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

EPTF Core Library Execution Control, User Guide

Contents

[1 Introduction 3](#_Toc485080581)

[1.1 Revision history 3](#_Toc485080582)

[1.2 About this Document 3](#_Toc485080583)

[1.2.1 How to Read this Document 3](#_Toc485080584)

[1.2.2 Abbreviations 4](#_Toc485080585)

[1.2.3 Terminology 4](#_Toc485080586)

[1.3 System Requirements 5](#_Toc485080587)

[2 Execution Control 5](#_Toc485080588)

[2.1 Overview 5](#_Toc485080589)

[2.2 Description of files in this feature 6](#_Toc485080590)

[2.3 Description of required features from TCC 7](#_Toc485080591)

[2.4 Installation 8](#_Toc485080592)

[2.5 Configuration 8](#_Toc485080593)

[2.5.1 Control parameters 9](#_Toc485080594)

[2.5.2 Scenarios and entity groups 10](#_Toc485080595)

[2.5.3 Phases 11](#_Toc485080596)

[2.5.4 Regulator parameters 12](#_Toc485080597)

[2.5.5 Time profile parameters 12](#_Toc485080598)

[2.5.6 LGen pool parameters 13](#_Toc485080599)

[2.5.7 Parameter ranges 13](#_Toc485080600)

[2.5.8 Parameter decomposition 14](#_Toc485080601)

[2.5.9 Obsolete parameters 14](#_Toc485080602)

[2.5.10 Traffic case custom parameters 15](#_Toc485080603)

[2.5.11 Customizable GUI 16](#_Toc485080604)

[3 Execution Control UI Handler 17](#_Toc485080605)

[3.1 Configuration 17](#_Toc485080606)

[3.1.1 Refresh rates 18](#_Toc485080607)

[3.1.2 Setting custom refresh rates for the GUI elements 19](#_Toc485080608)

[3.2 Using the GUI 20](#_Toc485080609)

[3.2.1 Statistics tab 21](#_Toc485080610)

[3.2.2 Entity Groups tab 30](#_Toc485080611)

[3.2.3 Client Resources tab 30](#_Toc485080612)

[3.3 Traffic Cases panel 32](#_Toc485080613)

[3.4 Regulator panel 33](#_Toc485080614)

[3.5 Phase Lists panel 34](#_Toc485080615)

[4 Execution Control Client 35](#_Toc485080616)

[4.1 Configuration 35](#_Toc485080617)

[5 Execution Control Client UI Handler 36](#_Toc485080618)

[5.1 Configuration 37](#_Toc485080619)

[6 Usage 37](#_Toc485080620)

[6.1 Using ExecCtrl 37](#_Toc485080621)

[6.2 Events in ExecCtrl 38](#_Toc485080622)

[6.3 Using LGen Pools 40](#_Toc485080623)

[6.4 Limited Traffic Execution 41](#_Toc485080624)

[7 Multiple Execution Phase Support 42](#_Toc485080625)

[7.1 Function specification 42](#_Toc485080626)

[7.2 Overview 43](#_Toc485080627)

[7.3 Declarations 43](#_Toc485080628)

[7.3.1 Phase Lists 43](#_Toc485080629)

[7.3.2 Scenario Groups 44](#_Toc485080630)

[7.4 Databases 44](#_Toc485080631)

[7.5 Execution 46](#_Toc485080632)

[7.6 ExecCtrl/ExecCtrlClient Phase PDUs 46](#_Toc485080633)

[7.6.1 Phase start request 46](#_Toc485080634)

[7.6.2 Phase stop request 47](#_Toc485080635)

[7.6.3 Phase stop test request 47](#_Toc485080636)

[7.6.4 Phase status response 47](#_Toc485080637)

[7.7 GUI for ScenarioGroups and PhaseLists 48](#_Toc485080638)

[8 Error messages 49](#_Toc485080639)

[9 Warning messages 49](#_Toc485080640)

[10 Examples 49](#_Toc485080641)

[10.1 Configuration file 50](#_Toc485080642)

