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

EPTF CLL HashMap, Function Description

Contents

[1 Introduction 2](#_Toc422736746)

[1.1 Revision history 2](#_Toc422736747)

[1.2 How to Read this Document 2](#_Toc422736748)

[1.3 Scope 2](#_Toc422736749)

[1.4 Recommended way of reading 2](#_Toc422736750)

[1.5 Typographical conventions 2](#_Toc422736751)

[1.6 Abbreviations 2](#_Toc422736752)

[1.7 Terminology 3](#_Toc422736753)

[2 General Description 3](#_Toc422736754)

[3 Functional Interface 4](#_Toc422736755)

[3.1 Naming Conventions 4](#_Toc422736756)

[3.2 Public Functions 4](#_Toc422736757)

[3.2.1 Initialization 4](#_Toc422736758)

[3.2.2 Cleanup 4](#_Toc422736759)

[3.2.3 Creating a new HashMap 4](#_Toc422736760)

[3.2.4 Deleting the HashMap 5](#_Toc422736761)

[3.2.5 Deleting the HashMap by ID 5](#_Toc422736762)

[3.2.6 Getting the ID of HashMap 5](#_Toc422736763)

[3.2.7 Assignment of a HashMap 5](#_Toc422736764)

[3.2.8 Dumping the content of one HashMap 6](#_Toc422736765)

[3.2.9 Dumping the content of one HashMap by ID 6](#_Toc422736766)

[3.2.10 Dumping the content of all HashMap 6](#_Toc422736767)

[3.2.11 Inserting an element 6](#_Toc422736768)

[3.2.12 Updating an element 7](#_Toc422736769)

[3.2.13 Finding an element 7](#_Toc422736770)

[3.2.14 Erasing an element 8](#_Toc422736771)

[3.2.15 Clearing all elements 8](#_Toc422736772)

[3.2.16 Getting the used size of HashMap 8](#_Toc422736773)

[3.2.17 Resizing the HashMap 8](#_Toc422736774)

[3.2.18 Get the size of HashMap 9](#_Toc422736775)

[3.2.19 Getting the maximum size of HashMap 9](#_Toc422736776)

[3.2.20 Check the content of HashMap 9](#_Toc422736777)

[3.2.21 Get the hash value of a key 9](#_Toc422736778)

[3.2.22 To get the beginning element 10](#_Toc422736779)

[3.2.23 To get the next element 10](#_Toc422736780)

[3.3 Summary Table of all public functions for EPTF HashMap 10](#_Toc422736781)

[4 References 14](#_Toc422736782)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-16 | PA1 | First draft version | EBENMOL |
| 2008-12-05 | PA2 | Second draft version | EBENMOL |
| 2008-01-15 | PA3 | Final version after review | EBENMOL |
| 2009-09-04 | PB1 | Cleanup functions added | EGBOTAT |
| 2011-10-14 | PC1 | HashMap component added | ETHJGI |
| 2015-05-04 | PD1 | Chapter “3.2.17 Resizing the HashMap” is updated | EZOLZSI |
| 2015-06-19 | D | Updated for release | ESZILSZ |

## How to Read this Document

This is the Function Description for the EPTF CLL HashMap 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.

## Scope

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

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of TitanSim CLL [3]. They should get familiar with the list of acronyms and the glossary in Section 1.6 and 1.7, respectively.

## Typographical conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context. Moreover, whenever a concept is mentioned that has a special meaning as described in the Glossary (Section 1.7) of this document, then these occurrences are marked with an initial arrow, e.g., 🡪 TitanSim Core (Load) Library (CLL).

## 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

*Core 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

*HashMap* is a hashed associate container that associates object of type key with object of type data

# General Description

This document specifies the HashMap feature of the TitanSim CLL. The feature provides access to a well-tested, industry standard *GCC* HashMap implementation located in *STL*. For gcc versions above 4.0 the HashMap feature uses the std::tr1::unordered\_map library. *HashMap* is a hashed associate container that associates object of type key with object of type data. It works by transforming the key using a hash function into a hash, a number that is used as an index in an array, to locate the desired location ("bucket") where the values should be. Looking up an element in a HashMap by its key is efficient, so HashMap is useful for “dictionaries” where the order of elements is irrelevant. The key in HashMap must be unique.

This component supports the following key –> data type associations:

|  |  |
| --- | --- |
| Key type | Data type |
| Integer | Integer |
| Charstring | Integer |
| Octetstring | Integer |

To be able to use EPTF HashMap feature, the user component should extend the EPTF\_HashMap\_CT component and before using other hashMap functions, should call its initializing function f\_EPTF\_HashMap\_init\_CT.

