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

EPTF CLL Improved Log, Function Description

Contents

[1 Introduction 2](#_Toc335038088)

[1.1 Revision history 2](#_Toc335038089)

[1.2 How to Read this Document 2](#_Toc335038090)

[1.3 References 2](#_Toc335038091)

[1.4 Scope 2](#_Toc335038092)

[1.5 Recommended way of reading 2](#_Toc335038093)

[1.6 Typographical conventions 2](#_Toc335038094)

[1.7 Abbreviations 2](#_Toc335038095)

[1.8 Terminology 3](#_Toc335038096)

[2 General Description 3](#_Toc335038097)

[2.1 Requirements 3](#_Toc335038098)

[2.1.1 Log as few as necessary 3](#_Toc335038099)

[2.1.2 Process data only if it's necessary 3](#_Toc335038100)

[2.1.3 Generally useful 3](#_Toc335038101)

[2.1.4 Required data 3](#_Toc335038102)

[2.1.5 Join elements to more chains 4](#_Toc335038103)

[2.2 Solution of the CLL feature 4](#_Toc335038104)

[2.2.1 Chains of log statements 4](#_Toc335038105)

[2.2.2 Modular structure 4](#_Toc335038106)

[2.2.3 Making decisions 6](#_Toc335038107)

[2.2.4 Turn logging on/off 6](#_Toc335038108)

[3 Functional Interfaces 6](#_Toc335038109)

[3.1 ILogBase 6](#_Toc335038110)

[3.1.1 Naming Conventions 6](#_Toc335038111)

[3.1.2 Public Functions 6](#_Toc335038112)

[3.2 ILog 8](#_Toc335038113)

[3.2.1 Naming Conventions 8](#_Toc335038114)

[3.2.2 Public Functions 8](#_Toc335038115)

[3.3 ILogString 11](#_Toc335038116)

[3.3.1 Naming Conventions 11](#_Toc335038117)

[3.3.2 Public Functions 11](#_Toc335038118)

[3.4 Summary Table of all public functions for EPTF ILog 12](#_Toc335038119)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2012-06-04 | PA1 | First draft version | EILDVNC |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## How to Read this Document

This is the Function Description for the ILog of the Ericsson Performance Test Framework (TitanSim), Core Load Library (CLL).

## 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. EPTF Core Library, CNL 113 512 R12A
3. Generic TitanSim Traffic Generator (GenApp), CNL 113 626 R10A
4. 155 17-CNL 113 512 Uen   
   TitanSim CLL for TTCN-3 toolset with TITAN, Function Specification

## Scope

This document is to specify the content and functionality of the ILog feature set 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

AppLib Application Library

CLL Core Load Library

## 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

## Requirements

TitanSim framework [3] shall support post-mortem logging of faulty traffics, and generating logs for successful traffic cases on demand.

### Log as few as necessary

The logging of long time traffic usually results in huge log files. Most of the log statements are useless, as following a successful traffic has no sense. It is also hard to find the cause of the failures in such huge files, among a lot of unnecessary data. The ILog feature allows writing detailed logs only in case of failed traffic and provides the possibility of writing filtered (short) logs in case of successful traffic cases.

### Process data only if it's necessary

Transforming data elements is very expensive. Therefore, the data elements shall be stored in their original format, and only need to convert if they had to be serialized.

### Generally useful

The primary target of the feature is the traffic logging of the application libraries. However, it shall be possible to use the feature from other features, with different purposes.

### Required data

In case of session-based protocols the logs must contain at least the following:

* Timestamp of the event in ms accuracy
* IP addresses of the session
* Session ID

### Join elements to more chains

There can be different views of event chains. E.g., users may want to log both the successful and failed traffics, the failed one with more details. Therefore, some events are shall be joined to more than one chain. The log item is still stored once but it referenced by multiple chains.

## Solution of the CLL feature

### Chains of log statements

The related log statements are joined to chains. When a procedure is started, a new chain is created. After the procedure is finished successfully, the elements of the chain can be deleted or logged out as a short description of the traffic. If any error occurred and the related chain(s) can be identified (e.g. invalid message received in a session), the content of the associated chain(s) must be logged out.

### Modular structure

The feature can be used from several different features: transport, CLL features, applications, for handling traffic log, or even for logging e.g. the loading of a configuration. The ILog feature supports the logging in any of these features by providing several APIs: ILogBase, ILogString and ILog.

#### Data handling

The log data is stored on the user side in most cases. The ILog feature provides the administrative work on join the references of the log items into chains. Data types of the log items are described by the sources, e.g. AppLib transports and these components store provide a function to serialize the data.

