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

EPTF CLL HashMap, User Guide

Contents

[1 Introduction 2](#_Toc310943598)

[1.1 Revision history 2](#_Toc310943599)

[1.2 About this Document 2](#_Toc310943600)

[1.2.1 How to Read this Document 2](#_Toc310943601)

[1.2.2 References 2](#_Toc310943602)

[1.2.3 Abbreviations 3](#_Toc310943603)

[1.2.4 Terminology 3](#_Toc310943604)

[1.3 System Requirements 3](#_Toc310943605)

[2 EPTF CLL HashMap 3](#_Toc310943606)

[2.1 Overview 3](#_Toc310943607)

[2.2 Description of files in this feature 3](#_Toc310943608)

[2.3 Description of required files from other features 4](#_Toc310943609)

[2.4 Installation 5](#_Toc310943610)

[2.5 Configuration 5](#_Toc310943611)

[3 Error messages 5](#_Toc310943612)

[4 Warning messages 6](#_Toc310943613)

[5 Examples 6](#_Toc310943614)

[5.1 Configuration file 6](#_Toc310943615)

[5.2 Demo Module 6](#_Toc310943616)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-16 | PA1 | First draft version | EBENMOL |
| 2011-10-14 | PB1 | HashMap component added | ETHJGI |
|  |  |  |  |
|  |  |  |  |

## About this Document

### How to Read this Document

This is the User Guide for the HashMap feature of the Ericsson Performance Test Framework (TitanSim), Core Load Library (CLL). TitanSim CLL is developed for the TTCN-3 ‎[1] Toolset with TITAN ‎[2]. This document should be read together with the Function Description of the HashMap feature ‎[6]. For more information on the TitanSim CLL please consult the Product Revision Information ‎[3], the Users Guide ‎[4] and the Function Specification ‎[5] of the TitanSim.

### 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. 198 17-CNL 113 512 Uen  
   TitanSim CLL for TTCN-3 toolset with TITAN, User Guide
6. 1/155 16-CNL 113 512  
   EPTF CLL HashMap Function Description
7. TitanSim CLL for TTCN-3 toolset with TITAN, Reference Guide  
   <http://ttcn.ericsson.se/products/libraries.shtml>

### 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

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

## System Requirements

In order to use the HashMap feature the system requirements listed in TitanSim CLL User Guide ‎[5] should be fulfilled.

# EPTF CLL HashMap

## Overview

The EPTF CLL HashMap component is a fundamental component that provides acces 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. 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.

## Description of files in this feature

The EPTF CLL HashMap API includes the following files:

* common files: Common functions and type definitions for all type of hashmaps:
  + EPTF\_CLL\_HashMap\_Definitions.ttcn: Defines the HashMap component
  + EPTF\_CLL\_HashMap\_Functions.ttcn: Defines general functions like init for all HashMaps.
  + EPTF\_CLL\_HashMap\_ExternalFunctions.hh: Template functions for all types of HashMap.
* str2int files: The functions using hashmaps containing <charstring, integer> <key, data> pairs
  + EPTF\_CLL\_HashMapStr2Int\_Functions.ttcn: This TTCN-3 module contains function definitions to GCC HashMap functions
  + EPTF\_CLL\_HashMapStr2Int\_ExternalFunctions.cc: This c++ source file contains the implementations of functions to call standard GCC HashMap handling and maintenance functions.
* oct2int files: The functions using hashmaps containing <octetstring, integer> <key, data> pairs
  + EPTF\_CLL\_HashMapOct2Int\_Functions.ttcn: This TTCN-3 module contains function definitions to GCC HashMap functions
  + EPTF\_CLL\_HashMapOct2Int\_ExternalFunctions.cc: This c++ source file contains the implementations of functions to call standard GCC HashMap handling and maintenance functions.
* int2int files: The functions using hashmaps containing <integer, integer> <key, data> pairs
  + EPTF\_CLL\_HashMapInt2Int\_Functions.ttcn: This TTCN-3 module contains function definitions to GCC HashMap functions
  + EPTF\_CLL\_HashMapInt2Int\_ExternalFunctions.cc: This c++ source file contains the implementations of functions to call standard GCC HashMap handling and maintenance functions.