# 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 integer to integer (key, data) functions have the prefix f\_EPTF\_int2int\_HashMap.

All charstring to integer (key, data) functions have the prefix f\_EPTF\_str2int\_HashMap.

All octetstring to integer (key, data) functions have the prefix f\_EPTF\_oct2int\_HashMap.

## Public Functions

### Initialization

Before using the EPTF HashMap functions the

f\_EPTF\_HashMap\_init\_CT(pl\_selfName)

has to be called. This initializes the EPTF HashMap feature.

Note, that the previous init functions are obsolete and kept for backward compatibility only. These functions call the init function above:

f\_EPTF\_int2int\_HashMap\_Init(),

f\_EPTF\_str2int\_HashMap\_Init(),

f\_EPTF\_oct2int\_HashMap\_Init()

### Cleanup

The cleanup of the HashMap database is managed by the Base feature. It is not necessary to call the cleanup function of the HashMap feature explicitly. Improper cleanup should not cause any problems in the HashMap feature. After the next initialization everything should behave smoothly.

### Creating a new HashMap

To create a new HashMap the following functions should be called:

f\_EPTF\_int2int\_HashMap\_New(pl\_name), f\_EPTF\_str2int\_HashMap\_New(pl\_name), f\_EPTF\_oct2int\_HashMap\_New(pl\_name)

This function creates a HashMap called with the argument *pl\_name*. The names of the HashMap and a generated ID will be stored in another HashMap called *hashnames*. In return the function gives this ID, which will be used in HashMap element handling for specifying the HashMap. The name of the HashMap cannot be an empty value.

### Deleting the HashMap

For deleting the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Delete(pl\_name), f\_EPTF\_str2int\_HashMap\_Delete(pl\_name), f\_EPTF\_oct2int\_HashMap\_Delete(pl\_name)

This function deletes a HashMap called with the argument *pl\_name*. The name and the ID of the HashMap will be deleted from the names HashMap too.

### Deleting the HashMap by ID

For deleting the HashMap by its ID, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_DeleteById(pl\_id), f\_EPTF\_str2int\_HashMap\_DeleteById(pl\_id), f\_EPTF\_oct2int\_HashMap\_DeleteById(pl\_id)

This function deletes a HashMap with the ID given in the argument *pl\_id*. The name and the ID of the HashMap will be deleted from the names HashMap too.

### Getting the ID of HashMap

For getting the ID of the HashMap the function

f\_EPTF\_int2int\_HashMap\_GetID(pl\_name),

f\_EPTF\_str2int\_HashMap\_ GetID(pl\_name),

f\_EPTF\_oct2int\_HashMap\_ GetID(pl\_name)

has to be called. This function returns the ID assigned to HashMap argument *pl\_name.*

### Assignment of a HashMap

To assign one HashMap to another the function

f\_EPTF\_int2int\_HashMap\_Assign(pl\_id1, pl\_id2), f\_EPTF\_str2int\_HashMap\_ Assign(pl\_id1, pl\_id2), f\_EPTF\_oct2int\_HashMap\_ Assign(pl\_id1, pl\_id2)

has to be called. This function assigns the HashMap with id pl\_id2 to the HashMap with id pl\_id1. In TTCN notation: HashMap1:=HashMap2 happens.

### Dumping the content of one HashMap

To dump the content of one HashMap the function

f\_EPTF\_int2int\_HashMap\_Dump(pl\_name), f\_EPTF\_str2int\_HashMap\_Dump(pl\_name), f\_EPTF\_oct2int\_HashMap\_Dump(pl\_name)

has to be called. This function dumps all the key, data pairs of a specified HashMap.

### Dumping the content of one HashMap by ID

To dump the content of one HashMap by ID the function

f\_EPTF\_int2int\_HashMap\_DumpByID(pl\_id), f\_EPTF\_str2int\_HashMap\_DumpByID (pl\_id), f\_EPTF\_oct2int\_HashMap\_DumpByID (pl\_id)

has to be called. This function dumps all the key, data pairs of a specified HashMap by the ID if the HashMap.

### Dumping the content of all HashMap

To dump the content of all HashMap the function

f\_EPTF\_int2int\_HashMap\_DumpAll(), f\_EPTF\_str2int\_HashMap\_DumpAll(), f\_EPTF\_oct2int\_HashMap\_DumpAll()

has to be called. This function dumps all the key, data pairs of a every existing HashMap.