The elements of the same type are stored in a record of data, hereinafter referred to as “database”. When an element is joined to a chain, a new chain item is generated and it stores the reference of the element (database Id and element index in the referred database). This solution provides the possibility of threading log elements with different types into the same chain.

The database elements are read and serialized in case of log out the chain(s) by which they are referenced. When a chain is deleted, its references to its log items are terminated as well. When a log item is no more referenced by any chain, the database element, where it was stored, is reused. Figure 1 shows a sample element allocation in a record of data.

Chain#n

Chain#m

Figure Sample allocation of elements in a log database

The management of the indexes of the free and used elements is done by the bottom layer, called ILogBase.

#### ILogBase

It is the bottom layer of the ILog feature that provides the base data types and the common functions that manage the data structures and the chains of the log items.

This common layer is type independent, as it stores only the chains and the references of user databases and log elements. It handles the chains by using the functions of the NQueue feature and manages the reusing of the items of the user databases by using the FBQ feature.

The chain operations are the following: create new chains, add new element to the chains, log out the content of a chain and delete a chain.

#### ILogString

The ILogString is an upper layer that is based on the ILogBase functionalities. This layer has a built in database, which is a record of charstring elements and it also provides the functions to create a new chain and add elements to the chain, which elements are stored in this database.

This layer also defines the callback functions that are required to create the log header and log out the content of a chain.

#### ILog

The upper layer provides the possibility to use the feature in applications and AppLibs. This layer stores the chain Ids in the application data of the FSM context. The functions that can be invoked from a FSM step are implemented in this layer.

A possible use case is to use several test steps to start a traffic flow in a scenario, and in another scenario they are inside a traffic flow. Only the business logic implemented in the Application level can decide when to start a new chain. Therefore the chains can be started by an FSM test step and obviously logging out the chains can be initiated by an FSM test step too.

### Making decisions

Making the decision e.g. if error occurred, or not. When to start or delete a new chain is the responsibility of the upper level feature. The ILog feature only executes the decisions.

### Turn logging on/off

The turn on/off logging is the responsibility of the caller features.

# Functional Interfaces

## ILogBase

### Naming Conventions

All function names start with f\_EPTF\_ILogBase\_ prefix.

### Public Functions

#### Initialize the ILogBase component

Before using the EPTF ILogBase feature, the following function has to be called:

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

#### Register new database

The user can register a new user database by calling the following function:

*f\_EPTF\_ILogBase\_register\_DB(pl\_logItemFn, pl\_paramList)*

The function requires a reference to the callback function that will be used for logging out the log items that are stored in this database, and a user defined parameter list, which is an empty list by default. The function returns the database id. The user has to use this id as the input of those functions, which requires the database Id.

#### Create new chain

To create a new chain the following function has to be used:

*f\_EPTF\_ILogBase\_newChain(pl\_name, pl\_type, pl\_logHeaderFn, pl\_paramList)*

This function creates a new chain, or reuses an existing one. The chain name has to be defined. The pl\_type parameter defines that the chain is detailed or short. The detailed chain means that all information about the traffic is logged out. The short chain contains less information about the same traffic. The default value of the parameter is detailed.

The chain requires a callback function reference, which function creates the log header. The user also can define a parameter list in the pl\_paramList. The default value of this parameter is an empty list.

#### Add elements to chains

The function *f\_EPTF\_ILogBase\_addToChains(pl\_dbId, pl\_chainIdList)* provides the possibility to add a new element to multiple chains, which are defined in the pl\_chainIdList parameter.

The pl\_dbId parameter requires a database Id that was created with the *f\_EPTF\_ILogBase\_register\_DB* function earlier.

The function returns an integer value that is the index of the log element in the user side database and stores the dbId and the element index in the chains, where the log element was linked. The function returns -1, if any error occurred during the operation and the log item could not be registered.

If the pl\_ chainIdList input parameter is an empty list, the function will link the log element into all registered detailed chains.

#### Log out the content of a chain

The content of a chain can be logged out by using the *f\_EPTF\_ILogBase\_logChain(pl\_chainId, pl\_deleteChain)* function. It logs out the chain, which is identified by the pl\_chainId parameter.

It can be defined with the pl\_deleteChain parameter, that the chain should be deleted after the logging procedure or not. This parameter has a true default value.

#### Log out the content of all chains

The function *f\_EPTF\_ILogBase\_logAll(pl\_deleteChains)* provides the possibility to log out the content of all valid chains. The function has one input parameter, which decides whether the chains have to be deleted after the logging procedure or not. The default value of this parameter is true.