## Description of required files from other features

The HashMap feature is part of the TitanSim EPTF Core Load Library (CLL).

It requires files from these CLL features:

* EPTF\_CLL\_Base

## Installation

Since EPTF\_CLL HashMap is used as a part of the TTCN-3 test environment this requires TTCN-3 Test Executor to be installed before any operation of these functions. For more details on the installation of TTCN-3 Test Executor see the relevant section of ‎[2].

If not otherwise noted in the respective sections, the following are needed to use EPTF\_CLL\_HashMap:

* Copy the files listed in section [2.2, ‎2.3] to the directory of the test suite or create symbolic links to them.
* Import the HashMap demo or write your own application using HashMap.
* Create Makefile or modify the existing one. For more details see the relevant section of ‎[2].
* Edit the config file according to your needs, see following section [‎2.5].

## Configuration

The executable test program behaviour is determined via the run-time configuration file. This is a simple text file, which contains various sections. The usual suffix of configuration files is .cfg. For further information on the configuration file see ‎[2].

This feature defines TTCN-3 module parameters as defined in ‎[2] clause 4. Actual values of these parameters – when no default value or a different from the default actual value wished to be used – shall be given in the [MODULE\_PARAMETERS] section of the configuration file.

This feature defines no module parameters.

# Error messages

Please note, that besides the below described error messages, error messages shown in ‎[2] or those of other used features or product may also appear.

No HashMap-specific error messages are defined.

# Warning messages

Please note, that besides the below described warning messages, warning messages shown in ‎[2] or those of other used features or product may also appear.

**Can't create new HashMap. HashMap name is not unique**

HashMap names must be unique. Creating a HashMap with the name already assigned to another HashMap will cause this warning. New HashMap won’t be created.

**Can't get ID. No HashMap found with this name**

To get the ID of a wrong HashMap name will cause this warning.

**Can't get ID. HashMap not deleted. No HashMap found with this name**

Trying to delete an already deleted HashMap will cause this warning.

**Can't delete hasmap: No HashMap found with this id**

Trying to delete an already deleted HashMap by ID will cause this warning.

# Examples

The “demo” directory of the deliverable contains the following examples:

* HashMap.cfg
* HashMap\_demo.ttcn

## Configuration file

The used configuration file (HashMap.cfg) is for the HashMap example is placed in the demo directory.

## Demo Module

The demo module (HashMap\_demo.ttcn) illustrates a typical usage of the HashMap feature.

Example 1 shows how to initialize, create, delete and handle a specific HashMap.

Module HashMap\_str2int\_demo {

import from EPTF\_CLL\_str2int\_HashMap all;

testcase tc\_HashMap()

runs on MyMTC\_CT

{

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

var charstring v\_first := "my\_first\_hashmap";

var integer v\_first\_int;

v\_first\_int := f\_EPTF\_str2int\_HashMap\_New (v\_first);

var integer v\_first\_check;

var boolean is\_first;

is\_first := f\_EPTF\_str2int\_HashMap\_GetID(v\_first,v\_first\_check);

log("The ID of ",v\_first,"is ",v\_first\_check);

f\_EPTF\_str2int\_HashMap\_Delete (v\_first);

}

control

{

execute( tc\_HashMap() );

}

}

Example 2 shows how to insert, find elements of HashMap.

Module HashMap\_str2int\_demo {

import from EPTF\_CLL\_str2int\_HashMap all;

testcase tc\_HashMap()

runs on MyMTC\_CT

{

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

var charstring v\_first := "my\_first\_hashmap";

var integer v\_first\_int;

v\_first\_int := f\_EPTF\_str2int\_HashMap\_New (v\_first);

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"hello", 11 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"world", 22 );

var integer v\_hm\_data;

if ( f\_EPTF\_str2int\_HashMap\_Find (v\_first\_int,"hello", v\_hm\_data ) )

{

log("Reading data from hashmap first, key = hello data: \n", v\_hm\_data );

}

f\_EPTF\_str2int\_HashMap\_Erase (v\_first\_int, "hello" );

f\_EPTF\_str2int\_HashMap\_Delete (v\_first);

}

control

{

execute( tc\_HashMap() );

}

}

Example 3 shows how to get and set the size of HashMaps.

