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

EPTF CLL Base, Function Description

Contents

[Introduction 3](#_Toc468445037)

[1.1 Revision history 3](#_Toc468445038)

[1.2 How to Read this Document 3](#_Toc468445039)

[1.3 Scope 3](#_Toc468445040)

[1.4 Recommended way of reading 3](#_Toc468445041)

[1.5 Typographical conventions 4](#_Toc468445042)

[1.6 Abbreviations 4](#_Toc468445043)

[1.7 Terminology 4](#_Toc468445044)

[2 General Description 4](#_Toc468445045)

[3 Functional Interface 5](#_Toc468445046)

[3.1 Naming Conventions 5](#_Toc468445047)

[3.2 Public Functions 5](#_Toc468445048)

[3.2.1 Initialization 5](#_Toc468445049)

[3.2.2 Registering the cleanup function 5](#_Toc468445050)

[3.2.3 Cleanup function 5](#_Toc468445051)

[3.2.4 Forcing stop 6](#_Toc468445052)

[3.2.5 Stopping a Remote Component 6](#_Toc468445053)

[3.2.6 Stopping all Components 6](#_Toc468445054)

[3.2.7 Getting the EPTF Self Name 6](#_Toc468445055)

[3.2.8 Main Event Loop 7](#_Toc468445056)

[3.2.9 Waiting for a Condition 7](#_Toc468445057)

[3.2.10 Getting the Relative Time 7](#_Toc468445058)

[3.2.11 Getting Relative Time Offset 7](#_Toc468445059)

[3.2.12 Getting the Absolute Time 7](#_Toc468445060)

[3.2.13 Assert 8](#_Toc468445061)

[3.2.14 Set an assert message pattern to expect as the first error occurred 8](#_Toc468445062)

[3.2.15 Determine the number of asserts 8](#_Toc468445063)

[3.2.16 To retrieve a certain assert message 8](#_Toc468445064)

[3.2.17 To check if an assert message matches with a given pattern 9](#_Toc468445065)

[3.2.18 Negative test mode 9](#_Toc468445066)

[3.2.19 Getting the process ID of the current PTC 9](#_Toc468445067)

[3.2.20 Getting the hostname of the current PTC 9](#_Toc468445068)

[3.2.21 To retrieve the process ID of other components 9](#_Toc468445069)

[3.2.22 To get detailed host information of a PTC by selfName 10](#_Toc468445070)

[3.2.23 To get the component information for all PTCs 10](#_Toc468445071)

[3.2.24 Enabling/Disabling DTE handling in CLL 10](#_Toc468445072)

[3.2.25 Get the current status of DTE handling 11](#_Toc468445073)

[3.2.26 Executing shell commands 11](#_Toc468445074)

[3.2.27 Getting the start command of the application 11](#_Toc468445075)

[3.2.28 Restart the application 12](#_Toc468445076)

[3.3 Summary Table of all public functions for EPTF Base 12](#_Toc468445077)

[4 References 14](#_Toc468445078)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-12-03 | PA1 | First draft version | EGBOTAT |
| 2007-12-06 | PA2 | Final version after review | EGBOTAT |
| 2008-11-21 | PB1 | New revision | EGBOTAT |
| 2009-11-04 | PC1 | EPTF\_DEBUG switch | EANDRKS |
| 2009-12-11 | PC2 | Negative test support | ETHJGI |
| 2011-09-16 | PD1 | Component Info | ETHJGI |
| 2012-10-08 | PE1 | WaitForCondition function added | ETHJGI |
| 2014-11-24 | PF1 | New chapters added | ESZILSZ |
| 2014-11-24 | PF2 | References updated | ESZILSZ |
| 2014-12-02 | F | Updated for release | ESZILSZ |
| 2015-06-29 | PG1 | 3.2.24 chapter updated | ESZILSZ |
| 2015-10-09 | PG2 | Subchapters 3.2.26-28 were added | EZOLZSI |
| 2015-12-03 | G | Updated for release | ESZILSZ |
| 2016-10-14 | PH1 | 3.2.26, 3.3 chapters updated | EVAURBA |
| 2016-10-15 | PH2 | Revision history updated | EVAURBA |
| 2016-10-17 | PH3 | Editorial changes | EIMRENA |
| 2016-10-17 | PH4 | Fixed revision | EIMRENA |
| 2016-10-17 | PH5 | Editorial changes | EIMRENA |
| 2016-12-02 | H | Updated for release | EIMRENA |