### Inserting an element

To insert a key, data pair to a HashMap

f\_EPTF\_int2int\_HashMap\_Insert(pl\_id, pl\_key, pl\_data), f\_EPTF\_str2int\_HashMap\_Insert(pl\_id, pl\_key, pl\_data), f\_EPTF\_oct2int\_HashMap\_Insert(pl\_id, pl\_key, pl\_data)

has to be called. This function inserts a key and a dependent data to the specified HashMap. The key must be unique.

### Updating an element

To update a key, data pair to a HashMap

f\_EPTF\_int2int\_HashMap\_Update(pl\_id, pl\_key, pl\_data), f\_EPTF\_str2int\_HashMap\_Update(pl\_id, pl\_key, pl\_data), f\_EPTF\_oct2int\_HashMap\_Update(pl\_id, pl\_key, pl\_data)

has to be called. This function checks whether the key exists in the HashMap. If the key not exists, inserts the key, data pair to the HashMap. If the key already exists updates the data belongs to the key.

### Finding an element

To find the data assigned to a key

f\_EPTF\_int2int\_HashMap\_Find(pl\_id, pl\_key, pl\_data), f\_EPTF\_str2int\_HashMap\_Find(pl\_id, pl\_key, pl\_data), f\_EPTF\_oct2int\_HashMap\_Find(pl\_id, pl\_key, pl\_data)

has to be called. This function finds the data with the help of the key in the specified Hashmap. If the data has not founded, the function returns false and the return value of data argument is -1.

### Erasing an element

For erasing an element from the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Erase(pl\_id, pl\_key), f\_EPTF\_str2int\_HashMap\_Erase(pl\_id, pl\_key), f\_EPTF\_oct2int\_HashMap\_Erase(pl\_id, pl\_key)

This function erases the key and the data from the specified HashMap. If the key has not founded, a warning message will be sent.

### Clearing all elements

To clear all elements from the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Clear(pl\_id), f\_EPTF\_str2int\_HashMap\_Clear(pl\_id), f\_EPTF\_oct2int\_HashMap\_Clear(pl\_id)

This function clears all key, data pairs from the specified HashMap. After calling this function the HashMap will be empty.

### Getting the used size of HashMap

To get the used size of the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Size(pl\_id), f\_EPTF\_str2int\_HashMap\_Size(pl\_id), f\_EPTF\_oct2int\_HashMap\_Size(pl\_id)

This function returns the number of key, data pairs in the HashMap.

### Resizing the HashMap

There is no need to use this functionality anymore since the size of the HashMap is handled dynamically. These functions are kept for backward compatibility, they should not be used.

To resize the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Resize(pl\_id,pl\_hashsize), f\_EPTF\_str2int\_HashMap\_Resize(pl\_id,pl\_hashsize), f\_EPTF\_oct2int\_HashMap\_Resize(pl\_id,pl\_hashsize)

This function increases the bucket count of the HashMap to at least of the specified size *pl\_hashsize*.

### Get the size of HashMap

To get the size the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_BucketCount(pl\_id), f\_EPTF\_str2int\_HashMap\_BucketCount(pl\_id), f\_EPTF\_oct2int\_HashMap\_BucketCount(pl\_id)

This function returns the actual bucket count of the HashMap. This size contains the number of used and empty but booked buckets of the HashMap.

### Getting the maximum size of HashMap

To get the maximum size of the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_MaxSize(pl\_id), f\_EPTF\_str2int\_HashMap\_MaxSize(pl\_id), f\_EPTF\_oct2int\_HashMap\_MaxSize(pl\_id)

This function returns the maximum allowed size of the HashMap.

### Check the content of HashMap

To get whether the HashMap is empty, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_Empty(pl\_id), f\_EPTF\_str2int\_HashMap\_Empty(pl\_id), f\_EPTF\_oct2int\_HashMap\_Empty(pl\_id)

This function returns a boolean value whether the specified HashMap is empty. If the HashMap is empty the return value will be true.

### Get the hash value of a key

To use the hash function of the HashMap, the following functions should be called:

f\_EPTF\_int2int\_HashMap\_CallHashFunc(pl\_id, pl\_key), f\_EPTF\_str2int\_HashMap\_CallHashFunc(pl\_id, pl\_key), f\_EPTF\_oct2int\_HashMap\_CallHashFunc(pl\_id, pl\_key)

This function calls the actual hashing function of the specified HashMap. With the help of this function it returns the hash value of an arbitrary key.

