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

EPTF Core Library Execution Control, Function Description

Contents

[1 Introduction 3](#_Toc485077650)

[1.1 Revision history 3](#_Toc485077651)

[1.2 How to Read this Document 4](#_Toc485077652)

[1.3 Scope 4](#_Toc485077653)

[1.4 Recommended way of reading 4](#_Toc485077654)

[1.5 Typographical conventions 4](#_Toc485077655)

[1.6 Abbreviations 4](#_Toc485077656)

[1.7 Terminology 5](#_Toc485077657)

[2 General Description 7](#_Toc485077658)

[2.1 Overview of Execution Control Functionality 7](#_Toc485077659)

[2.2 Configuration 8](#_Toc485077660)

[2.3 LGen Pools 9](#_Toc485077661)

[2.4 Initialization sequence of ExecCtrl 9](#_Toc485077662)

[2.4.1 The initialization process 9](#_Toc485077663)

[2.4.2 Messages during initialization 12](#_Toc485077664)

[2.5 Time Profiles 15](#_Toc485077665)

[2.6 Regulators 16](#_Toc485077666)

[2.7 Group Finish Condition Handling 17](#_Toc485077667)

[2.7.1 The onLaunchFinish finish type 17](#_Toc485077668)

[2.7.2 The onGroupFinishCondition finish type 18](#_Toc485077669)

[2.8 Single Shot Traffic 21](#_Toc485077670)

[2.9 Limited Traffic Execution 21](#_Toc485077671)

[2.10 Multiple Execution Phases 21](#_Toc485077672)

[2.11 Execution of scenarios and traffic cases 22](#_Toc485077673)

[2.12 Final test report generation 23](#_Toc485077674)

[2.13 GUI and Statistics 24](#_Toc485077675)

[2.13.1 Distributing statistics calculation 24](#_Toc485077676)

[2.13.2 Progress information during startup 25](#_Toc485077677)

[2.14 Command Line Interface 25](#_Toc485077678)

[2.14.1 List of ExecCtrl commands 25](#_Toc485077679)

[3 Functional Interface 27](#_Toc485077680)

[3.1 Naming Conventions 27](#_Toc485077681)

[3.2 Public Functions 27](#_Toc485077682)

[3.2.1 Initialization 27](#_Toc485077683)

[3.2.2 Loading the configuration parameters 28](#_Toc485077684)

[3.2.3 Enabling progress updates in UIHandler 29](#_Toc485077685)

[3.2.4 Starting the operation of ExecCtrl 29](#_Toc485077686)

[3.2.5 Terminate ExecCtrl 29](#_Toc485077687)

[3.3 Summary Table of all public functions for EPTF ExecCtrl 30](#_Toc485077688)

[3.3.1 Functions that were removed or became private in the R3 version of ExecCtrl 37](#_Toc485077689)

[3.4 Summary Table of all public functions for EPTF ExecCtrl UI Handler 38](#_Toc485077690)

[3.5 Summary Table of all public functions for EPTF ExecCtrl CLI 39](#_Toc485077691)

[3.6 Execution Control Client 39](#_Toc485077692)

[3.7 Summary Table of all public functions for EPTF ExecCtrlClient 40](#_Toc485077693)

[3.8 Summary Table of functions for EPTF ExecCtrlClient that are no longer public in R3 41](#_Toc485077694)

[3.9 Summary Table of all public functions for EPTF ExecCtrlClient UIHandler 42](#_Toc485077695)

[3.10 Public Variable names in ExecCtrl UIHandler 43](#_Toc485077696)

[3.11 Public widget IDs on the ExecCtrl UIHandler GUI 46](#_Toc485077697)

[4 References 46](#_Toc485077698)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-12-06 | PA1 | First draft version | ETHZTO |
| 2007-12-07 | PA2 | Updated according to review comments | ETHZTO |
| 2008-01-30 | PA3 | ExecCtrl UI handler added | ETHJGI |
| 2008-03-17 | PA4 | TimeProfile/Regulator added | ETHJGI |
| 2009-02-19 | PB1 | Updated for the R3 version | ETHJGI |
| 2009-04-20 | PB2 | Added more R3 functionality | ETHJGI |
| 2009-05-29 | PB3 | R3 update | ETHJGI |
| 2009-09-10 | PC1 | Update of initialization msg sequence | ETHJGI |
| 2009-10-22 | PC2 | Phase dependent regulation | ETHJGI |
| 2010-01-13 | PC3 | StatusVars added, new public functions added | ETHJGI |
| 2010-03-26 | PD1 | CLI commands added, new public functions added | ETHJGI |
| 2010-06-04 | PE1 | New functions added, manual entity group distribution added, start function, terminate test execution added, ExecCtrlClient loadconfig added | ETHJGI |
| 2010-07-02 | PF1 | Custom GUI refresh rate added | ETHJGI |
| 2011-01-21 | PG1 | Limited Traffic Execution added | ETHJGI |
| 2011-03-03 | PG2 | Data elements and iterators for custom GUI layout | ETHJGI |
| 2011-08-05 | PH1 | TC statistics reset at start | ELSZSKU |
| 2012-07-26 | PJ1 | Update of the progressbar in UIHandler | ETHJGI |
| 2012-09-13 | PK1 | New API functions added: getLGenCompRefsOfPool, checkCreatorFnNameOfPool | ETHJGI |
| 2013-01-28 | PL1 | New delta statistics to be defined and made them datasourced for the given set of basic cll statistics. | EKOVIST |
| 2014-11-25 | PM1 | New chapter added (2.14.1.9) | ESZILSZ |
| 2014-12-03 | M | Updated for release | ESZILSZ |
| 2015-05-20 | PN1 | Graphics replaced with new ones | ESZILSZ |
| 2015-06-19 | N | Updated for release and graphics changed back to the old ones. | ESZILSZ |
| 2016-05-19 | PS1 | StatHandler distribution + group finish evaluation update | ETHJGI |
| 2016-06-09 | S | Updated for release | ESZILSZ |
| 2016-10-14 | PT1 | Updated chapters 2.13.1, 3.3.1 | EVAURBA |
| 2016-10-15 | PT2 | Updated chapter 3.3 | EVAURBA |
| 2016-10-15 | PT3 | Updated revision history | EVAURBA |
| 2016-10-17 | PT4 | Editorial changes | EIMRENA |
| 2016-10-17 | PT5-PT6 | Editorial changes | EIMRENA |
| 2016-12-02 | T | Updated for release | EIMRENA |
| 2017-01-20 | PU1-PU3 | Chapter update (2.13.1) and editorial changes | EVAURBA |
| 2017-03-21 | PU4-PU5 | Chapter update (3.3) | EZSFTRA |
| 2017-06-09 | U | Updated for release | EIMRENA |

## How to Read this Document

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

## Scope

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

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of EPTF 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. Moreover, whenever a concept is mentioned that has a special meaning as described in the Glossary (Section 1.8) of this document, then these occurrences are marked with an initial arrow, e.g., *🡪 Entities.*

## Abbreviations

CLI Command Line Interface, its functionality is provided by the EPTF CommandLineInterface feature

CLL Core Library

EPTF Ericsson Performance Test Framework

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

*ExecCtrl* is function set for provisioning and executing test *🡪 scenarios* with TitanSim based load generators

*ExecCtrl CLI* this provides ExecCtrl commands on the Command Line Interface

*ExecCtrl UI Handler* extends the *🡪 ExecCtrl* functionality with user interface

*ExecCtrlClient* the client for *🡪 ExecCtrl*, it is the interface of LGenBase towards *🡪 ExecCtrl* used to control the LGenBase

*ExecCtrlClient UI Handler* extends the *🡪 ExecCtrlClient* functionality with user interface

*Behavior type* realizing methods conforming to given function signatures prescribed by the LGenBase feature. Behavior types are to be declared dynamically during run-time to the LGenBase by the component-type initialization founction of some AppLib. Usually an AppLib product has only one behavior type, but it is permitted for AppLibs to declare more than one behavior types, if necessary. Behavior types are identified by their indices determined by the order of declaration and the declarations are stored in v\_LGenBase\_behaviorTypes

*Entities* are “things/objects” that are simulated by the LGen. Such “things” are usually the “users”, “calls”, “servers”, “terminations”, or anything whose external behavior is to be simulated by the LGen. Usually an LGen simulates more than one entity.