## How to Read this Document

This is the Function Description for the Base feature of the Ericsson Performance Test Framework (TitanSim), Core Library (CLL). Core Library is developed for the TTCN-3 [1] Toolset with TITAN [2]. For more information on the Core Library please consult the Product Revision Information.

## Scope

This document is to specify the content and functionality of the Base feature of the Core Library.

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of Core Library [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.

## Abbreviations

CLL Core Library

EPTF Ericsson Performance Test Framework

MTC Main Test Component

PTC Parallel Test Component

TitanSim New synonym for the EPTF 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 Core Library is to be supplied and supported by the TCC organization. Any Core Library development is to be funded centrally by Ericsson.

# General Description

This document specifies the Base feature of the Core library.

The EPTF Base component is a fundamental component that handles common tasks that should be implemented in all EPTF components. All EPTF components should extend this component (i.e. explicitly) or a component that extends this component (i.e. implicitly).

This component

* stores the name of the component
* manages the clean up so that the component could terminate gracefully. This solution is similar to the C++ “destructor” concept.
* provides relative and absolute time querying functions
* provides functions to convert the component type to/from integer
* makes negative testing possible with error and assert functions
* provides process information on every PTC running on components extending the EPTF Base component
* enables/disables DTE handling functionality in the whole CLL

To be able to use EPTF Base, the user component should extend the EPTF\_Base\_CT component and call its initializing function. When the component terminates it should call one of the cleanup/stop/stopAll functions of the Base component.

# Functional Interface

Apart from this description a cross-linked reference guide for the Core Library Functions can be reached for on-line reading [4].

## Naming Conventions

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

## Public Functions

### Initialization

Before using the EPTF Base functions the

*f\_EPTF\_Base\_init\_CT(pl\_selfName)*

function should be called. This initializes the EPTF Base feature. The EPTF self name of the component shall be passed as parameter (pl\_selfName).

### Registering the cleanup function

In a component that extends EPTF\_Base\_CT (explicitly or implicitly) the function

*f\_EPTF\_Base\_RegisterCleanup(cleanup function reference)*

should be called with the cleanup function of the given component as its argument. This function adds the given cleanup function to the list of cleanup functions. The registered cleanup function will be called in a reversed order before terminating the component.

### Cleanup function

Before terminating a component execution that extends EPTF\_Base\_CT (explicitly or implicitly) the function

*f\_EPTF\_Base\_cleanup\_CT ()*

has to be called.

This function calls the cleanup function of the given component and all its parents (if they were registered). After the function returns, the component can be terminated.

(This function is registered automatically in f\_EPTF\_Base\_init\_CT() as the cleanup function of the EPTF\_Base\_CT.)

### Forcing stop

To force to stop a component execution because of an error, the function

*f\_EPTF\_Base\_stop()*

has to be called.

It calls the f\_EPTF\_Base\_cleanup\_CT function and stops the execution automatically. It is possible to set the verdict of the test case with this function. By default the verdict is set to ‘fail’. Setting the verdict to ‘none’, the verdict of the test case will not change.

### Stopping a Remote Component

To function

*f\_EPTF\_Base\_stopRemote(component reference)*

can be used to stop a remote component as if its cleanup function was called.

### Stopping all Components

The function

*f\_EPTF\_Base\_stopAll()*

can be used to stop all components. This function sets the verdict to ‘fail’ by default and stops the MTC.

### Getting the EPTF Self Name

The function

*f\_EPTF\_Base\_selfName()*

can be used to retrieve the EPTF self name of the component.

