1 Overview of the Red Black Tree

Figure 1 shows an example overview of the internal data representation of a Red Black Tree and its associated array (item keys and leaf/null nodes are not shown). The links between tree nodes, chain items and the free items are all stored in the items themselves. Any change in the chaining (e.g. tree rotation) does not modify the index of the items.

Items removed are stored in a stack (free stack or free chain), and are reused in reverse order of their removal when inserting items (if there are no available pre-allocated items).

In the example above, there are 6 nodes in the tree, meaning that 6 individual keys were used when inserting items. The node at item 9 has a chain length of three, i.e. three items with this key were inserted: 9, 10 and 11 in this order (note: the head and tail of this chain refers to each other in order to speed up insertion at the end of the chain). Sorting the items would result in index list 1, 3, 2, 7, 6, 9, 10 and 11. The next item inserted (assuming there are no pre-allocated items) would be item 0 (and the free head would then point to item 4).

# Functional Interface

Currently there are three kinds of trees in the Red Black Tree feature, distinguished by the key type of the items. These are:

* integer tree
* float tree
* charstring tree

There are separate functions for each tree type if the function has the key as parameter or if it returns the key. These functions refer to the key type with their name (e.g. f\_EPTF\_RBT\_insert**Int**Item)

Other functions that don’t depend on the key type are unified (e.g. f\_EPTF\_RBT\_removeItem).

The component EPTF\_RBT\_CT must be extended and initialized (3.1) to use the Red Black Tree feature. The external functions in this feature do not have a runs-on clause, but they must be handled as if they would “runs on EPTF\_RBT\_CT”.

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

Apart from this description a cross-linked reference guide for the TitanSim CLL Functions can be reached for on-line reading ‎[5].

## Initialization

The function f\_EPTF\_RBT\_init\_CT(pl\_selfName) must be called before using the Red Black Tree feature.

## Creating a Tree

The following functions can be used to create a new Red Black Tree:

f\_EPTF\_RBT\_createIntTree(pl\_name) return EPTF\_RBT\_TreeId

f\_EPTF\_RBT\_createFloatTree(pl\_name) return EPTF\_RBT\_TreeId

f\_EPTF\_RBT\_createCharstringTree(pl\_name) return EPTF\_RBT\_TreeId

## Deleting a Tree

The function f\_EPTF\_RBT\_deleteTree(pl\_tree) can be used to delete a tree and free all its resources.

## Inserting an Item

The following functions can be used to insert (create) a new item in a tree:

f\_EPTF\_RBT\_insertIntItem(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

f\_EPTF\_RBT\_insertFloatItem(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

f\_EPTF\_RBT\_insertCharstringItem(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

## Removing an Item

The function f\_EPTF\_RBT\_removeItem(pl\_tree, pl\_item) can be used to remove an item from a tree. Removed items will be reused in subsequent insertions.

## Getting the Smallest Item

The function f\_EPTF\_RBT\_getItemWithSmallestKey(pl\_tree) returns the index of the item with the smallest key in the tree, or −1 if the tree is empty. The index of the smallest item is cached, and its cache is updated at both insertion and removal in a way that searching for the smallest item is never needed.

## Getting the Key of an Item

The following functions return the key of an item in a tree:

f\_EPTF\_RBT\_getKeyOfIntItem(pl\_tree, pl\_item) return integer

f\_EPTF\_RBT\_getKeyOfFloatItem(pl\_tree, pl\_item) return float

f\_EPTF\_RBT\_getKeyOfCharstringItem(pl\_tree, pl\_item) return charstring

## Finding an Item by Key

The following functions return the index of the first item with a given key or −1 if no item is found:

f\_EPTF\_RBT\_findFirstItemByIntKey(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

f\_EPTF\_RBT\_findFirstItemByFloatKey(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

f\_EPTF\_RBT\_findFirstItemByCharstringKey(pl\_tree, pl\_key) return EPTF\_RBT\_ItemIdx

Use f\_EPTF\_RBT\_getNextInChain to get further items with the same keys (if any), see 3.9.