[10.2 Demo Module 50](#_Toc485080643)

[11 References 51](#_Toc485080644)

# 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 | UI handler added | ETHJGI |
| 2008-01-31 | PA4 | Updated after review | ETHJGI |
| 2008-03-17 | PA5 | Time profile/Regulator added | ETHJGI |
| 2009-02-17 | PC1 | R3 update | ETHJGI |
| 2009-04-23 | PC2 | R3 update cont. | ETHJGI |
| 2009-05-25 | PC3 | R3 update cont. | ETHJGI |
| 2009-06-08 | PC4 | New distr. chart modulepars added | ETHJGI |
| 2009-10-22 | PD1 | Phase dependent regulation added | ETHJGI |
| 2010-01-13 | PD2 | StatusVars added,new events added | ETHJGI |
| 2010-06-04 | PE1 | Module par loading, text corrections, custom entity group distribution, start function, terminate test button added, load config from client added | ETHJGI |
| 2010-07-02 | PF1 | Custom refresh rates added | ETHJGI |
| 2011-01-21 | PG1 | Limited traffic execution added | ETHJGI |
| 2011-03-03 | PG2 | Data elements and iterators for custom GUI layout | ETHJGI |
| 2016-05-18 | PH1 | 3.2.1.2 & 3.2.1.3 chapters updated | ESZILSZ |
| 2016-06-09 | H | Updated for release. | ESZILSZ |
| 2016-10-12 | PJ1 | Removed chart functionality restriction | EIMRENA |
| 2016-12-02 | J | Updated for release | EIMRENA |
| 2016-12-14 | PK1 | Added GoS to Entity Stats | EIMRENA |
| 2017-01-20 | PK2 | Updated references | EIMRENA |
| 2017-06-09 | K | Updated for release | EIMRENA |

## About this Document

### How to Read this Document

This is the User Guide 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]. This document should be read together with the Function Description of the ExecCtrl feature [5]. For more information on the EPTF Core Library please consult the Product Revision Information, the Users Guide [4] and the Function Specification [5] of the TitanSim.

### Abbreviations

CLL Core Library

EPTF Ericsson Performance Test Framework

TitanSim New synonym for the EPTF Framework

TCC Test Competence Centre in ETH R&D.

TTCN-3 Testing and Test Control Notation version 3 ‎[1]

### Terminology

*Core Library (CLL)* is that part of the TitanSim software that is totally project independent. (i.e., which is not protocol-, or application-dependent). The Core Library is to be supplied and supported by the TCC organization. Any Core Library development is to be funded centrally by Ericsson.

*ExecCtrl* is control logic for provisioning and executing test *🡪 scenarios* with TitanSim based load generators

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

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

*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.

*Entity type* The type of the entity. It defines what kind of behavior types the entity supports, what kind of functionality the entity is capable of. The entity type of the *🡪 entity group* on which the *🡪 scenario* is deployed, and the entity type of the *🡪 traffic case*s in the scenario, has to be the same.

*LGen* Load Generator: a component providing an implementation for final state machine handling and traffic case management in a load test environment. ExecCtrl manages LGens via its ExecCtrlClient interface.

*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.

*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 traffic case. An entity group may have more than one concurrent traffic cases

## System Requirements

In order to use the ExecCtrl feature the system requirements listed in EPTF Core LibraryUser Guide [4] should be fulfilled.

# Execution Control

## Overview

The EPTF Core Library ExecCtrl in connection with ExecCtrlClient components are fundamental components providing an implementation for traffic case execution control in a TitanSim-based load test environment.   
ExecCtrl provides central configuration and execution control support when using multiple, even different types of Load Generators during parallel test execution.

The ExecCtrl feature is designed to control and distribute the traffic case execution on several LGens. This makes possible to execute the same test using more CPUs. The management and synchronization of the LGen components is the main role of the ExecCtrl feature.

The ExecCtrl component is able to deploy the LGen components that execute the traffic cases automatically. It uses user written LGen creator functions that are registered into ExecCtrl to create the LGens. The LGens can be organized into groups. These groups are called LGen-pools. A scenario, which is basically a list of traffic cases, is executed only on the LGens of a single LGen pool. This makes the configuration easily manageable.