### Main Event Loop

The function

*f\_EPTF\_Base\_wait4Shutdown()*

can be used to wait until the test stops. This function shall be the main event loop of each component instance using EPTF Base. It consists of an alt statement that allows all EPTF default altsteps to run.

### Waiting for a Condition

The function

f\_EPTF\_Base\_waitForCondition(pl\_checkCondFn, pl\_maxWaitTime)

can be used to wait until the given callback function returns true. A maximal waiting time can be specified if needed, after which the function exits even if the callback function returns false. This can be used to prevent blocking forever. If the waitForCondition function returns false, the condition was not fulfilled.

By default the maximal waiting time is set to infinity. In this case the execution is blocked until the specified call-back function returns true.

If the callback function is not given, the function blocks execution until the maximal waiting time expires.

### Getting the Relative Time

The function

f\_EPTF\_Base\_getRelTimeInSecs()

can be used to retrieve the relative time elapsed since the component was initialized.

### Getting Relative Time Offset

The function

*f\_EPTF\_Base\_getRelTimeOffsetInSecs()*

can be used to get the time-of-day at component initialization as a float value.

### Getting the Absolute Time

The function

*f\_EPTF\_Base\_getAbsTimeInSecs()*

can be used to get the absolute time, similar to the C function gettimeofday as a float value. The value returned is the relative time plus the relative time offset, which is supposed to be faster than a gettimeofday function call (i.e. getting the system time since 00:00 UCT, Jan 1, 1970).

### Assert

The function

*f\_EPTF\_Base\_assert(pl\_assertMessage, pl\_predicate)*

can be used to put assertions in code. The function itself is implemented as an external C++ function, which makes it possible to exclude it from the final executable if the preprocessor flag –DEPTF\_DEBUG is not added to CPPFLAGS and –O2 to CXXFLAGS in the Makefile.

The function logs *pl\_assertMessage* as ERROR\_UNQUALIFIED and stops the component by calling *f\_EPTF\_Base\_stop()* if compiled into the code and if *pl\_predicate* evaluates to false.

### Set an assert message pattern to expect as the first error occurred

The function

f\_EPTF\_Base\_setExpectedAssertMsg(pl\_expectedAssert)

can be used to set an assert pattern that will be check during cleanup automatically. If the first assert message was the one that matches to this pattern, the verdict of the test case is set to pass automatically. If it does not match, then the verdict of the test case is set to fail.

An example for the expected assert pattern that checks the end of the assert message can be:

“\*Some assert occurred”

This function can be used in negative testing.

### Determine the number of asserts

The number of asserts occurred is returned by the function

f\_EPTF\_ Base\_nofAssertMsgs()

### To retrieve a certain assert message

A given assert message text can be retrieved by the function

f\_EPTF\_ Base\_getAssertMsg(pl\_assertNum)

where pl\_assertNum gives the id of the assert message. The id if the assert message starts from zero, i.e. it is zero for the first assert, one for the second an so on.

### To check if an assert message matches with a given pattern

To check if an assert message matches with a given pattern, the function

f\_EPTF\_Base\_checkExpectedAssert(pl\_expectedAssert, pl\_assertNum)

can be used. The pl\_expectedAssert is the assert pattern that is matched with the occurred assert with id pl\_assertNum. The id of the asserts starts from zero. The function returns true if the pattern matches and false if it does not.

### Negative test mode

Negative test mode can be set by the function

f\_EPTF\_Base\_setNegativeTestMode()

This function disables the setverdict in the functions: f\_EPTF\_Base\_stop, f\_EPTF\_Base\_stopAll and f\_EPTF\_Base\_assert, so that the verdict can be set to pass if an error or assert happened.

### Getting the process ID of the current PTC

The process ID of the current PTC can be retrieved by the function

f\_EPTF\_Base\_getPid()

This function returns the process ID of the current PTC that can also be shown e.g. by the ‘top’ unix command.