## Other Getter Functions

In addition to the functions in previous sections, the following getter functions are defined:

* f\_EPTF\_RBT\_getRoot(pl\_tree): returns the root node or −1 if tree is empty
* f\_EPTF\_RBT\_getLeft(pl\_tree, pl\_item): returns the left child of a node or −1 if none (no child or item is not node)
* f\_EPTF\_RBT\_getRight(pl\_tree, pl\_item): returns the right child of a node or −1 if none (no child or item is not node)
* f\_EPTF\_RBT\_getParent(pl\_tree, pl\_item): returns the parent of a node or −1 if none (item is root or not a node)
* f\_EPTF\_RBT\_getNextInChain(pl\_tree, pl\_item): returns the next item with the same key as the key of pl\_item or −1 if none
* f\_EPTF\_RBT\_getPrevInChain(pl\_tree, pl\_item): returns the previous item with the same key as the key of pl\_item or −1 if none

## Iterating through Items

The function f\_EPTF\_RBT\_iterateIncremental(pl\_tree, pl\_item) can be used to iterate through the items of the tree, including items with the same key (in insertion order).

## Sorting Items

The following function returns the item order (including items with same key) in its out parameter pl\_itemsSorted:

f\_EPTF\_RBT\_sortIncremental(pl\_tree, pl\_itemsSorted)

## Summary Table of all public functions for EPTF Red Black Tree

Table : Summary of Red Black Tree functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_RBT\_init\_CT | Init the Red Black Tree feature |
| f\_EPTF\_RBT\_createIntTree | Create an integer tree |
| f\_EPTF\_RBT\_createFloatTree | Create a float tree |
| f\_EPTF\_RBT\_createCharstringTree | Create a charstring tree |
| f\_EPTF\_RBT\_deleteTree | Delete a tree |
| f\_EPTF\_RBT\_insertIntItem | Insert an item with integer key |
| f\_EPTF\_RBT\_insertFloatItem | Insert an item with float key |
| f\_EPTF\_RBT\_insertCharstringItem | Insert an item with charstring key |
| f\_EPTF\_RBT\_removeItem | Remove an item |
| f\_EPTF\_RBT\_getItemWithSmallestKey | Get item with the smallest key |
| f\_EPTF\_RBT\_getKeyOfIntItem | Get integer key of an item |
| f\_EPTF\_RBT\_getKeyOfFloatItem | Get float key of an item |
| f\_EPTF\_RBT\_getKeyOfCharstringItem | Get charstring key of an item |
| f\_EPTF\_RBT\_findFirstItemByIntKey | Find item by integer key |
| f\_EPTF\_RBT\_findFirstItemByFloatKey | Find item by float key |
| f\_EPTF\_RBT\_findFirstItemByCharstringKey | Find item by charstring key |
| f\_EPTF\_RBT\_getRoot | Get root node |
| f\_EPTF\_RBT\_getLeft | Get left child node |
| f\_EPTF\_RBT\_getRight | Get right child node |
| f\_EPTF\_RBT\_getNextInChain | Get next item with same key |
| f\_EPTF\_RBT\_getPrevInChain | Get previous item with same key |
| f\_EPTF\_RBT\_iterateIncremental | Iterate through the items |
| f\_EPTF\_RBT\_sortIncremental | Sort the items |

# Obsolete Functional Interface

Currently, the following **obsolete** functionality is also included. Please note that these functions will be removed in a later release of the Core Load Library.

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 integer Red Black Tree functions have the prefix f\_EPTF\_RBTreeI.

All float Red Black Tree functions have the prefix f\_EPTF\_RBTreeF.

## Public Functions

### Initialization

Before using the EPTF Red Black Tree functions the

*f\_EPTF\_RBTreeI\_initIntegerTree(rb\_Tree)* or

*f\_EPTF\_RBTreeF\_initFloatTree(rb\_Tree)*

function should be called. This initializes the Red Black Tree passed as the parameter.