*Entity context* is an index-able generic data record describing an entity. It serves as a hub connecting all information related to the entity together by referring the respective associated *🡪 behavior contexts* and *🡪 FSM contexts* via their indices. The entity contexts are identified via their indices with respect to the entity context database stored in the v\_LGenBase\_entities component variable.

*Entity type* is a named list of references to declared *🡪 behavior types*. As such, it declares what types of behavior contexts are available on entities of this type.

*Entity group* A group of entities with homogenous entity type. The indices of the entities of the group make a continuous interval that does not overlap with the indices range of any other entity group.

*LGen* Load Generator that uses the LGenBase feature, *🡪 ExecCtrlClients* are also LGens

*Scenario* is a collection of traffic cases deployed on an entity group

*Finite State Machine* is a formal description of *🡪 Control Logic*. It is a tabular description of what *🡪test steps* to call when a given *🡪reported event* occurs, when the entity is in a given declared 🡪 *state of the FSM*. After executing these *🡪 test steps* an optional next *🡪 FSM state* can be also defined.

*FSM table* is a formal specification of the communication rules of Finite State Machines. These indexable records are stored in v\_LGenBase\_fsmTables. FSM tables realizing *🡪 traffic cases* must obey certain rules.

*Traffic case type* is a prototype used as an input for instantiating traffic case instances on a specific entity group. It defines what *🡪 FSM table* to use, what *🡪 entity type* must be used by the *🡪 entity group* where the *🡪 traffic case* is to be deployed

*Traffic case* is a behavior, defined by an FSM table, that is executed on a given entity group. All entities of the entity group participate in executing the the traffic case. An entity group may have more than one concurrent traffic cases.

# General Description

This document specifies the Execution Control feature of the TitanSim CLL.

The EPTF Execution Control feature makes it possible to

* Manage entity groups, scenario types, traffic case types
* Manage scenario type – entity group associations
* Decompose parameters for Load Generators
* Provision Load Generators with traffic case parameters
* Start/stop traffic scenarios on entity groups on multiple Load Generators

The aim of the EPTF ExecCtrl feature is to provide common framework for configuring and handling multiple Load Generators in one test scenario.

To be able to use EPTF ExecCtrl, the user should start a component that extends the EPTF\_ExecCtrl\_CT component and the Load Generators should extend at least EPTF\_LGenBase\_CT [5] and EPTF\_ExecCtrlClient\_CT components.

The additional Execution Control UI handler feature makes it possible to display some important Execution Control information on the GUI and also creates Start/Stop buttons. The Execution Control UI handler is an extension of the Execution Control feature.

## Overview of Execution Control Functionality

The Execution Control feature consists of two parts: the ExecCtrl component and the ExecCtrlClient component. The ExecCtrl component is the main component, it controls the ExecCtrlClient components, which are the actual LGen components extending the LGenBase feature.

The ExecCtrl component

* creates LGen pools if configured to do so
* waits for the preconfigured number of ExecCtrlClient connections
* decomposes and distributes the entity group, traffic case, traffic scenario descriptors to ExecCtrlClients (Load Generators)
* initiates and keeps track of traffic scenario execution
* synchronizes group finish events among ExecCtrlClients
* calls user defined scenario state change events handlers when the state of the scenario is in-sync on all LGens
* manages execution phases
* calculates statistics
* manages final test report generation

The ExecCtrlClient component:

* notifies the ExecCtrl about its presence, sends resource info
* creates entity groups, scenarios, traffic cases as requested from the ExecCtrl
* calls user defined callbacks when the configuration is finished
* notifies ExecCtrl that the component is ready to start executing traffic cases
* executes traffic scenarios as requested by ExecCtrl
* synchronizes groupFinish conditions among LGens
* handles execution phases

The user should specify the test scenario description via Titan run-time configuration file.

A TITAN Runtime GUI is required to run ExecCtrl component and provides interactivity with the tester if the following components are used: The ExecCtrl UI Handler component can be used instead of the ExecCtrl component to display Execution Control information on the GUI under the Execution Control tab. This component makes all functionality of the ExecCtrl feature available also.The EPTF\_ExecCtrlClient\_UIHandler\_CT component displays the LGenBase statistics on the Clients tab of the ExecCtrl panel.

## Configuration

The ExecCtrl feature uses module parameters to configure its behavior. Module parameters are used for setting up the scenarios, entity groups, traffic cases, time profiles, regulators, scenario groups and phases used in the test. Some of the module parameters come from the LGenBase feature. They are reused in ExecCtrl when scenarios are configured. Module parameters control also the debugging and manual control modes and specify how many LGens are created manually (i.e. by the user which will be placed into the default LGen pool). More details on module parameters can be found in the User Guide of ExecCtrl.

The configuration of ExecCtrl is also possible using the loadConfig API function (see 3.2.2). The ExecCtrl module parameters are loaded only if not disabled in the init function of ExecCtrl. By default, the module parameters are loaded automatically.

## LGen Pools

It is possible to organize the LGens into pools. LGen pools consist of LGens running on different hosts. It is possible to define that on which hosts the LGens should be running, how many LGens should be created on a host, and how should the LGen be created and which behavior function should it execute. The ExecCtrl component deploys the pools automatically at startup. The component indexes belonging to any given pool can be requested later on. It is also possible to check if a given creator function was used in a given pool.

The LGens of the pools are created by user defined functions, called the LGen creator functions. The LGen creator functions are implemented by the user. They should create and start the LGen components and should be registered into ExecCtrl so that the ExecCtrl can use them. The only restriction on the LGen creator functions is that they should run on a component that extends the ExecCtrl component. It is possible to configure the LGen pools via the configuration file also.

Entity groups can be assigned to LGen pools. In this way an entity group will be distributed on the LGens of the assigned pool only, so the traffic cases will be distributed and executed only on the LGens of the pool. One entity group can be assigned only to one (or zero) LGen pool.

The LGen pool creation is automatic; the user does not have to bother with the LGen deployment. However, LGens created by the user will be automatically added to the ‘default pool’. Entity Groups that are not added to any LGen pool will be distributed on the default pool.

## Initialization sequence of ExecCtrl

### The initialization process

The initialization of ExecCtrl and the ExecCtrlClient is done by calling their init functions. The configuration parameters of ExecCtrl are loaded automatically from the module parameters. If additional configuration parameters have to be loaded, it can be done by calling the configuration loading function. This function can be called more than one times, and it can be called before the init function, but has to be called before the start function of ExecCtrl is called. Configuration data loaded after the start function are ignored. The init function of ExecCtrl calls the start function automatically if not disabled, and creates the LGens of the configured LGen pools.

Note, that on the ExecCtrlClient component the LGenBase and the application specific data, including entity types has to be initialized. The ExecCtrlClient init function should be called after that, because it uses this data to inform ExecCtrl about the supported entity types.

It is possible to initialize application data after the init function of ExecCtrlClient, but in this case the automatic calling of ReadyToStart function in the init function should be disabled and the readyToStart function should be called after all entity types are declared.

Configuration data can be loaded into ExecCtrl on the ExecCtrl client before readyToStart is called by the load config function. After readyToStart was called, no more configuration data can be loaded.

The following figure shows the initialization process of ExecCtrl:

ExecCtrl

ExecCtrlClient

f\_EPTF\_ExecCtrl\_init\_CT

f\_EPTF\_ExecCtrl\_loadConfig, setRefreshRates

Enter main alt-loop

f\_EPTF\_LGenBase\_init

Applib init, including f\_EPTF\_LGenBase\_declareBehaviorType

f\_EPTF\_LGenBase\_declareStep

f\_EPTF\_LGenBase\_declareCompactFsmTable

f\_EPTF\_LGenBase\_declareEntityType

Applib init can be here as well, but in this case readyToStart should be disabled in the init function

f\_EPTF\_ExecCtrlClient\_readyToStart if ExecCtrlClient init did not call it

Enter main alt-loop

Figure 1: Initialization of ExecCtrl

Or Create pools automatically

f\_EPTF\_ExecCtrlClient \_init\_CT

f\_EPTF\_ExecCtrl\_LGenPool\_createLGens

f\_EPTF\_ExecCtrl\_start

f\_EPTF\_ExecCtrlClient\_loadConfig

### Messages during initialization

When the ExecCtrlClient is ready to start, it notifies ExecCtrl by sending its resource information to it. This message contains the supported entity types and that how much is available from them. After ExecCtrl had received all resource messages from its clients it starts to configure them. The entity groups are allocated first. After each entity group of a given entity type is allocated on an LGen, the LGen sends the resource info message again back to ExecCtrl so that it can update its resource database.