ExecCtrl supports multiple phase execution. This means that several scenarios can be organized into groups with a given phase list. The execution of the scenarios in these scenario groups is repeated for every phase in the phase list in sequential order. The next phase is only started when all scenarios finished their execution in the current phase. This functionality can be used to synchronize the execution of different scenarios, and to make possible phase dependent scenario and traffic case execution.

ExecCtrl can be configured with the help of module parameters. It uses the same module parameters as LGenBase uses to declare scenarios, entity groups and traffic cases, but has its own module parameters also.

## Description of files in this feature

The EPTF Core Library ExecCtrl API includes the following files:

* ExecCtrl and ExecCtrlClient:
  + EPTF\_CLL\_ExecCtrl\_Definitions.ttcn: This TTCN-3 module contains common type definitions that should be used in ExecCtrl and ExecCtrlClient components.
  + EPTF\_CLL\_ExecCtrl\_Functions.ttcn: This TTCN-3 module contains the implementation of Execution Control functions.
  + EPTF\_CLL\_ExecCtrlUIHandler\_Definitions.ttcn: This TTCN-3 module contains common type definitions that should be used in ExecCtrl\_UIHandler components.
  + EPTF\_CLL\_ExecCtrlUIHandler\_Functions.ttcn: This TTCN-3 module contains the implementation of Execution Control UI handler functions.
  + EPTF\_CLL\_ExecCtrlTimeProfile\_Definitions.ttcn: Types needed for TimeProfile functionality
  + EPTF\_CLL\_ExecCtrl\_LoggingFunctions.ttcn: Functions used for logging in ExecCtrl
  + EPTF\_CLL\_ExecCtrl\_PhaseDefinitions.ttcn: Type definitions for phases
  + EPTF\_CLL\_ExecCtrl\_PhaseFunctions.ttcn: Function definitions for using phases
  + EPTF\_CLL\_ExecCtrl\_ScenarioDefinitions.ttcn: Type definitions for scenario, traffic case and entity group databases
  + EPTF\_CLL\_ExecCtrl\_ScenarioFunctions.ttcn: Scenario database handling functions
  + EPTF\_CLL\_ExecCtrlClient\_Functions.ttcn: Functions that used by ExecCtrlClient

## Description of required features from TCC

The ExecCtrl feature is part of the TitanSim EPTF Core Load Library (CLL). It relies on several features of the CLL and also some protocol modules and test ports. To use the ExecCtrl, the user has to obtain the respective files from the following features:

Features from CLL:

* Base
* Common
* LGenBase
* Variable
* HashMap
* UIHandler
* Logging
* RandomNArray
* LoadRegulator
* Scheduler
* RedBlackTree
* FreeBusyQueue
* Semaphore
* StatHandler
* StatMeasure

Protocol modules:

* XSDNASN protocol module
* XTDP protocol module

TCC Useful library:

* DateTime, Encoding, FileIO, Maths, Conversion from TCCUsefulFunctions

Test ports:

* XTDP test port
* AbstractSocket test port
* Telnet test port
* UDP test port
* Socket API

## Installation

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

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

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

## Configuration

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

The ExecCtrl and ExecCtrlClient feature in general relies on TTCN-3 module parameters defined for entity groups, traffic scenarios, traffic case types in LGenBase ‎[9]. ExecCtrl centralizes and pre-processes the handling of such traffic descriptor parameters and makes the configuration and control easier in complex load generator scenarios. Actual values of these parameters – when no default value or a different from the default actual value is to be used – shall be given in the [MODULE\_PARAMETERS] section of the configuration file.

Configuration of ExecCtrl is possible with the help of the loadConfig API function as well. Parameters loaded by this function are appended to existing data. Confiuration data can be loaded by the loadConfig function before ExecCtrl was started. 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.