### Creating a New Node

The function *f\_EPTF\_RBTreeI\_createNewIntegerNode(rb\_Tree, key, data)* or *f\_EPTF\_RBTreeF\_createNewFloatNode(rb\_Tree, key, data)* can be used to create a new node in the tree.

### Removing a Node

The following functions can be used to remove a node from the tree:

*f\_EPTF\_RBTreeI\_removeIntegerNodeWithSameKey(rb\_Tree, pl\_nodeToDeleteIndex)*

*f\_EPTF\_RBTreeI\_removeIntegerNode(rb\_Tree, pl\_z, pl\_destroyNode)*

*f\_EPTF\_RBTreeF\_removeFloatNodeWithSameKey(rb\_Tree, pl\_nodeToDeleteIndex)*

*f\_EPTF\_RBTreeF\_removeFloatNode(rb\_Tree, pl\_z, pl\_destroyNode)*

### Getting Elements from the Tree

The following functions can be used to get elements from the tree:

*f\_EPTF\_RBTreeI\_getIntegerElementNodeByKey(rb\_Tree, key, nodeFound, indexFound)*

*f\_EPTF\_RBTreeI\_searchSmallestIntegerNode(tree, nodeFound, indexFound)*

*f\_EPTF\_RBTreeI\_getSmallestNodeFromCache(rb\_Tree, nodeFound)*

*f\_EPTF\_RBTreeI\_getBusyEventHeadIndex(rb\_Tree, idx)*

*f\_EPTF\_RBTreeF\_getFloatElementNodeByKey(rb\_Tree, key, nodeFound, indexFound)*

*f\_EPTF\_RBTreeF\_searchSmallestFloatNode(tree, nodeFound, indexFound)*

*f\_EPTF\_RBTreeF\_getSmallestNodeFromCache(rb\_Tree, nodeFound)*

*f\_EPTF\_RBTreeF\_getBusyEventHeadIndex(rb\_Tree, idx)*

*f\_EPTF\_RBTreeF\_getNextSmallestFloatElementIndex(pl\_tree, pl\_curSmallestIndex, pl\_indexFound)*

## Summary Table of obsolete functions for EPTF Red Black Tree

Table : Summary of obsolete Red Black Tree functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_RBTreeI\_initIntegerTree | initialize an integer red black tree |
| f\_EPTF\_RBTreeF\_initFloatTree | initialize a float red black tree |
| f\_EPTF\_RBTreeI\_createNewIntegerNode | create a new integer node |
| f\_EPTF\_RBTreeF\_createNewFloatNode | create a new float node |
| f\_EPTF\_RBTreeI\_removeIntegerNodeWithSameKey | remove a node |
| f\_EPTF\_RBTreeI\_removeIntegerNode | remove a node |
| f\_EPTF\_RBTreeF\_removeFloatNodeWithSameKey | remove a node |
| f\_EPTF\_RBTreeF\_removeFloatNode | remove a node |
| f\_EPTF\_RBTreeI\_getIntegerElementNodeByKey | get an element by key |
| f\_EPTF\_RBTreeI\_searchSmallestIntegerNode | get smallest element |
| f\_EPTF\_RBTreeI\_getSmallestNodeFromCache | get smallest element |
| f\_EPTF\_RBTreeI\_getBusyEventHeadIndex | get smallest element |
| f\_EPTF\_RBTreeF\_getFloatElementNodeByKey | get an element by key |
| f\_EPTF\_RBTreeF\_searchSmallestFloatNode | get smallest element |
| f\_EPTF\_RBTreeF\_getSmallestNodeFromCache | get smallest element |
| f\_EPTF\_RBTreeF\_getBusyEventHeadIndex | get smallest element |
| f\_EPTF\_RBTreeF\_getNextSmallestFloatElementIndex | get next smallest element |