After the entity groups are configured, the traffic cases and the scenarios are declared on the LGens. When the configuration of an ExecCtrlClient is finished, ExecCtrl sends an EndOfConfig message to the ExecCtrlClient. It is possible to set callback functions in the client associated for this event. These callback functions are called when the client receives the EndOfConfig message. After all callback functions are called, the ExecCtrlClient sends a ReadyToRun message to the ExecCtrl. ExecCtrl waits until ReadyToRun messages from all ExecCtrlClients arrive. If ExecCtrl UI handler is used, this is when it updates the GUI. Then the execution of traffic cases can begin.

The following figure shows the initialization message sequence between the ExecCtrl and the ExecCtrlClient.

ExecCtrl

ExecCtrlClient

Login with Resource info

(Ready to start)

Wait until all clients are logged in

CreateEntityGrpList msgs

DeclareTcs msgs

CreateScenario msgs

EndOfConfig

Call EndOfConfig callbacks,

ExecCtrlClient\_UI creates its GUI

ReadyToRun

Wait until all clients are ready to run

Resource info msg after each CreateEntityGrpList msg

Ready to execute TCs

Start execute TCs, ExecCtrlUIHandler creates the GUI

Ready to start

GuiDone

GuiDone

Wait until GuiDone is received from all clients

Load config

Start

Figure 2. Message sequence between ExecCtrl and ExecCtrlClient during initialization

#### Entity Group distribution

The ExecCtrl distributes entity groups on the LGens automatically based on the declared entity type information on the LGens and the entity group-LGen pool associations. Only those scenarios and traffic cases will be created on an LGen which belong to the entity groups distributed on them.

The entity groups are distributed automatically on the LGens, but a custom distribution can be specified by module parameter.

The entity group distribution algorithm tries to allocate equal number of entities on the LGen supporting the entity type of the entity group in the LGen pool. If it is not possible, entities are allocated to the LGens until all of them is allocated, or until there is no more entity left on the LGens.

The manual entity distribution can be specified for an entity group so that for every LGen a weight factor has to be given. The entity group will be distributed according to the specified weights. If no weight is specified for an LGen the weight will be considered as being zero. For the last LGen all the remaining entities will be allocated.

All entities have a global and unique ID. When the entity groups are distributed, each LGen is notified about the global ID of the first entity in the global entity group (the global offset), and the offset within the global entity group of the first entity in the entity group distributed to the LGen. See the next figure for details:

LGen 1

(X, 0)

X

X+N

Number of entities: n1

LGen 2

(X, n1)

Number of entities: n2

LGen 3

(X, n1+n2)

Number of entities: n3

(N = n1+n2+n3)

X+n1

X+n1+n2

Global Id

(Global Id, relative Id)

All entities

Entity group

Number of entities: N

0

M

Number of entities: M

Figure 3. Entity group distribution on LGens

## Time Profiles

It is possible to define time dependent traffic case or scenario CPS adjustments with the help of time profiles. Time profiles can be created graphically by the TimeProfileEditor CLL feature. ExecCtrl can use the created time profiles to control the CPS level of traffic cases or scenarios. Each traffic case CPS or weighted scenario CPS can be associated with a time profile at configuration time.

The time profiles are started automatically when the “Start Test” button is pressed on the ExecCtrl GUI or when all scenarios are started. The time of the time profile i.e. the ‘simulated time’ is shown at the bottom of the GUI as the “Time elapsed since Test was started”. The time profiles are not started if a scenario or a traffic case is started separately.

## Regulators

Regulators are similar to Time Profiles in a sense that they can also be used to regulate the CPS of scenarios and traffic cases. The regulators created using the LoadRegulator CLL feature can be connected to ExecCtrl and can be configured to control the CPS levels so that e.g. the load level on a target host should reach a predefined target load level. Certain parameters of the regulators and what they control are displayed on the ExecCtrl GUI.

The same regulator can be used to control several CPS-es at the same time. It is possible to define a weight factor for every controlled CPS to determine how much fraction of the total CPS should belong to a given traffic case/scenario CPS.

The regulation can be selected for a given execution phase, i.e. if the scenario belongs to a scenario group with different phases, the regulator can be configured to regulate the CPS only in a given phase. In this case the regulator regulates the CPS of a scenario or traffic case only if the currently executing phase is the same as which was selected for the regulator.

Regulators can be assigned to CPS-es during run-time also using the ExecCtrl GUI.

The regulators have the following states:

* Disconnected: regulator is configured, but not started (not registered into ExecCtrl)
* Connected: regulator is registered into ExecCtrl
* Disabled: regulator does not assigned to any CPS-es, or none of the regulated traffic cases/scenarios are running
* Unstable: regulator is working, the target load is far from the currently measured value, CPS is changed dynamically
* Stable: target load is reached, CPS is stabilized
* Auto-off: The limit-max regulator detected that the load is below the regulation threshold, no regulation is necessary. Regulation is switched off until the measured load exceeds the target value.

When ExecCtrl GUI is used, these states are displayed on the GUI. The TargetValue (total CPS) is distributed among the regulated CPS values according their weight factors. This value cannot be changed if the regulator in not in Disconnected state, its value is set by the regulator automatically. In Disconnected state changing this value by the user makes it possible to regulate the CPS ‘manually’ as if the regulator would do.

If the regulator is in Auto-off state, it is possible to change the regulated CPS values of the traffic cases/scenarios. In this case the TotalValue is updated automatically. When the load exceeds the target value, regulator starts regulating the CPS to keep the load at the target value. When load decreases below the target load, the CPS is increased until the previous value in Auto-off state is reached. The regulator in this case will enter into Auto-off state again.

During regulation, the target load value can be changed anytime. The regulator will try to adjust the CPSes until the measured load reaches the new target load.

## Group Finish Condition Handling

The ExecCtrl feature distributes a scenario to a couple of LGens. This means that the same scenario is executed on several LGen components at the same time. The parameters of the scenarios and the traffic cases in them are split among the LGens. This parallel execution requires the synchronization of the different conditions in the scenario. This section explains how the group finish condition is synchronized among the ExecCtrlClient components.

### The onLaunchFinish finish type

The fulfillment of onLaunchFinish finish type is reported by each ExecCtrlClient component to the ExecCtrl component, which waits until all clients reported it. If the condition is fulfilled on all clients, ExecCtrl notifies them, and they continue their operation as if the condition was fulfilled right now, i.e. the actions for the condition are called.

Figure 4. The onLaunchFinish synchronization

ExecCtrl

ExecCtrlClient

ExecCtrlClient

onLaunchFinish fulfilled

onLaunchFinish fulfilled

Check if onLaunchFinish is fulfilled on all LGens

onLaunchFinish fulfilled

Send onLaunchFinish fulfilled to all clients

Call actions for onLaunchFinish

Call actions for onLaunchFinish

### The onGroupFinishCondition finish type

The onGroupFinishCondition differs from onLaunchFinish that it has certain conditions that should be fulfilled. These conditions should be synchronized among the LGens.

All conditions

* are evaluated locally at the ExecCtrlClient and then synchronized on the ExecCtrl component. The clients are notified if the condition is fulfilled.

The conditions are checked at the LGen and then the ExecCtrl is notified. Then the ExecCtrl synchronizes these conditions by waiting until all LGens had reported the fulfillment of the condition. Then ExecCtrl notifies the LGens that the condition is fulfilled in all LGens, and the LGens call their registered groupFinish actions. This synchronization is similar to the onLaunchFinish synchronization. At the end the onGroupFinish callback functions in ExecCtrl are called. If a condition that is finished on one LGen does not get fulfilled on all LGens in a certain time (configurable by tsp\_EPTF\_ExecCtrl\_GroupFinishedGuardTime) ExecCtrl will redistribute the remaining part of the condition among the LGens to speed up the fulfillment. This is true only for redistributable conditions.

List of non/redistributable onGroupFinish conditions:

* nofRangeLoop,
* execTime,
* entitiesFinished,
* custom,
* allDone

These conditions are evaluated in the ExecCtrl. This means the ExecCtrl collects all necessary data from the ExecCtrlClients that are needed for the evaluation of the condition. If the condition is evaluated true, the ExecCtrlClients are notified so that they can call the groupFinish actions.