Configuration data from ExecCtrl clients can be loaded before the client reports readyToStart to ExecCtrl. To be able to load configuration from the ExecCtrlClients, reporting ready to start in the init function has to be disabled. Additionally, the ExecCtrl client has to be created ‘manually’, because LGens belonging to LGen pools are created after the configuration data was loaded into ExecCtrl.

Configuration data is loaded from the module parameters automatically. This can be disabled in the ExecCtrl init function.

The module parameters used by the ExecCtrl feature are described in the following sections.

### Control parameters

These parameters control the behaviour of the ExecCtrl.

**tsp\_ExecCtrl\_ignoreOutOfResource**

This boolean type module parameter is defined in module EPTF\_CLL\_ExecCtrl\_Definitions. It is used to define the behaviour of entity group creation method. When the number of entities to be created in a group exceeds the maximum number of possible entities declared by the Load Generator, and this flag is set to false, the test system stops with configuration error. Default value is: true.

**tsp\_ExecCtrl\_manualControl**

This boolean type module parameter is defined in module EPTF\_CLL\_ExecCtrl\_Definitions. It is used to define the behaviour of the ExecCtrl state machine. When this flag is set to true, the configured scenarios will be started only upon user interaction, via Runtime GUI or CLI. Default value is: false.

**tsp\_EPTF\_ExecCtrl\_StartStopScenarioDisable**

This parameter can be used to specify which Scenario Start/Stop checkboxes should be enabled/disabled at startup. If not specified the Scenario Start/Stop checkboxes are enabled by default, which means they can be started/stopped using the GUI during runtime.

**tsp\_ExecCtrl\_LGenBaseStatsUI\_columnDescriptorList**

This module parameter can be used to enable/disable elements on the ExecCtrl/Clients tab. See LGenBase documentation for details. Default setting is to display all available information with read-only format.

**tsp\_ExecCtrl\_LGenBaseStatsUI\_scColumnDescriptorList**

This module parameter can be used to enable/disable elements on the ExecCtrl/Clients tab. See LGenBase documentation for details. Default setting is to display all available information with read-only format.

**tsp\_EPTF\_ExecCtrl\_finalTestReportFile**

This module parameter specifies the name of the file where the final test report is written. It can also be used to disable final test report generation by setting it to empty string. If enabled (file name specified), the final test report generator functions that were registered in ExecCtrl and the ExecCtrlClients are called, and the report is written (or appended) to the specified file. The file name can also be a format string where the time-stamp can be defined. For allowed formats see the f\_getTimeFormatted function description in TCCDateTime\_Functions. The default value of this module parameter uses this kind of format. Default value: "EPTF\_ExecCtrl\_FinalTestReport\_%Y-%m-%d\_%H.%M.%S.txt".

### Scenarios and entity groups

Some of these parameters are defined in LGenBase, but also used by ExecCtrl.

**tsp\_LGenBase\_EntityGrpDeclarators**

This module parameter is defined in LGenBase. It is used to declare a list of entity groups. The referred entity types should be already declared by application libraries or from user code using f\_EPTF\_LGenBase\_declareEntityType() in the ExecCtrl client.

**tsp\_LGenBase\_TcMgmt\_tcTypeDeclarators2**

This module parameter is defined in LGenBase. It is used to define a list of traffic case types. The traffic case names should be unique. The referred FSM names should be already declared by application libraries or from user code using e.g. f\_EPTF\_LGenBase\_declareFSMTable () in the ExecCtrl client.

Makes the tsp\_LGenBase\_tcTypeDeclarators obsolete.

**tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators3**

This module parameter is defined in LGenBase. It is used to declare a list of traffic scenarios of type EPTF\_LGenBase\_ScenarioTypeDeclaratorList. The referred traffic case type names should be consistent with traffic case declarations. Empty scenarios (scenario with zero nr of traffic cases) are not allowed. This module parameter makes the following scenario declarators obsolete:

* tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators
* tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators2
* tsp\_LGenBase\_TcMgmt\_WeightedScenarioDeclarators

**tsp\_EPTF\_ExecCtrl\_Scenario2EntityGroupList**

This module parameter is used to define a list of entity group – traffic scenario associations. The referred entity group and scenario names should be declared also in tsp\_LGenBase\_EntityGrpDeclarators and tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators3 (or in other scenario declarators). A scenario instance name can also be associated to the scenario deployed on the given entity group. In addition to the given instance name, the following instance name is automatically associated to the scenario instance: <EntityGroupName>+ “.” + <ScenarioName>.