### To get the beginning element

To determine the beginning element of a HashMap

f\_EPTF\_int2int\_HashMap\_Begin(pl\_id, pl\_ iter), f\_EPTF\_str2int\_HashMap\_Begin(pl\_id, pl\_ iter), f\_EPTF\_oct2int\_HashMap\_Begin(pl\_id, pl\_ iter)

has to be called. This function determines the first element of the specified HashMap. It returns a Boolean value whether the first element is found. If no element in HashMap it returns, false. The parameter *pl\_ iter* countains the returned key of the first element.

### To get the next element

To determine the next element of a HashMap

f\_EPTF\_int2int\_HashMap\_Next(pl\_id, pl\_iter), f\_EPTF\_str2int\_HashMap\_Next(pl\_id, pl\_iter), f\_EPTF\_oct2int\_HashMap\_Next(pl\_id, pl\_iter)

has to be called. This function determines the next element of the specified HashMap pointed by *pl\_iter* value. Incoming value and next value is *pl\_iter*. It returns a boolean, whether the next element is found.

## Summary Table of all public functions for EPTF HashMap

Table 1. Summary of str2int HashMap functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_HashMap\_init\_CT | Initializes the hash table. |

Table 2. Summary of str2int HashMap functions

|  |  |
| --- | --- |
| Function name | Description |
| ASSERT\_VALID\_ID | Assert macro, for checking whether the HashMap ID is correct. |
| f\_EPTF\_str2int\_HashMap\_Init | Initializes the hash table. |
| f\_EPTF\_str2int\_HashMap\_Cleanup | Deletes all hashmaps. |
| f\_EPTF\_str2int\_HashMap\_New | Creates a new hashmap. |
| f\_EPTF\_str2int\_HashMap\_GetID | To get the ID of the hashmap from the name. |
| f\_EPTF\_str2int\_HashMap\_Delete | Deletes the hashmap. |
| f\_EPTF\_str2int\_HashMap\_DeleteById | Deletes the hashmap by ID. |
| f\_EPTF\_str2int\_HashMap\_DumpByID | Dump the content of a specific hashmap by ID. |
| f\_EPTF\_str2int\_HashMap\_Dump | Dump the content of a specific hashmap by name. |
| f\_EPTF\_str2int\_HashMap\_DumpAll | Dump the content of all hashmaps. |
| f\_EPTF\_str2int\_HashMap\_Assign | Assignment of one hashmap to another. |
| f\_EPTF\_str2int\_HashMap\_Insert | Inserts an element to hashmap. |
| f\_EPTF\_str2int\_HashMap\_Update | Updatess an element in hashmap. |
| f\_EPTF\_str2int\_HashMap\_Find | Finds an element in hashmap. |
| f\_EPTF\_str2int\_HashMap\_Erase | Erases an element from hashmap. |
| f\_EPTF\_str2int\_HashMap\_Clear | Clear all elements from hashmap. |
| f\_EPTF\_str2int\_HashMap\_Size | Returns the size of hashmap. |
| f\_EPTF\_str2int\_HashMap\_MaxSize | Returns the maximum size of hashmap. |
| f\_EPTF\_str2int\_HashMap\_CallHashFunc | Calls the hash function of a given hashmap. |
| f\_EPTF\_str2int\_HashMap\_Empty | Check whether the hashmap is empty. |
| f\_EPTF\_str2int\_HashMap\_Resize | Increases the bucket count to at least pl\_hashsize. Deprecated, should not be used anymore. |
| f\_EPTF\_str2int\_HashMap\_BucketCount | Returns the number of buckets used by the hash\_map. |
| f\_EPTF\_str2int\_HashMap\_Begin | Function call to get the first element of hashmap. |
| f\_EPTF\_str2int\_HashMap\_Next | Function call to get the next element of hashmap. |