List of redistributable onGroupFinish conditions:

* nofExecStart,
* nofSuccess,
* nofFail
* nofTimeout
* nofError

Note that it is not guaranteed, that the redistributable conditions are fulfilled exactly. The fulfillment of the redistributable conditions may be detected a bit late, this means that e.g. the nofExecStart condition will be fulfilled when the nofExecStart counter reaches or exceeds the defined threshold.

Figure 5. The redistributable onGroupFinishCondition synchronization

ExecCtrl

ExecCtrlClient

ExecCtrlClient

Periodic update of data source on LGens

Is onGroupFinishCondition fulfilled globally?

onGroupFinishCondition fulfilled

Send onGroupFinishCondition fulfilled to all clients

Call actions for onGroupFinishCondition

No

Yes

Call actions for onGroupFinishCondition

Call callbacks of onGroupFinish

Redistribute condition if guard time expires

## Single Shot Traffic

For each traffic case it is possible to start the traffic case only on one entity. This is called single shot traffic. The entity to run the single shot traffic can be selected by the user from the entities of the entity group the traffic case belongs to. Otherwise it will be determined automatically by the system. Also the logging of single shot traffic can be enabled or disabled. The single shot traffic can be started in any state of the traffic case. If the entity group is deployed on more than one LGen component, ExecCtrl will automatically determine which LGen component contains the selected entity and start the traffic case on that LGen.

## Limited Traffic Execution

It is possible to limit the number of entities which are active during a traffic case execution. This number of active entities can be any value between zero and the number of allocated entities. The number of allocated entities is equal or smaller than the size of the entity group. It is smaller if it was not possible to allocate the entire entity group on the LGens due to limited resources. The initial number of active entities of an entity group is set to the number of allocated entities and later its value can be set or get by API functions or can be adjusted on the ExecCtrl GUI. The value set for the number of active entities is then distributed on the LGens proportional to the number of allocated entities on them. The CPS distribution is determined proportionally to the number of active entities on the LGens.

If a traffic case or a scenario is started on an entity group of which the number of active entities is reduced, the traffic will be executed only on the active entities. In case of one active entity, limited execution resembles the single shot traffic, but is different because in the limited traffic execution case traffic finish actions are executed as well. For further details see the LGenBase documentation.

## Multiple Execution Phases

Execution Control component supports synchronized execution phases.

TitanSim R1 supported three phases of execution: pre, load generation and post phase. This was removed from the ExecCtrl of TitanSim R2. A scheme similar to R1 is implemented in the ExecCtrl, but with arbitrary number of named execution phases.

It is possible to configure which scenarios/traffic cases are enabled for which execution phases. The currently selected execution phase can be sent to all ExecCtrlClients and set in LGenBase.

* The user is able to declare phases that are executed consecutively on a group of Scenarios. (Declaration can appear as function call or in config file)
* Phases can be executed in MANUAL or AUTOMATIC modes. In the MANUAL mode only the current phase is executed, while in AUTOMATIC mode, the next phase is started after the current one has finished.
* Phases are synchronized for a Scenario group.
* The Execution Control GUI provides the means to control (start/stop/terminate) the scenario groups and their state information is also displayed.
  + The current phase in scenario group can be stopped. In this case when the current phase has finished, the next state is started in AUTOMATIC mode.
  + The execution of the scenario group can be terminated. In this case the current phase is stopped and also all traffic cases. The next phase will not be started automatically; the execution of all traffic cases in the scenario group will be stopped.
* Regulation of CPS-es can be selected for a given phase
* Call back function can be registered to the phase change event

## Execution of scenarios and traffic cases

The execution of scenarios can be configured to start automatically after startup, or started by the user. The execution of scenarios and traffic cases can be started/stopped manually by pressing the “Start Test”/”Stop Test” buttons on the ExecCtrl GUI, or by pressing the start/stop button of the scenarios or traffic cases separately. Execution of all traffic cases can be terminated by pressing the “Terminate Test” button.

Individual scenarios can be started, stopped or reset by the “Start Scenario”, “Stop Scenario” and “Reset Scenario” buttons.

Scenarios that are in scenario groups can only be started/stopped by the “StartCurrentPhase” checkbox. To terminate the execution of the scenario group the “TerminateGroup” button has to be pressed. Terminating a scenario group means that the state of the current phase will be changed to idle, and no phase finish actions will be called. When the scenario group is stopped by unpicking the “StartCurrentPhase” checkbox, the phase finish actions are executed and in AUTOMAIC mode, the next phase will be started after the current phase is stopped.

The scenario group state can be reset by pressing the “ResetGroup” checkbox. All data in the scenario will be reset, and the phase will be changed back to the initial phase.

## Final test report generation

The ExecCtrl feature initiates the final test report mechanism when the test campaign ends. The test finishes, when the Exit button is pressed, or if any of the LGens notifies the ExecCtrl that the test ended. It is possible to register report generator functions in ExecCtrl and in ExecCtrlClients. These functions may return a charstring that will be combined in ExecCtrl to create the final test report message. The following figure shows the final test report generation mechanism.

Figure 6. The final test report generation

ExecCtrl

ExecCtrl GUI

ExecCtrlClient

Wait for Exit Button press, or EofTest in any Clients

EofTest to all clients

EofTestDone containing the report

Wait until all clients responded

Call final test report generator functions

Call the final test report generator functions

EofTest

Exit Button pressed

OR

Terminate execution if not in manual control mode

## GUI and Statistics

The ExecCtrl\_UIHandler extension of ExecCtrl visualizes the ExecCtrl parameters and statistics on the RuntimeGUI. It provides control buttons to perform various actions during the test.

If you start a traffic case from the GUI, it resets the statistics of the appropriate traffic case to avoid the unexpected behavior.

Statistics are corrected during runtime and are displayed on the GUI.

The refresh period for different GUI element categories can be set to custom values using module parameters or API functions.

It is possible to create custom GUI for ExecCtrl. This can be done by using the data elements and iterators defined by ExecCtrl. These iterators and data elements are created if the dataSource component is passed to the ExecCtrl init function. The ExecCtrlUIHandler is not needed to create the custom GUI. The detailed list of available data elements and iterators, and their parameters are available in the HTML API doc.

### Distributing statistics calculation

If a lot of statistics (both TC and FSM) needs to be calculated during runtime, the calculation may overload the ExecCtrl component. To avoid this overload, statistics calculation can be configured to be calculated on separate components. These components will be started automatically by ExecCtrl. The number of statistics calculating components can be specified by the pl\_nofExternalStatHandlers argument of the ExecCtrl init function f\_EPTF\_ExecCtrl\_init\_CT. The statistics are distributed among these components, so that the statistics for the same traffic case are calculated by the same StatHandler component.

If the pl\_nofExternalStatHandlers parameter is set to 0 (default), ExecCtrl will calculate all statistics by its internal StatHandler. For any positive value the number of new StatHandlers will be determined by this parameter. This means that the statistics calculation will be performed on the new StatHandler components, and the ExecCtrl component will not calculate statistics.

Please note, that the internal traffic between the ExecCtrl and the additional StatHandler components can be high due to the internal communication between ExecCtrl and StatHandler components. This can be reduced by using pull-model to update the values of the statistics. This means that only those statistics will be calculated which values are requested. Also, internal traffic will be decreased due to the fact that not all statistics values are needed at all times. To enable this feature, parameter pl\_usePullModeForStats of the init function of ExecCtrl has to be set to true value (default: false).

Accessing statistics data while statHandler separation is active (pl\_nofExternalStatHandlers >0), the following functions can be used:

* f\_EPTF\_ExecCtrl\_StatHandler\_getVarIdx to get the variable index containing the value of the statistics.
* f\_EPTF\_ExecCtrl\_StatHandler\_getVarNameByStatName to get the name of the variable containing the value of the statistics.
* f\_EPTF\_ExecCtrl\_StatHandler\_getAuxVars to get the names of the auxiliary variables for the statistics.

### Progress information during startup

The ExecCtrl can update the progressbar dataSource in UIHandler during startup with information about its progress. This can be enabled/disabled by an API function. The progress information passed to UIHandler is based on the progressBar dataSources in ExecCtrl which are updated independently from the UIHandler update, even if the UIHandler progress bar update is disabled.

## Command Line Interface

The ExecCtrl has an extension called ExecCtrl\_CLI that initializes the EPTF CommandLineInterface by registering the ExecCtrl commands into it.