Module HashMap\_str2int\_demo {

import from EPTF\_CLL\_str2int\_HashMap all;

testcase tc\_HashMap()

runs on MyMTC\_CT

{

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

var charstring v\_first := "my\_first\_hashmap";

var integer v\_first\_int;

v\_first\_int := f\_EPTF\_str2int\_HashMap\_New (v\_first);

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"hello", 11 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"world", 22 );

var integer v\_hm\_size;

v\_hm\_size := f\_EPTF\_str2int\_HashMap\_Size (v\_first\_int);

log( "Number of elements in hashmap: \n", v\_hm\_size );

var float v\_max\_size;

v\_max\_size := f\_EPTF\_str2int\_HashMap\_MaxSize (v\_first\_int);

log( "Maximum size of hashmap: \n", v\_max\_size );

log( "Resizing the hashmap to countain at least 500 empty buckets”);

f\_EPTF\_str2int\_HashMap\_Resize (v\_first\_int,500);

f\_EPTF\_str2int\_HashMap\_Delete (v\_first);

}

control

{

execute( tc\_HashMap() );

}

}

Example 4 shows a while cycle with begin and next functions.

Module HashMap\_str2int\_demo {

import from EPTF\_CLL\_str2int\_HashMap all;

testcase tc\_HashMap()

runs on MyMTC\_CT

{

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

var charstring v\_first := "my\_first\_hashmap";

var integer v\_first\_int;

v\_first\_int := f\_EPTF\_str2int\_HashMap\_New (v\_first);

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"hello", 11 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"world", 22 );

var charstring v\_readkey:="";

var boolean v\_cycleIsOver:= f\_EPTF\_str2int\_HashMap\_Begin(v\_first\_int, v\_readkey);

if( not v\_cycleIsOver )

{

log("No elements in hashmap" );

};

while (v\_cycleIsOver)

{

var integer v\_hm\_data;

if ( f\_EPTF\_str2int\_HashMap\_Find (v\_first\_int, v\_readkey, v\_hm\_data ) )

{

log("key,data pairs: \n", v\_readkey," ", v\_hm\_data );

}

v\_cycleIsOver := f\_EPTF\_str2int\_HashMap\_Next(v\_first\_int, v\_readkey);

}

f\_EPTF\_str2int\_HashMap\_Delete (v\_first);

}

control

{

execute( tc\_HashMap() );

}

}

Example 5 shows the dumping HashMap functions.

Module HashMap\_str2int\_demo {

import from EPTF\_CLL\_str2int\_HashMap all;

testcase tc\_HashMap()

runs on MyMTC\_CT

{

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

var charstring v\_first := "my\_first\_hashmap";

var integer v\_first\_int;

v\_first\_int := f\_EPTF\_str2int\_HashMap\_New (v\_first);

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

var charstring v\_second := "my\_second\_hashmap";

var integer v\_second\_int;

v\_second\_int := f\_EPTF\_str2int\_HashMap\_New (v\_second);

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"hello", 11 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_first\_int,"world", 22 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_second\_int,"good", 88 );

f\_EPTF\_str2int\_HashMap\_Insert ( v\_second\_int,"day", 99 );

log("Dumping first hashmap by name");

f\_EPTF\_str2int\_HashMap\_Dump(v\_first);

log("Dumping first hashmap by ID");

f\_EPTF\_str2int\_HashMap\_DumpByID(v\_first\_int);

log("Dumping all hashmaps");

f\_EPTF\_str2int\_HashMap\_DumpAll();

f\_EPTF\_str2int\_HashMap\_Delete (v\_first);

f\_EPTF\_str2int\_HashMap\_Delete (v\_second);

}

control

{

execute( tc\_HashMap() );

}

}