#### Delete a chain

To delete a chain, the user has to invoke the following function:

*f\_EPTF\_ILogBase\_deleteChain(pl\_chainId)*. The pl\_chainId parameter identifies the chain that will be deleted. This function marks the chain and the chain items free and decreases the number of chains that reference the log items that were contained by the deleted chain.

If there is any log items that were referenced only by this chain, the function marks it free as well and this element can be reused in the user side database.

#### Delete all chains

The *f\_EPTF\_ILogBase\_deleteAll()* function deletes all the valid chains.

#### Get the name of a chain

The function *f\_EPTF\_ILogBase\_getName(pl\_chainId)* returns the name of the chain defined with the pl\_chainId parameter. It returns an empty string, if the value of the parameter was invalid.

## ILog

### Naming Conventions

All function names start with f\_EPTF\_ILog\_ prefix.

### Public Functions

#### Initialize the ILog component

Before using the EPTF ILog feature, the following function has to be called:

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

#### Register new database

The following function registers a new user database:

*f\_EPTF\_ILog\_register\_DB(pl\_logItemFn, pl\_paramList := {} )*.

This function invokes the *f\_EPTF\_ILogBase\_register\_DB* function with the parameters given in the input parameters. The role of this function is to hide the ILogBase layer.

The function returns the database id that was returned by the *f\_EPTF\_ILogBase\_register\_DB* function.

#### Create a new detailed and a new short chain

In the ILog feature, the chains are created in pairs. When traffic is logged, a short chain registers log items of the upper layers as AppLib and the application, while the detailed chain logs both this information and the log items created on the bottom layer (transport).

The function, which is called to create these two chains, is the following:

*f\_EPTF\_ILog\_newChains(pl\_eAbsIdx, pl\_fsmCtx, pl\_logHeaderDetailedFn,*

*pl\_paramDetailed, pl\_logHeaderShortFn, pl\_paramShort)*

The function requires the entity index and the FSM context index, to which the chains are defined for. The function stores the chain ids in the application data of the FSM context. The remaining input parameters as the logHeader function references and the pl\_param integer lists are used for calling the *f\_EPTF\_ILogBase\_newChain* function. The default values of these parameters are null and empty list.

#### Get the Id of a chain

The chain Ids are stored in the FSM context. The Id of the detailed and short chain can be retrieved by using the following function:

*f\_EPTF\_ILog\_getChainId(pl\_eAbsIdx, pl\_fsmCtx, pl\_type)*

The function needs the entity index, the FSM context index and the chain type as input parameters and returns the id of the chain as an integer value.

#### Add element to chains

The following function adds a new element to the chains, which Ids are stored in the application data:

f\_EPTF\_ILog\_addToChains(pl\_ptr, pl\_dbIdx)

The pl\_ptr input parameter contains the entity and the FSM index. These determine the chain Ids placed in FSM context, to which the element needs to be added. As the logging into the short chain is turned off by default, the function has to check the state of the logging with the *f\_EPTF\_ILog\_shortLogEnabled* function (see description in 3.2.2.11). If the short log enabled the function adds the new element to both chains, otherwise it adds only to the detailed chain.

The function returns the index of the log element that identifies its place in the user side database.

#### Logging successful traffic

By using the following function the content of the short chain can be logged out by the appropriate step of the FSM:

*f\_EPTF\_ILog\_step\_logSuccess(pl\_ptr)*.

The function logs out the content of the short chain, which defined with the pl\_ptr pointer. The function also deletes the detailed chain, as it is no more used.

#### Logging failed traffic

In case of failed traffic, both the short and the detailed chain have to be logged out. The following function is used by the FSM step, which invokes the logging mechanism:

*f\_EPTF\_ILog\_step\_logFailed(pl\_ptr)*.

This function creates the log for both detailed and short chains, which ids are retrieved by using the *f\_EPTF\_ILog\_getChainId* function. The chains are deleted after the end of the logging mechanism.

#### Delete log

In case of successful traffic, the Application level can delete both the detailed and the short chains by using a step in the FSM, which invokes the following function:

*f\_EPTF\_ILog\_step\_deleteLog(pl\_ptr)*.

This function deletes the detailed and the short chain, which belongs to the entity identified by the pl\_ptr input parameter.

#### Enable short log for the next step

By default the logging into the short chains is disabled and there is a possibility to enable it for one step at a time. The step, which enables the it calls the *f\_EPTF\_ILog\_step\_shortLogNext(pl\_ptr)* function to set a flag in the AppData of the FSM context. The short chain logging is disabled at the end of the step that it was enabled for.