The ExecCtrl UIHandler by default initializes the ExecCtrl Command Line Interface. The CLI component used can be specified in one of its argument. By default ‘null’ is used, which means that the ExecCtrl CLI will use the UIHandler component as the CLI. The CLI can be switched off by setting the CLI argument to ‘omit’. If CLI is enabled, the user can connect the Command Line Interface and execute ExecCtrl commands. The CLI ports should be configured in the configuration file in this case.

### List of ExecCtrl commands

This section lists the commands defined by the ExecCtrl CLI. All of these commands can be prefixed with a user defined charstring. This prefix can be set in the init function of ExecCtrl\_CLI or the ExecCtrl\_UIHandler. By default no prefix is used. If there is a chance that any of the ExecCtrl command names may coincide with other command names, it is recommended to use a unique prefix for the ExecCtrl commands (e.g. “exec\_”) or for the other commands. All command names (together with the prefix) should follow the command name rules of the EPTF CLI feature, e.g. they cannot contain spaces. All ExecCtrl command names (with the prefix) are case insensitive. More detailed description of a given ExecCtrl command is available on the CLI terminal by issuing “<commandName> help”.

#### startExec

The startExec command can be used to start the execution of a scenario,scenario group, traffic case or the whole test.

#### stopExec

The stopExec command can be used to stop the execution of a scenario,scenario group, traffic case or the whole test. When the scenario group is stopped, phase actions are executed and next phase may start.

#### terminateExec

The terminateExec command can be used to terminate the execution of a scenario group or the whole test. When the scenario group is terminated, the state of the current phase will be changed to idle, no actions are executed, all scenarios in it will be stopped.

#### setCPS

This command can be used to set the target CPS for a weighted scenario or for a traffic case in a non-weighted scenario.

#### getTargetCPS

This command returns the target CPS value for a weighted scenario or for a traffic case in a non-weighted scenario.

#### getCurrentCPS

This command returns the current CPS value for a weighted scenario or for any traffic case.

#### getWeight

This command returns the weight value of a traffic case in a weighted scenario.

#### setWeight

This command can be used to set the weight value of a traffic case in a weighted scenario.

#### getTotalCounter

This command can be used to get the value of a given accumulated statistics. The statistics can be specified by one of the following literals:

“Starts”, “Success”, “Fail”, “Timeout”, “Error”. The result of the command is an integer number, corresponding to the specified statistics. (This is the sum of all the corresponding traffic case statistics.)

**Example:** to determine the sum of failed traffic case executions, you can use the following CLI command:

TTCN> getTotalCounter Fail

120

TTCN>

#### exitTTCN

This command can be used to terminate test execution, generate the final test report and exit from the application. The command behaves as if the ‘Exit’ button was pressed on the ExecCtrl GUI.

# Functional Interface

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

## Naming Conventions

All data types have the prefix EPTF\_ExecCtrl\_ , all functions have the prefix f\_EPTF\_ExecCtrl\_ or f\_EPTF\_ExecCtrlClient\_.

## Public Functions

When using the ExecCtrl feature the listed public functions are provided as an interface. No component variables, timers, ports, or other functions are recommended for use. Using them may result in erroneous behaviour. Also the backward compatibility of these items is not guaranteed.

### Initialization

Before using the EPTF ExecCtrl functions the

f\_EPTF\_ExecCtrl\_init\_CT(…)

function should be called. This initializes the EPTF ExecCtrl feature.

The Execution Control clients should call

f\_EPTF\_ExecCtrlClient\_init\_CT(…)

respectively. It is important that the ExecCtrlClient initialization should be invoked last, only after all other LGenBase components extensions initialization. To avoid data inconsistency, the user should not declare entity groups, scenarios, traffic case types directly in the application init (using the related LGenBase API functions) when using ExecCtrl, since those steps will be done by ExecCtrlClient.

The ExecCtrl\_CLI can be initialized by the function

f\_EPTF\_ExecCtrl\_CLI\_init\_CT(…)

It calls the f\_EPTF\_ExecCtrl\_init\_CT function also.

Before using the ExecCtrl\_UIHandler feature the

f\_EPTF\_ExecCtrl\_UIHandler\_init(…)

function should be called. This initializes the EPTF ExecCtrl UI Handler feature. It calls the f\_EPTF\_ExecCtrl\_CLI\_init\_CT(…) also. If the CLI component reference is not given explicitly, the UIHandler component is used as the CLI component.

### Loading the configuration parameters

The configuration parameters are defined by the ExecCtrl module parameters. They are loaded automatically unless disabled in the ExecCtrl init function. It is possible to use additional module parameters or variables to configure the ExecCtrl. To load these parameters into ExecCtrl the function

f\_EPTF\_ExecCtrl\_loadConfig

can be used. This function can be called more than once. Its arguments are appended to the existing ExecCtrl configuration database. Note, that in order to work properly, the ExecCtrl component should know about all possible scenarios, traffic cases etc. used in the system before ExecCtrl is started, so they all should be loaded into ExecCtrl either by its module parameters or by using this function.

This function should be called before the start function of the ExecCtrl feature is called, because configuration parameters loaded after the start function are invisible for ExecCtrl. Since the start function is called automatically in the ExecCtrl init function if not disabled, calling of start function should be disabled in the init function to be able to load configuration after the init function is called.

The loadConfig function can also be called before the init function of the ExecCtrl feature.

Tip: It you need to load data corresponding to only one of the arguments of the function, the following notation can simplify the code since all arguments of this function are optional:

f\_EPTF\_ExecCtrl\_loadConfig(pl\_EPTF\_ExecCtrl\_LGenFunction\_Entry\_List := vl\_lgenCreatorFunctions);

It is also possible to load configuration data into ExecCtrl from the ExecCtrl client. This can be done by using the following function:

f\_EPTF\_ExecCtrlClient\_loadConfig

This function works similarly and has similar parameters as the f\_EPTF\_ExecCtrl\_loadConfig function. The configuration data can be loaded from the ExecCtrlClient before the f\_EPTF\_ExecCtrlClient\_readyToStart() function is called. Since this function is called in the f\_EPTF\_ExecCtrlClient\_init\_CT by default, this has to be disabled in the init function. It is not possible to load configuration data from ExecCtrlClients that belong to LGen-pools. These LGens are created by ExecCtrl after the ExecCtrl was started and the configuration data was already loaded. Configuration can be loaded from LGens that are created ‘manually’.

### Enabling progress updates in UIHandler

The progressBar dataSource in UIHandler can be updated with ExecCtrl progress information. It is enabled by default. It can be disabled/enabled by the function

f\_EPTF\_ExeCtrl\_enableUIHandlerProgress(…)

### Starting the operation of ExecCtrl

To start the operation of ExecCtrl, the function

f\_EPTF\_ExecCtrl\_start(…)

has to be called. Configuration data can be loaded into ExecCtrl before this function is called. This function is called automatically by the ExecCtrl init function if not disabled.

### Terminate ExecCtrl

To finish the test, generate a final test report and then terminate execution the function

f\_EPTF\_ExecCtrl\_exit()

has to be called.