### Getting the hostname of the current PTC

The hostname of the current PTC can be retrieved by the function

f\_EPTF\_Base\_getHostName()

This function returns the name of the host where the current PTC is running. This name is what the ‘hostname’ unix command returns.

### To retrieve the process ID of other components

The process ID of any test component (PTC) that executes a function that runs on a component that extends the Base component can be retrieved by the following function on all components:

f\_EPTF\_Base\_getPidByName (pl\_name)

The pl\_name parameter is the name passed to the init function of the Base feature on some PTC. The returned integer is the process ID of the PTC that called the Base init function with the specified name. This is the name that the f\_EPTF\_Base\_selfName function returns on that PTC.

If more than one PTC exists with the given name, the PID for the one that is found first is returned.

### To get detailed host information of a PTC by selfName

The host information, that includes the name of the PTC, the TTCN-3 component reference, the hostname and the PID of the process on that host, can be retrieved of any PTC that called the Base init function. The following function can be used for that purpose:

f\_EPTF\_Base\_getComponentInfoByName(pl\_name, pl\_compInfo)

The function returns the component information in its pl\_compInfo argument for the component that initialized the Base feature with the specified pl\_name argument. If the component information is not available, the function returns a nonzero integer, otherwise the function returns zero. If more than one PTC exists with the given name, the component info for the one that is found first is returned.

### To get the component information for all PTCs

The list of component information for all PTCs can be retrieved by the function

f\_EPTF\_Base\_getComponentInfoAll()

It returns a list of component information (selfName, component reference, hostname, process ID) for all PTCs that run on a component that extend the Base component and called its init function.

### Enabling/Disabling DTE handling in CLL

It is possible to enable/disable the DTE handling functionality in the whole CLL by calling the function:

f\_EPTF\_Base\_setDTEHandling(pl\_dteHandling)

When DTE handling is enabled, other CLL features that support DTE handling (e.g. LGenBase, Scheduler, Transport) will not stop when DTE occurs in user-callback functions.

Instead, they will print the error message as a warning, call a user-given handler function and continue the execution.

By default DTE handling is switched on.

**Note:** Titan versions before TITAN/4/R2A, including this version, will not print the call stack of the actual place of the error. Newer Titan version will support this.

### Get the current status of DTE handling

If DTE handling is enabled or not, can be determined by calling the function:

f\_EPTF\_Base\_isEnabledDTEHandling()

This function returns true if DTE handling is enabled in the CLL, false if not.

By default DTE handling is disabled.

### Executing shell commands

It is possible to execute shell commands with the following function:

f\_EPTF\_Base\_executeShell(

in charstring pl\_command,

out charstring pl\_stdOut,

out charstring pl\_stdErr,

in boolean pl\_enableAlts := true)

This function returns the error code of the execution and stores the standard output / error of the execution in pl\_stdOut / pl\_stdErr .

When the argument pl\_enableAlts is true, the shell command execution does not block the default altsteps. If it is false, the default altsteps are not running, so they are blocked until the function returns. By default this argument is true.

If the stdout/stderr is not relevant the following function can be used:

public external function f\_EPTF\_Base\_system(in charstring pl\_command) return integer;

This function calls the system c-function. It returns the error code.

### Getting the start command of the application

It is possible to get the start command of the current application with the following function:

f\_EPTF\_Base\_getStartCommand(

out charstring pl\_executable,

out charstring pl\_configFile,

out EPTF\_CharstringList pl\_testCases ()

This function returns the start command.

Example:

“/app/TITAN/5\_R3A/LMWP3.1/bin/ttcn3\_start EPTF\_Base\_Test ../EPTF\_Base\_Test.cfg EPTF\_Base\_Test\_Testcases.tc\_EPTF\_Base\_Test\_getStartCommand”

As you can see, this string consists of the following values separated by spaces: start script of TTCN-3 parallel mode, name of the executable, name of the configuration file and the test case. In addition, the last three values are returned as outgoing parameters as well.