Note, that the entity type of the entity group, and the entity type of the traffic cases in the scenario has to be the same. This is checked at startup, and an error is generated if the entity types do not match.

This module parameter makes the following declarator obsolete:

* tsp\_LGenBase\_Scenarios2Grps

It is allowed to associate more than one scenario to a single entity group. This means that these scenarios can be executed in parallel on the same entity group.

Also it is allowed to assign the same scenario to different entity groups. This means that the same scenario can be executed on different entity groups assuming they support the entity type of the traffic cases in the scenario.

**tsp\_EPTF\_ExecCtrl\_EntityGroupDistributionDeclarators**

Using this module parameter custom entity group distribution can be specified. The name of the entity group identifies the entity group for which the distribution is given. The distribution is a list of LGen names and weight pairs. The LGen name should match with the name of an LGen returned by the f\_EPTF\_Base\_selfName function. Two distribution for the same entity group is not allowed.

### Phases

**tsp\_EPTF\_ExecCtrl\_PhaseList\_Declarators**

This module parameter is used to define phase lists. Default value: empty list, i.e. no phase lists declared.

**tsp\_EPTF\_ExecCtrl\_ScenarioGroup\_Declarators**

This module parameter can be used to define scenario groups. Scenario groups are used for syncronized phase handling. Each scenario in a scenario group should support the same phase lists. The scenario names in this modulepar should be one of the scenario instance names defined in the modulepar tsp\_EPTF\_ExecCtrl\_Scenario2EntityGroupList**.** Default value: empty list.

### Regulator parameters

**tsp\_EPTF\_ExecCtrl\_RegulatorNames**

This module parameter specifies the names of the regulators that can be used to regulate scenario parameters such us Scenario CPS or Traffic case CPS. The regulators should have selfNames chosen from this list.

**tsp\_EPTF\_ExecCtrl\_RegulatedItems**

This parameter defines the assignment between regulators and regulated items. The regulated items can be Scenario CPS that are identified by EntityGroup name and Scenario name, and Traffic case CPS, identified by EntityGroup name, Scenario name and TrafficCase name. If the scenario is in a scenario group with multiple execution phases, then the phase in which the regulation should take place has to be set also. The regulated items belonging to a regulator or the phase in which the regulation should take place can be changed during runtime from the GUI also.

### Time profile parameters

**tsp\_EPTF\_ExecCtrl\_TimeProfileDescrList**

This module parameter defines the list of available TimeProfile descriptors. Each time profile descriptor has a name and a list of time-targetValue pair. So each TimeProfile descriptor is simply a time-series of target values. These time profile descriptors can be used to build complex time profiles.

**tsp\_EPTF\_ExecCtrl\_TimeProfileList**

This module parameter defines the list of TimeProfiles. Each time profile is built up from TimeProfile descriptors. It is possible to set the start time (with respect to the start of the scenario), period of repetition (time at which the time series will be repeated) and the number of repetition (the number of times the time series is repeated) for each TimeProfile descriptor (time series). These time profile descriptors can be assigned to Traffic Case CPS-s and Scenario CPS-es to define time-dependent CPS levels.

**tsp\_EPTF\_ExecCtrl\_TimeProfile2TcList**

This module parameter can be used to assign a time profile to the CPS of a traffic case identified by the EntityGroup name, Scenario name and the TrafficCase name, or a CPS of a weighted scenario. The traffic case name for weighted scenario is ignored when identifying the scenario.

### LGen pool parameters

**tsp\_EPTF\_ExecCtrl\_LGenPool\_Declarators**

This module parameter can be used to define the LGen pools. LGen pools are collections of load generators on specific hosts created by user written creator functions.