## Summary Table of all public functions for EPTF ExecCtrl

Table 1. Summary of ExecCtrl functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrl\_init\_CT | Initializes the ExecCtrl Component |
| f\_EPTF\_ExecCtrl\_checkManualLGenStarted | True if all manual LGens have started, false if not. |
| f\_EPTF\_ExecCtrl\_start | Starts the execution of the ExecCtrl |
| f\_EPTF\_ExecCtrl\_getManualControl | Gets the manual control mode |
| f\_EPTF\_ExecCtrl\_setManualControl | Sets the manual control mode (overwrites config setting) |
| f\_EPTF\_ExecCtrl\_getTimePeriodForTcDeltaStats | Gets The time period when delta stats are refreshed |
| f\_EPTF\_ExecCtrl\_setTimePeriodForTcDeltaStats | Tells that the ExecCtrl's Time period when delta stats are refreshed |
| f\_EPTF\_ExecCtrl\_loadConfig | Loads all modulepar-type parameter into the ExecCtrl configuration database |
| f\_EPTF\_ExecCtrl\_setTimeFormat | Sets the timeformat for the timeElapsed variable |
| f\_EPTF\_ExeCtrl\_enableUIHandlerProgress | Enables/disables update of UIHandler progressBar dataSource |
| f\_EPTF\_ExecCtrl\_numEntities | Returns the total number of entities |
| f\_EPTF\_ExecCtrl\_eGrp\_lgenList | Returns the LGen list data for an entity group |
| f\_EPTF\_ExecCtrl\_eGrp\_scenarios | Returns the scenarios of entity group in the instance DB |
| f\_EPTF\_ExecCtrl\_dumpTypeDB | Write the type DB into the log file for debug purposes |
| f\_EPTF\_ExecCtrl\_eGrp\_name | Returns the name of entity group in the instance DB |
| f\_EPTF\_ExecCtrl\_getActiveEntities | Returns the number of active entities for an entity group |
| f\_EPTF\_ExecCtrl\_setActiveEntities | Sets the number of active entities for an entity group |
| f\_EPTF\_ExecCtrl\_checkReadyToRun | Returns true if ExecCtrl is ready to run |
| f\_EPTF\_ExecCtrl\_waitForCondition | General function to block execution until a specific condition becomes true |
| f\_EPTF\_ExecCtrl\_getEntityGroupIdx | Returns the id of an entity group in the instance DB |
| f\_EPTF\_ExecCtrl\_getAllocatedEntities | Returns the total number of allocated entities on all LGens for a given Entity Group |
| f\_EPTF\_ExecCtrl\_getTrafficCaseIdx | Returns the ID of a traffic case |
| f\_EPTF\_ExecCtrl\_getTcOfScenarioIdx | Returns the ID of a traffic case inside a scenario |
| f\_EPTF\_ExecCtrl\_getScenarioIdxForTc | Returns the ID of the scenario instance the traffic case belongs to |
| f\_EPTF\_ExecCtrl\_getTrafficCaseName | Returns the name of the traffic case instance for the given traffic case instance index |
| f\_EPTF\_ExecCtrl\_registerOnGroupFinishCallbackFn | Register function to be called when onGroupFinish condition changes |
| f\_EPTF\_ExecCtrl\_deregisterOnGroupFinishCallbackFn | Deregister callback function for onGroupFinish condition changes |
| f\_EPTF\_ExecCtrl\_checkOnGroupFinished | Returns true if any onGroupFinish condition is fulfilled |
| f\_EPTF\_ExecCtrl\_getScenarioIdx | Returns the index of the scenario in the instance DB |
| f\_EPTF\_ExecCtrl\_getScenarioIdxByInstanceName | Returns the index of the scenario in the instance DB, using the scenario instance name |
| f\_EPTF\_ExecCtrl\_getScenarioName | returns the scenario type name for a scenario instance |
| f\_EPTF\_ExecCtrl\_getScenarioInstanceName | Returns the name of the scenario instance for the given scenario instance index |
| f\_EPTF\_ExecCtrl\_isWeightedScenario | Returns true if the scenario instance in weighted scenario |
| f\_EPTF\_ExecCtrl\_numTcOfScenario | returns the number of traffic cases in the scenario |
| f\_EPTF\_ExecCtrl\_getAvailableEntitiesOnLGenForEGrp | Returns the available entities on an LGen for a given entity group |
| f\_EPTF\_ExecCtrl\_dumpInstanceDB | Write the intance DB into the log file for debug purposes |
| f\_EPTF\_ExecCtrl\_getLGenIdxInEGrpForLGenAndEgrp,  f\_EPTF\_ExecCtrl\_getLGenNumForTc,  f\_EPTF\_ExecCtrl\_getLGenIdx,  f\_EPTF\_ExecCtrl\_getLGenName,  f\_EPTF\_ExecCtrl\_numLGens,  f\_EPTF\_ExecCtrl\_getLGenNumEGroups,  f\_EPTF\_ExecCtrl\_getLGenEGrpIdx | Useful get functions for accessing LGen data |
| f\_EPTF\_ExecCtrl\_startTimeProfiles,  f\_EPTF\_ExecCtrl\_stopTimeProfiles | Start/stop time profiles |
| f\_EPTF\_ExecCtrl\_startAllScenarios,  f\_EPTF\_ExecCtrl\_stopAllScenarios,  f\_EPTF\_ExecCtrl\_terminateAllScenarios | Start/stop/terminate all scenarios, including scenario groups |
| f\_EPTF\_ExecCtrl\_startAllScenarioGroups,  f\_EPTF\_ExecCtrl\_stopAllScenarioGroups,  f\_EPTF\_ExecCtrl\_terminateAllScenarioGroups | Start/stop/terminate scenario groups |
| f\_EPTF\_ExecCtrl\_stopCurrentPhaseForAllScenarioGroups | Stop the current phase in all scenario group (and continue with the next phase) |
| f\_EPTF\_ExecCtrl\_setCps\_TC,  f\_EPTF\_ExecCtrl\_setCps\_SC | Set the CPS of a traffic case/scenario and send it to the clients |
| f\_EPTF\_ExecCtrl\_setStartDelay\_TC | sets the Traffic case startDelay in all related LGens |
| f\_EPTF\_ExecCtrl\_getCPSToReach\_TC,  f\_EPTF\_ExecCtrl\_setCPSToReach\_TC,  f\_EPTF\_ExecCtrl\_getCPSToReach\_SC,  f\_EPTF\_ExecCtrl\_setCPSToReach\_SC | Get/set the target CPS (cps to reach) for a traffic case or a scenario |
| f\_EPTF\_ExecCtrl\_getTCWeight,  f\_EPTF\_ExecCtrl\_setTCWeight | Get/set the weight of a traffic case inside a weighted scenario |
| f\_EPTF\_ExecCtrl\_registerCPSChangedCallback\_TC,  f\_EPTF\_ExecCtrl\_deregisterCPSChangedCallback\_TC,  f\_EPTF\_ExecCtrl\_registerCPSChangedCallback\_SC,  f\_EPTF\_ExecCtrl\_deregisterCPSChangedCallback\_SC | Register or deregister call-back functions that are called when the CPS of the given Traffic Case or the Scenario changes |
| f\_EPTF\_ExecCtrl\_Regulator\_addRegulator | Add a regulator |
| f\_EPTF\_ExecCtrl\_Regulator\_getRegulatorId | Get the regulator ID |
| f\_EPTF\_ExecCtrl\_Regulator\_setTotalCps,  f\_EPTF\_ExecCtrl\_Regulator\_getTotalCps | Set/Get total CPS to the regulator (the target value) |
| f\_EPTF\_ExecCtrl\_Regulator\_getRegulatedItemId | Determine the ID of a regulated item |
| f\_EPTF\_ExecCtrl\_Regulator\_getRegulatorName,  f\_EPTF\_ExecCtrl\_Regulator\_setRegulatorName | Get/set the name of a regulator for a regulated item |
| f\_EPTF\_ExecCtrl\_Regulator\_logAll | Log everything connected to regulators for debug purposes |
| f\_EPTF\_ExecCtrl\_Regulator\_findRegulatorsForScenario | find the regulators that regulate items in the current scenario |
| f\_EPTF\_ExecCtrl\_sendUpdatedCps,  f\_EPTF\_ExecCtrl\_sendUpdatedScenarioCps | Send traffic case/scenario CPS to clients |
| f\_EPTF\_ExecCtrl\_registerScenarioGroupPhaseChangedCallback | Function to register a callback function for the scenarioGroup phase change event |
| f\_EPTF\_ExecCtrl\_ScenarioGroup\_get\_byScIndex | Returns a scenario group instance idx of a scenario instance |
| f\_EPTF\_ExecCtrl\_ScenarioGroup\_getIdx | Returns the index of a scenario group instance using its name |
| f\_EPTF\_ExecCtrl\_ScenarioGroup\_start | Starts the actual phase of the scenario group |
| f\_EPTF\_ExecCtrl\_ScenarioGroup\_stop | Stops the actual phase of the scenario group |
| f\_EPTF\_ExecCtrl\_getScenarioGroupInstanceName | Returns the scenario group instance name |
| f\_EPTF\_ExecCtrl\_scenarioIsNotInScenarioGroup | Checks if a scenario belongs to a scenario group |
| f\_EPTF\_ExecCtrl\_registerScenarioStateChangedCallback | Function to register a callback function for the scenario state change event |
| f\_EPTF\_ExecCtrl\_registerTrafficCaseStateChangedCallback | Function to register a callback function for the traffic case state change event |
| f\_EPTF\_ExecCtrl\_enableStartStopScenario,  f\_EPTF\_ExecCtrl\_disableStartStopScenario | Enable/disable start/stop buttons and associated variables for the GUI |
| f\_EPTF\_ExecCtrl\_isDisabledStartStopScenario | Check if the start/stop button is disabled or not for the scenario |
| f\_EPTF\_ExecCtrl\_getStartStopScenarioIdx | Returns the ID of the database item associated with the start/stop scenario buttons |
| f\_EPTF\_ExecCtrl\_enableStartStopTC,  f\_EPTF\_ExecCtrl\_disableStartStopTC | Enable/disable start/stop buttons for traffic case |
| f\_EPTF\_ExecCtrl\_isDisabledStartStopTC | Check if the start/stop button is disabled or not for the traffic case |
| f\_EPTF\_ExecCtrl\_getStartStopTCIdx | Returns the ID of the database item associated with the start/stop traffic case buttons |
| f\_EPTF\_ExecCtrl\_startScenarioOnLGensByName,  f\_EPTF\_ExecCtrl\_stopScenarioOnLGensByName | Start/stop a scenario on ExecCtrlClient/LGen |
| f\_EPTF\_ExecCtrl\_startScenarioOnLGens,  f\_EPTF\_ExecCtrl\_stopScenarioOnLGens | Start/stop scenario on LGens |
| f\_EPTF\_ExecCtrl\_startTCOnLGens,  f\_EPTF\_ExecCtrl\_stopTCOnLGens | Start/stop traffic case on LGens |
| f\_EPTF\_ExecCtrl\_singleShotTc | Start traffic case for one entity on one of the LGens (single shot) |
| f\_EPTF\_ExecCtrl\_resetScenarioOnLGens | Reset scenario on LGens |
| f\_EPTF\_ExecCtrl\_stopScenarioGroupOnLGens | Stops the execution of the scenario group on LGens, next phase is not started |
| f\_EPTF\_ExecCtrl\_scenarioEnabled | Check if the scenario is enabled |
| f\_EPTF\_ExecCtrl\_LGenPool\_createLGens | Creates the LGens for all LGen pools |
| f\_EPTF\_ExecCtrl\_nrOfClients | Returns the number of ExecCtrlClient components |
| f\_EPTF\_ExecCtrl\_LGenPool\_dumpInstanceDB | Writes the LGenPool intance DB into the log file for debug purposes |
| f\_EPTF\_ExecCtrl\_addLGenFunctionsFromDeclarators | Register the LGen creator functions to create LGen pools |
| f\_EPTF\_ExecCtrl\_getLGenCompRefsOfPool | Get a list of LGen component indexes for a given LGen pool name |
| f\_EPTF\_ExecCtrl\_checkCreatorFnNameOfPool | Checks if the given LGen creator function was used to create LGens in the given pool |
| f\_EPTF\_ExecCtrl\_registerFinalTestReportGeneratorFn | Registers final test report generator function |
| f\_EPTF\_ExecCtrl\_deregisterFinalTestReportGeneratorFn | Deregisters final test report generator function |
| f\_EPTF\_ExecCtrl\_generateFinalTestReport | Generates the final test report |
| f\_EPTF\_ExecCtrl\_registerFSMStatsCreatedCallback | Registers callback function to call when FSM statistics created for traffic cases |
| f\_EPTF\_ExecCtrl\_deregisterFSMStatsCreatedCallback | Removes the FSM statistics created callback function |
| f\_EPTF\_ExecCtrl\_getFSMStatisticsOfTcs | Returns all FSM statistics declared for all traffic cases |
| f\_EPTF\_ExecCtrl\_getNrOfExpectedClients | Gets the number of Expected Clients set previously by f\_EPTF\_ExecCtrl\_setNrOfExpectedClients() |
| f\_EPTF\_ExecCtrl\_setNrOfExpectedClients | Sets the number of Expected Clients - this number is used during displaying the progress of the initialisation of the ExecCtrl lgenpool section. |
| f\_EPTF\_ExecCtrl\_behavior | Example behavior of EPTF\_ExecCtrl feature |
| f\_EPTF\_ExecCtrl\_exit | Terminate test execution and generate final test riport |
| f\_EPTF\_ExecCtrl\_setGenerateOwnReport | Tells that the ExecCtrl's own final report has to be generated or not |
| f\_EPTF\_ExecCtrl\_StatHandler\_getVarIdx | Returns the variable index for a statistics declared by ExecCtrl |
| f\_EPTF\_ExecCtrl\_StatHandler\_getVarNameByStatName | Returns the name of the variable for a statistics declared by ExecCtrl |
| f\_EPTF\_ExecCtrl\_StatHandler\_getAuxVars | Returns the names of auxiliary variables for a statistics declared by ExecCtrl |
| f\_EPTF\_ExecCtrl\_resetFSMStat | Resets a given FSM Statistics of a Traffic Case |