Table 3. Summary of oct2int HashMap functions

|  |  |
| --- | --- |
| Function name | Description |
| ASSERT\_VALID\_ID | Assert macro, for checking whether the HashMap ID is correct. |
| f\_EPTF\_oct2int\_HashMap\_Init | Initializes the hash table. |
| f\_EPTF\_oct2int\_HashMap\_Cleanup | Deletes all hashmaps. |
| f\_EPTF\_oct2int\_HashMap\_New | Creates a new hashmap. |
| f\_EPTF\_oct2int\_HashMap\_GetID | To get the ID of the hashmap from the name. |
| f\_EPTF\_oct2int\_HashMap\_Delete | Deletes the hashmap. |
| f\_EPTF\_oct2int\_HashMap\_DeleteById | Deletes the hashmap by ID. |
| f\_EPTF\_oct2int\_HashMap\_DumpByID | Dump the content of a specific hashmap by ID. |
| f\_EPTF\_oct2int\_HashMap\_Dump | Dump the content of a specific hashmap by name. |
| f\_EPTF\_oct2int\_HashMap\_DumpAll | Dump the content of all hashmaps. |
| f\_EPTF\_oct2int\_HashMap\_Assign | Assignment of one hashmap to another. |
| f\_EPTF\_oct2int\_HashMap\_Insert\_ext | Inserts an element to hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Find\_ext | Finds an element in hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Erase\_ext | Erases an element from hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Clear | Clear all elements from hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Size | Returns the size of hashmap. |
| f\_EPTF\_oct2int\_HashMap\_MaxSize | Returns the maximum size of hashmap. |
| f\_EPTF\_oct2int\_HashMap\_CallHashFunc\_ext | Calls the hash function of a given hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Empty | Check whether the hashmap is empty. |
| f\_EPTF\_oct2int\_HashMap\_Resize | Increases the bucket count to at least pl\_hashsize. Deprecated, should not be used anymore. |
| f\_EPTF\_oct2int\_HashMap\_BucketCount | Returns the number of buckets used by the hash\_map. |
| f\_EPTF\_oct2int\_HashMap\_Begin\_ext | Function call to get the first element of hashmap. |
| f\_EPTF\_oct2int\_HashMap\_Next\_ext | Function call to get the next element of hashmap. |

Table 4. Summary of int2int HashMap functions

|  |  |
| --- | --- |
| Function name | Description |
| ASSERT\_VALID\_ID | Assert macro, for checking whether the HashMap ID is correct. |
| f\_EPTF\_int2int\_HashMap\_Init | Initializes the hash table. |
| f\_EPTF\_int2int\_HashMap\_Cleanup | Deletes all hashmaps. |
| f\_EPTF\_int2int\_HashMap\_New | Creates a new hashmap. |
| f\_EPTF\_int2int\_HashMap\_GetID | To get the ID of the hashmap from the name. |
| f\_EPTF\_int2int\_HashMap\_Delete | Deletes the hashmap. |
| f\_EPTF\_int2int\_HashMap\_DeleteById | Deletes the hashmap by ID. |
| f\_EPTF\_int2int\_HashMap\_DumpByID | Dump the content of a specific hashmap by ID. |
| f\_EPTF\_int2int\_HashMap\_Dump | Dump the content of a specific hashmap by name. |
| f\_EPTF\_int2int\_HashMap\_DumpAll | Dump the content of all hashmaps. |
| f\_EPTF\_int2int\_HashMap\_Assign | Assignment of one hashmap to another. |
| f\_EPTF\_int2int\_HashMap\_Insert | Inserts an element to hashmap. |
| f\_EPTF\_int2int\_HashMap\_Find | Finds an element in hashmap. |
| f\_EPTF\_int2int\_HashMap\_Erase | Erases an element from hashmap. |
| f\_EPTF\_int2int\_HashMap\_Clear | Clear all elements from hashmap. |
| f\_EPTF\_int2int\_HashMap\_Size | Returns the size of hashmap. |
| f\_EPTF\_int2int\_HashMap\_MaxSize | Returns the maximum size of hashmap. |
| f\_EPTF\_int2int\_HashMap\_CallHashFunc | Calls the hash function of a given hashmap. |
| f\_EPTF\_int2int\_HashMap\_Empty | Check whether the hashmap is empty. |
| f\_EPTF\_int2int\_HashMap\_Resize | Increases the bucket count to at least pl\_hashsize. Deprecated, should not be used anymore. |
| f\_EPTF\_int2int\_HashMap\_BucketCount | Returns the number of buckets used by the hash\_map. |
| f\_EPTF\_int2int\_HashMap\_Begin | Function call to get the first element of hashmap. |
| f\_EPTF\_int2int\_HashMap\_Next | Function call to get the next element of hashmap. |

# References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. [Part 1: Core Language](http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.02.01_60/es_20187301v030201p.pdf)
2. 1/198 17-CRL 113 200/5 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 155 17-CNL 113 512 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
4. EPTF Core Library for TTCN-3 toolset with TITAN, [Reference Guide](http://ttcn.ericsson.se/TCC_Releases/Libraries/EPTF_Core_Library_CNL113512/doc/apidoc/html/index.html)