**tsp\_EPTF\_ExecCtrl\_EntityGroup2LGenPool\_List**

This module parameter is used to assign an EntityGroup to an LGen pool. An entity group can be assigned to one LGen pool only, but several entity groups can be assigned to the same LGen pool.

If an entity group is assigned to an LGen pool, all scenario instances belonging to the entity group will be executed on the LGens of the LGen pool.

If an entity group is not assigned to any LGen pools, it will be deployed on the default pool. The default pool is the collection of LGens that are created by the user. The number of LGens in the default pool should be specified in the init function of ExecCtrl.

### Parameter ranges

Parameter ranges are “chained counters” used to iterate through several integer ranges in a hierarchical way. An integer range defines finite number of increasing consecutive integer values with a start value (baseOffset) and count (count). Ranges are used for iterate through its element. The iterator (the actual value of the range) can be used directly or for referencing or indexing other list types inside a traffic generator application. A list of range items defines a chained range. A traffic case in a scenario can have zero or more parameter ranges chained together. The next element of a chained range is calculated by invoking f\_EPTF\_Common\_incParamRanges(). When a chained range increased, always the least significant range iterator is increased with one. When all the values used out (“overflow”) it increases the next range iterator recursively, etc. The function returns with true, when the most significant iterator (the 0th element inside the range list) finished. When nrOfRangeLoop was set for this traffic case type in its declarator, a traffic case finish condition check is also performed.

Parameter ranges may be decomposed into several Load Generators in order to create discrete, non overlapping sub-ranges of the original range by setting the flag enableSplit to true.

Example: Scenario “scenario\_0” has 3 ranges handled as one chained range. The 0th element will be decomposed by ExecCtrl during initialization (enableSplit := true).

tsp\_LGenBase\_ScenarioDeclarators := {

{

name := "scenario\_0",

tcList := {

{

tcName := "TC0",

startDelay := 0.0,

cpsToReach := 1.0,

enableEntities := true,

enable := true,

ranges := {

{name := "myRange\_MSB",

enableSplit := true,

baseOffset := 10,

count := 30},

{name := "myRange\_middle",

enableSplit := false,

baseOffset := 10,

count := 30},

{name := "myRange\_LSB",

enableSplit := false,

baseOffset := 0,

count := 9}

}

//…

### Parameter decomposition

The following traffic scenario parameters are decomposed during the initialization:

* cpsToReach: always decomposed. The total traffic case cps values are split among load generators running the corresponding scenario. The amount of CPS distributed proportional to the entity number distribution
* cpsToReach: always decomposed. The total weighted scenario cps values are split among load generators running the corresponding scenario. The amount of CPS distributed proportional to the entity number distribution
* ranges: decomposition can be disabled. Creates discrete sub-ranges of the original one. For more info see ‎2.5.7
* group finish conditions are not decomposed. All conditions except nrOfExecStart, nrOfSuccesses, nrOfFails, nrOfTimeouts, nrOfErrors are distributed without splitting (e.g. execTime). They are evaluated locally on the clients. However, the exceptional ones listed above are not distributed. They are evaluated globally in ExecCtrl by summing the appropriate values in the clients.

### Obsolete parameters

The module parameters listed in this section are obsolete. They are kept for backward compatibility.

**tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators**

This module parameter is defined in module EPTF\_CLL\_LGenBase\_Definitions. It is used to declare a list of traffic scenarios. The referred traffic case type names should be consistent with traffic case declarations. Empty scenarios (scenario with zero nr of traffic cases) are not allowed.

**tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators2**

A newer version of tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators.

**tsp\_LGenBase\_TcMgmt\_WeightedScenarioDeclarators**

A scenario declarator with CPS. In this scenario the traffic cases have weights which determine how the scenario CPS is distributed among its traffic cases.

**tsp\_LGenBase\_tcTypeDeclarators**

The old version of the tsp\_LGenBase\_tcTypeDeclarators2. This module parameter can only be used with scenarios defined in tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators.