### 

### Functions that were removed or became private in the R3 version of ExecCtrl

Table 2. Summary of ExecCtrl functions that are no longer public or removed

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrl\_initHashMaps | Initializes hash tables used in ExecCtrl |
| f\_EPTF\_ExecCtrl\_cleanup\_CT | Cleanup function of ExecCtrl |
| f\_EPTF\_ExecCtrl\_sendCreateGrp | Initiates entity group creation on load generators |
| f\_EPTF\_ExecCtrl\_resetGrp | Initiates entity group deletion on load generators |
| f\_EPTF\_ExecCtrl\_splitTargetCps4groups | On-demand decomposition of targetCps for Load Generators |
| f\_EPTF\_ExecCtrl\_sendUpdatedCps | Sends updated Cps values for Load Generators |
| f\_EPTF\_ExecCtrl\_declareTrafficCasesOnClients | Declares traffic case types on ExecCtrlClients/LGens from v\_ExecCtrl\_tcTypeDeclarators |
| f\_EPTF\_ExecCtrl\_declareSceanriosOnClients | Declares traffic scenarios types on ExecCtrlClients/LGens |
| f\_EPTF\_ExecCtrl\_addScenarioToEntityGroup | Assigns a scenario to entity group on ExecCtrlClient/LGen |
| f\_EPTF\_ExecCtrl\_startScenarioOnLGens(ByName) | Starts a traffic scenario on ExecCtrlClient/LGen |
| f\_EPTF\_ExecCtrl\_stopScenarioOnLGens(ByName) | Stops a traffic scenario on ExecCtrlClient/LGen |
| as\_EPTF\_ExecCtrl\_MgmtIf | Main event handler of ExecCtrl |
| f\_EPTF\_ExecCtrl\_updateExecStatusDB | Updates execution status DB according the last received client message |
| f\_EPTF\_ExecCtrl\_checkScenarioStatus | Check if the execution state of a specified scenario is consistent in all LGens |

## Summary Table of all public functions for EPTF ExecCtrl UI Handler

Table 3. Summary of ExecCtrl\_UIHandler functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrl\_UIHandler\_init\_CT | Initializes the ExecCtrl UI Handler Component |
| f\_EPTF\_ExecCtrl\_UIHandler\_setRefreshRates | Set custom refresh period for different elements on the GUI |
| f\_EPTF\_ExecCtrl\_UIHandler\_checkGuiDone | Returns true when the creation of the GUI is finished |
| f\_EPTF\_ExecCtrl\_UIHandler\_behavior | Example behavior of EPTF\_ExecCtrl\_UIHandler feature |

## Summary Table of all public functions for EPTF ExecCtrl CLI

Table 3. Summary of ExecCtrl\_CLI functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrl\_CLI\_init\_CT | Initializes the ExecCtrl CLI Component |

## Execution Control Client

ExecCtrlClient with interaction of ExecCtrl provides traffic case execution and traffic case provisioning support along multiple Load Generators. In general, ExecCtrl initiates actions via internal management messages on multiple Load Generators, ExecCtrlClients processing the requests using LGenBase public API functions.

During/after the initialization:

* ExecCtrl Client can load configuration data to ExecCtrl
* ExecCtrlClient reports the available resources (list of maximum available amount of different entity types)
* ExecCtrl initiates entity group creations via internal management messages
* ExecCtrlClient creates entity groups and reports the new resource allocation to ExecCtrl.
* ExecCtrlClient creates and initializes traffic cases, traffic scenarios according the decomposed descriptors and sends initial execution status message to ExecCtrl
* EndOfConfig message arrives from ExecCtrl when the configuration is finished
* ExecCtrlClient calls registered callback functions when configuration ends
* ExecCtrlClient notifies ExecCtrl that execution of traffic cases can begin
* ExecCtrlClient notifies ExecCtrl when the GUI is ready. This can be done automatically or by calling the function f\_EPTF\_ExecCtrlClient\_reportGUIDone.