### Restart the application

It is possible to restart the current application with the following function:

f\_EPTF\_Base\_restart(in charstring pl\_executable := "-", in charstring pl\_configFile := "-",in EPTF\_CharstringList pl\_testCase := {"-"})

It will stop the application (all PTCs) and then restart the application in a separate xterm. The following parts of the start command can be manipulated by the arguments.

Example:

“/app/TITAN/5\_R3A/LMWP3.1/bin/ttcn3\_start <pl\_executable> <pl\_configFile> <pl\_testCase>”

All of the arguments have hyphen string (“-”) as default value. It means that the application will be restarted with the original parameters as it was originally in the start command.

## Summary Table of all public functions for EPTF Base

Table 1. Summary of Base functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_Base\_init\_CT | Initializes the Base Component |
| f\_EPTF\_Base\_registerCleanup | Registers the cleanup function |
| f\_EPTF\_Base\_cleanup\_CT | Calls the registered cleanup functions |
| f\_EPTF\_Base\_stop | Sets verdict to ‘fail’, calls the cleanup functions and terminates the component |
| f\_EPTF\_Base\_stopRemote | Stop a remote component |
| f\_EPTF\_Base\_stopAll | Set verdict to ‘fail’ and stop all components |
| f\_EPTF\_Base\_selfName | Retrieve the EPTF self name |
| f\_EPTF\_Base\_wait4Shutdown | Main event loop |
| f\_EPTF\_Base\_waitForCondition | Block execution until a given condition is fulfilled |
| f\_EPTF\_Base\_cleanupIsInProgress | Returns true if cleanup process is started |
| f\_EPTF\_Base\_upcast  f\_EPTF\_Base\_downcast | To convert EPTF\_Base\_CT to/from integer |
| f\_EPTF\_Base\_getRelTimeInSecs | Get time elapsed since component initialization |
| f\_EPTF\_Base\_getRelTimeOffsetInSecs | Get relative time offset |
| f\_EPTF\_Base\_getAbsTimeInSecs | Get the absolute system time |
| f\_EPTF\_Base\_getTimeOfDay | Returns the system time since 00:00 UCT, Jan 1, 1970 |
| f\_EPTF\_Base\_setNegativeTestMode | Set the negative-test mode |
| f\_EPTF\_Base\_getNegativeTestMode | Returns true of negative-test mode is enabled |
| f\_EPTF\_Base\_assert | Print out message in debugging mode if a condition evaluates false. Also stops the component execution. |
| f\_EPTF\_Base\_nofAssertMsgs | Returns the number of assert messages generated by previous f\_EPTF\_Base\_assert function calls |
| f\_EPTF\_Base\_getAssertMsg | Returns an assert message from the previous assert messages |
| f\_EPTF\_Base\_setExpectedAssertMsg  f\_EPTF\_Base\_setExpectedErrorMsg | Sets the assert/error message that is expected as the first assert message in negative-test mode |
| f\_EPTF\_Base\_checkExpectedAssert  f\_EPTF\_Base\_checkExpectedError | Checks if the given assert/error message arrived at correct time |
| f\_EPTF\_Base\_getPid | Returns the PID of the current process |
| f\_EPTF\_Base\_getHostName | Returns the hostname of the current process |
| f\_EPTF\_Base\_getComponentInfoByName | Returns the component information for a component with a given selfName |
| f\_EPTF\_Base\_getPidByName | Returns the PID for a component with a given selfName |
| f\_EPTF\_Base\_getComponentInfoAll | Returns the component info for all components that run on Base |
| f\_EPTF\_Base\_setDTEHandling | Enables/disables DTE handling in CLL |
| f\_EPTF\_Base\_isEnabledDTEHandling | Returns true if DTE handling is enabled |
| f\_EPTF\_Base\_system | Executes system command from TTCN |
| f\_EPTF\_Base\_executeShell | Executes shell command from TTCN |
| f\_EPTF\_Base\_getStartCommand | Returns the command the application was started with |
| f\_EPTF\_Base\_restart | Restarts the application |

# 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)