Note, that the tsp\_LGenBase\_TcMgmt\_tcTypeDeclarators2 can be used with the scenario declarators tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators2, tsp\_LGenBase\_TcMgmt\_WeightedScenarioDeclarators and tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators3. They replace the pair tsp\_LGenBase\_TcMgmt\_ScenarioDeclarators and tsp\_LGenBase\_TcMgmt\_tcTypeDeclarators. This latter pair of parameters is kept for compatibility. These parameter pairs can be used at the same time also.

**tsp\_LGenBase\_Scenarios2Grps**

This module parameter is defined in module EPTF\_CLL\_LGenBase\_Definitions. It is used to define a list of entity group – traffic scenario associations. The referred entity group and scenario names should be also declared in tsp\_LGenBase\_EntityGrpDeclarators and one of the scenario declarators.

### Traffic case custom parameters

Traffic case custom parameters provide the possibility to assign arbitrary traffic case type specific parameters inside the common traffic case type descriptors. Since the parameter names and even the values are represented only with charstrings, it is the responsibility of the application libraries to provide function references to convert the parameters to the desired data types during the traffic case initialization. The corresponding functions should be declared prior the test case initializations by f\_EPTF\_LGenBase\_declareFunction().

Example: “TC0” has 2 custom parameters, expecting here 2 different registered functions for the parameters, since they used in different application libraries.

tsp\_LGenBase\_tcTypeDeclarators := {

{

name := "TC0",

//…

params := {

{aName := "applib\_A\_paramInterpreter", pName := "header", pValue := "$EXTPATH/sip\_support/templates/deregister.header"},

{aName := "applib\_B\_paramInterpreter", pName := "boost", pValue := "true"}

}

//…

Since this type of custom parameters makes the configuration too confusing it is adviced to avoid using them. An alternative way to define custom parameters is to use custom module parameters. Inside this custom module parameter place the identification information of the traffic case to which the custom parameter should belong. For example for a traffic case instance this identification information is the triplet of the entity group name, scenario name and traffic case name. For a traffic case type, it is the name of the traffic case type. Of course custom parameters can be defined not only for traffic cases but anything identifiable you like. The straightforward advantage of custom module parameters over the traffic case custom parameters is that custom module parameters can be structured and not only charstring type values are allowed.

So the example above can be rewritten like this using custom module parameters:

tsp\_Applib\_A\_paramInterpreter\_customParam := {

tcInstanceId := {

eGrpName := "egrp",

scName := "sc",

tcName := "TC0"

},

header :=

"$EXTPATH/sip\_support/templates/deregister.header"

}

tsp\_Applib\_B\_paramInterpreter\_customParam := {

tcTypeId := "TC0",

boost := true

}

### Customizable GUI

The ExecCtrl GUI layout can be customized. This can be done without using the Execution Control UI Handler. The custom GUI layout can be created by loading and appropriate XML into the UIHandler feature. This XML should contain dataSources for ExecCtrl. The dataSource name of the ExecCtrl is defined by the constant c\_ExecCtrl\_DataSource\_sourceId. The required data elements and iterators are created by ExecCtrl if the dataSource component is passed to the init function of ExecCtrl.

These data elements and iterators have certain parameters. The value of these parameters usually come from the iterators. The description of the data elements, iterators and their parameters can be found in the HTML API doc.

# Execution Control UI Handler

The Execution Control UI Handler (ExecCtrl\_UIHandler) is an extension of the Execution Control in order to display/edit important Execution Control specific information on the GUI. It also creates the Start/Stop/Snapshot/Exit buttons and the status text lines.

To use the Execution Control UI Handler the component should extend the ExecCtrl\_UIHandler\_CT component instead of the ExecCtrl\_CT. Its init function needs two additional parameters: the component reference to the UIHandler and the widget ID where the Execution Control tab will be placed. The f\_EPTF\_ExecCtrl\_UIHandler\_behavior function starts the execution of the Execution Control UI handler. It calls the behaviour function of the Execution Control f\_EPTF\_ExecCtrl\_behavior at the end. These behaviour functions are just examples and need to be modified by the user of ExecCtrl UIHandler feature.