Test scenario execution:

* ExecCtrl initiates traffic scenario starts on Load Generators
* ExecCtrlClient starts scenario and reports its execution status back to ExecCtrl whenever it changes.

## Summary Table of all public functions for EPTF ExecCtrlClient

Table 4. Summary of ExecCtrlClient public functions

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrlClient\_init\_CT | Initializes the ExecCtrlClient Component |
| f\_EPTF\_ExecCtrlClient\_loadConfig | Loads configuration data to ExecCtrl configuration database |
| f\_EPTF\_ExecCtrlClient\_registerEndOfConfigCallback | Register callback function that is called when the configuration of the client is finished |
| f\_EPTF\_ExecCtrlClient\_readyToStart | Notifies ExecCtrl that execution of traffic cases can begin |
| f\_EPTF\_ExecCtrlClient\_registerFinalTestReportGeneratorFn | Registers final test report generator function |
| f\_EPTF\_ExecCtrlClient\_deregisterFinalTestReportGeneratorFn | Deregisters final test report generator function |
| f\_EPTF\_ExecCtrlClient\_checkEndOfConfig | Returns true when the configuration of the ExecCtrlClient is finished |
| f\_EPTF\_ExecCtrlClient\_registerGuiDoneCallback | Register callback function which is called when the ExecCtrl UIHandler finished the GUI creation |
| f\_EPTF\_ExecCtrlClient\_checkGuiDone | Returns true when the ExecCtrl GUI is ready |
| f\_EPTF\_ExecCtrlClient\_reportGUIDone | Reports the GUIDone event to ExecCtrl |
| f\_EPTF\_ExecCtrlClient\_checkGuiDoneReported | Returns true when the ExecCtrlClient reported the GUIDone event to ExecCtrl |
| f\_EPTF\_ExecCtrlClient\_getFirstEntityGlobalOffset | Returns the global offset of the first entity of the entity group deployed on the LGen |
| f\_EPTF\_ExecCtrlClient\_getEntityGroupGlobalOffset | Returns the global offset of the first entity of the entity group |
| f\_EPTF\_ExecCtrlClient\_getFirstEntityOffsetInEGrp | Returns the relative offset of the first entity of the entity group deployed on the LGen in the entity group |
| f\_EPTF\_ExecCtrlClient\_getEntityGroupSize | Returns the size of the of the entity group |

## Summary Table of functions for EPTF ExecCtrlClient that are no longer public in R3

Table 5. Summary of ExecCtrlClient functions that no longer public or removed

| Function name | Description |
| --- | --- |
| f\_EPTF\_ExecCtrlClient\_cleanup\_CT | Cleanup function of ExecCtrlClient Component |
| f\_EPTF\_ExecCtrlClient\_sendAvailableResourceList | Function for sending the LGen's resource availability and allocation info to ExecCtrl |
| f\_EPTF\_ExecCtrlClient\_sendExecStatus | Reports execution status changes to ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_createGrp | Creates entity groups on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_deleteGrp | Removes entity groups on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_UpdateCps | Updates targetCps values on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_DeclareScenario | Declares traffic scenarios on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_AddScenarioToEntityGroup | Assigns entity groups to traffic scenarios on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_DeclareTc | Declares traffic case types on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_StartScenario | Starts traffic scenario on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf\_StopScenario | Stops traffic scenario on a Load Generator upon a request from ExecCtrl |
| as\_EPTF\_ExecCtrlClient\_MgmtIf | Main event handler of ExecCtrlClient feature |

## Summary Table of all public functions for EPTF ExecCtrlClient UIHandler

Table 6. Summary of ExecCtrlClient\_UIHandler public functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_ExecCtrlClient\_UIHandler\_init\_CT | Initializes the ExecCtrlClient\_UIHandler Component |
| f\_EPTF\_ExecCtrlClient\_UIHandler\_setRefreshRates | Set custom refresh rates for certain elements on the GUI |

## Public Variable names in ExecCtrl UIHandler

This section lists the variable names created by ExecCtrl\_UIHandler with the help of EPTF\_Var feature that are publicly available and can be used to access GUI functions for example from the CommandLine Interface of the UIHandler feature. Variables listed here are shown also in the GUI created by ExecCtrl\_UIHandler. All variables are on the UIHandler component, and all variables have the prefix: “GUI.” However, the following variables are on the ExecCtrl\_UIHandler component, and have no “GUI.” prefix:

* Weighted scenario current CPS
* All Traffic case statistics

In Table 7 the following TTCN-3 style notation is used (& is the concatenation operator):

* c\_EPTF\_ExecCtrl\_statisticsRoot: “EPTF\_ExecCtrl.Statistics”
* vl\_eGrpName: Name of the entity group
* vl\_scTypeName: name of the scenario type
* vl\_tcName: name of the traffic case
* vl\_scGrpName: scenario group name
* c\_EPTF\_ExecCtrl\_tcStatNames[st]: name of the statistics at index st
* c\_EPTF\_ExecCtrl\_tcDeltaStatVarIds[idx] indexes of delta statistics for c\_EPTF\_ExecCtrl\_tcStatNames statistics at index idx

Table 7. Summary of public variable names in ExecCtrl UIHandler

| Variable name | Description |
| --- | --- |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".nofLGens" | Number of LGens the entity group was deployed on |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".StatusLED" | Scenario StatusLED |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TargetCPS" | Weighted scenario TargetCPS |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TargetCPSRegulator.Name" | Name of the regulator that regulates the weighted scenario TargetCPS |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".SCGrp."&vl\_scGrpName&".StartPhase" | StartPhase of the Scenario group |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".SCGrp."&vl\_scGrpName&".Mode" | GroupMode of the scenario group |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".SCGrp."&vl\_scGrpName&".GroupStatusLED" | Scenario group StatusLED |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".SCGrp."&vl\_scGrpName&".GroupStart" | Scenario group Start button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".SCGrp."&vl\_scGrpName&".GroupReset" | Scenario group reset button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".ControlButtons.Start" | Scenario start button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".ControlButtons.Stop" | Scenario Stop button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".ControlButtons.Reset" | Scenario Reset button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".GroupFinishStatusLED" | GroupFinish StatusLED for the traffic case |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".StatusLED" | Traffic case StatusLED |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".Start" | Traffic case start checkbox |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".Stop" | Traffic case stop checkbox |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".SingleShotTc.EIdx | The entity index inside the entity group used by single shot |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".SingleShotTc.EnableLog" | The enable log checkbox for single shot traffic |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".SingleShotTc.Start" | The single shot start button |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".TargetCPSOrWeight" | Traffic case target CPS if scenario is non-weighted, traffic case weight if weighted |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".TargetCPSRegulator.Name" | Regulator of the TargetCPS of a traffic case |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName &".CurrentCPS" | Weighted scenario current CPS  (No “GUI.” prefix!) |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName &"."&c\_EPTF\_ExecCtrl\_tcStatNames[st] | Traffic case statistics for each statistics in c\_EPTF\_ExecCtrl\_tcStatNames  (No “GUI.” prefix!) |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName &"."&c\_EPTF\_ExecCtrl\_tcStatNames[c\_EPTF\_ExecCtrl\_tcDeltaStatVarIds[idx]]&”.delta” | Traffic case delta statistics for each statistics which indices in c\_EPTF\_ExecCtrl\_tcDeltaStatVarIds table  (delta statistics the last changed delta value in a given time period) |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&”.GroupFinishCondStatusLED."&c\_EPTF\_ExecCtrl\_tcStatNames[st] | The Status LED for onGroupFinish condition status |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".eventStatus.EndOfConfig.StatusLED" | The Status LED for the EndOfConfig event |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".eventStatus.ReadyToRun.StatusLED" | The Status LED for the ReadyToRun event |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".eventStatus.GUIDone.StatusLED" | The Status LED for the GUIDone event |
| c\_EPTF\_ExecCtrl\_statisticsRoot&".eventStatus.EndOfTest.StatusLED" | The Status LED for the EndOfTest event |

## Public widget IDs on the ExecCtrl UIHandler GUI

The following widget Id can be used to put user defined custom information to the traffic cases information on the **Statistics** panel under **Custom stats** tab: c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".userSpecificTcData".

# 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 113 200/6 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)
5. 7/155 16-CNL 113 512  
   EPTF Core Library LGenBase, Function Description