#### Disable short log

The short logging can be disabled with the following function, which sets an element in the application data assigned to the FSM context:

*f\_EPTF\_ILog\_disableShortLog(pl\_ptr)*.

#### Check if short log is enabled

The following function checks whether the short log enabled or not:

*f\_EPTF\_ILog\_shortLogEnabled(pl\_eAbsIdx, pl\_fsmCtx)*. It returns true if it is true and false otherwise.

## ILogString

### Naming Conventions

All function names start with f\_EPTF\_ILogString\_ prefix.

### Public Functions

#### Initialize the ILogString component

Before using the EPTF ILogString feature, the following function should be called:

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

#### Create new chain

The function *f\_EPTF\_ILogString\_newChain (pl\_name, pl\_logHeaderFn, pl\_paramList)* creates a new chain by using the *f\_EPTF\_ILogBase\_newChain* function and returns the chain Id of the crated chain.

#### Add new element to chains

New element can be added to chains with the *f\_EPTF\_ILogString\_addToChains(pl\_chainIdList, pl\_logElement, pl\_timeOfLog)* function.

This function stores the charstring given in the pl\_logElement input parameter in its own database with the timestamp defined in the pl\_timeOfLog parameter. The function returns 0, if the operation was successful and -1 otherwise.

#### Log header function

The header for the chains is created with the following function:

*f\_EPTF\_ILogString\_logHeader(pl\_chainId, pl\_paramList)*.

It generates the following string to the start of the chain log:

“It is the log of CHAIN NAME chain, with params: PARAMETERS“

The CHAIN NAME identifies the name of the chain, which is retrieved with the *f\_EPTF\_ILogBase\_getName* function and the PARAMETERS means the elements of the user defined parameter list.

#### Log item function

The content of the chains is logged out with the following function:

*f\_EPTF\_ILogString\_logItem (pl\_elementIdx, pl\_dbId, pl\_chainId, pl\_paramList).*

It creates a string starting with the time stamp in milliseconds precision, the log items that were stored in the logged out chain, the database id and the user defined parameter list.

## Summary Table of all public functions for EPTF ILog

Table 1. Summary of ILog functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_ILog\_init\_CT | Initializes the ILog feature |
| f\_EPTF\_ILog\_register\_DB | Registers a new user database |
| f\_EPTF\_ILog\_newChains | Create new chains: a short and a detailed chain |
| f\_EPTF\_ILog\_getChainId | The function retrieves the id of a chain from the application data, assigned to the FSM context |
| f\_EPTF\_ILog\_addToChains | Add log item to the detailed (and to the short) chain(s) |
| f\_EPTF\_ILog\_step\_logSuccess | Log out the sort chain and delete the detailed chain |
| f\_EPTF\_ILog\_step\_logFailed | Log out the short and the detailed chains |
| f\_EPTF\_ILog\_step\_deleteLog | Delete the short and the detailed chains |
| f\_EPTF\_ILog\_step\_shortLogNext | Enable the short chain log to the next step of the FSM |
| f\_EPTF\_ILog\_disableShortLog | Disable the short chain log |
| f\_EPTF\_ILog\_shortLogEnabled | Check if short log is enabled |
| f\_EPTF\_ILogBase\_init\_CT | Initializes the ILogBase feature |
| f\_EPTF\_ILogBase\_register\_DB | Registers a new user side database |
| f\_EPTF\_ILogBase\_newChain | Creates a new chain |
| f\_EPTF\_ILogBase\_addToChains | Add a new log item to chains |
| f\_EPTF\_ILogBase\_getName | Returns the new of a chain |
| f\_EPTF\_ILogBase\_deleteChain | Deletes a chain(marks it and its items free) |
| f\_EPTF\_ILogBase\_logChain | Logs out the content of a chain and deletes it by default |
| f\_EPTF\_ILogBase\_logAll | Logs out the content of all valid chains in the chain database and deletes them by default |
| f\_EPTF\_ILogBase\_deleteAll | Deletes all valid chains using the f\_EPTF\_ILogBase\_deleteChain function |
| f\_EPTF\_ILogBase\_dumpToPng | Debug function to check the content of the chains and the FBQs of the user databases |
| f\_EPTF\_ILogString\_init\_CT | Initializes the ILogString feature |
| f\_EPTF\_ILogString\_newChain | Creates a new chain |
| f\_EPTF\_ILogString\_addToChains | Adds a new elements to chains and saves the element in its internal charstring database |
| f\_EPTF\_ILogString\_logHeader | Callback function to create the log header to a chain |
| f\_EPTF\_ILogString\_logItem | Callback function to log out a log item |