Most of the ExecCtrl functionality is displayed on the GUI created by the ExecCtrl\_UIHandler under the ExecCtrl tab. To access this GUI from the CLI the EPTF Variable names associated to the widgets on the GUI shall be used. These names are listed in the Function Description document of ExecCtrl ‎[6].

The module parameters used by ExecCtrl UIHandler are listed in the following section.

## Configuration

**tsp\_EPTF\_ExecCtrl\_StartStopTCDisable**

This module parameter can be used to enable/disable the StartTC/StopTC checkboxes on the traffic cases panel.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_disableAutoEnablingOfManualControlModeWhenStopButtonIsPressed**

This module parameter can be used to disable the enabling of the manual control mode when the stop button is pressed.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_generateReport**

This module parameter enables/disables the built-in final test report generation in the ExecCtrl UIHandler. If it is enabled, ExecCtrl UIHandler will write its traffic case statistics report into the final test report file. Default value: false

**tsp\_EPTF\_ExecCtrl\_UIHandler\_noCPSChart**

This module parameter can be used to disable the CPS Chart displayed on the ExecCtrl statistics panel. Default setting: false (i.e. the chart is enabled)

**tsp\_EPTF\_ExecCtrl\_UIHandler\_noDensityChart**

This module parameter can be used to disable the Density Chart displayed on the ExecCtrl statistics panel for density type FSM statistics. Default setting: false (i.e. the chart is enabled)

**tsp\_EPTF\_ExecCtrl\_UIHandler\_CPSChart\_maxPointsCurrent**

This module parameter can be used to set the maximal points displayed on the CPS Chart for the traffic cases for the current CPS graph. Default: 2048

**tsp\_EPTF\_ExecCtrl\_UIHandler\_CPSChart\_maxPointsTarget**

This module parameter can be used to set the maximal points displayed on the CPS Chart for the traffic cases for the target CPS graph. Default: 2048

**tsp\_ExecCtrl\_UIHandler\_densityChartOrientation**

This module parameter can be used to set the orientation of the Density Chart on the **FSM Stats** tab on the **Statistics** panel. The default orientation is vertical. Possible values are defined by the type XULOrientation.

**tsp\_ExecCtrl\_UIHandler\_densityChartLabelType**

This module parameter defines the label type used for the density Chart bars on the **FSM Stats** tab on the **Statistics** panel. Default value is percentage. Possible values are defined by the type DistributionchartLabeltype.

### Refresh rates

To be able to define different refresh rates for different ExecCtrl GUI elements, the following configuration parameters are provided. The type of all parameters are integers. If zero value is given, the default refresh period will be used, i.e. the syncInterval defined in EPTF Variable. For positive values, the refresh period will be set to the minSyncInterval time defined in EPTF Variable multiplied by the number given for the module parameter.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_ScGrpStatus**

This parameter defines the refresh rate of the ScenarioGroup status LEDs.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_CPSChart**

This parameter defines the refresh rate of the CPS Chart on the Traffic case data panel.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_TCStats**

This parameter defines the refresh rate of the Traffic case Entity statistics on the Traffic case data panel.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_TimeElapsed**

This parameter defines the refresh rate of the elapsed time on the status line.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_EntityResourcesTab**

This parameter defines the refresh rate of the Entity resource data on the Client Resources panel.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_targetCPS**

This parameter defines the refresh rate of the target CPS values.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_currentCPS**

This parameter defines the refresh rate of the current CPS values.

**tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_regulatedItem**

This parameter defines the refresh rate of the regulated item data on the Regulator panel.

**tsp\_EPTF\_ExecCtrlClient\_UIHandler\_refreshRate\_FSMStats**

This parameter defines the refresh rate of the FSM statistics on the Traffic case data panel. This parameter belongs to the ExecCtrlClient\_UIHandler.

### Setting custom refresh rates for the GUI elements

Using the module parameters or API functions it is possible to set custom refresh rates for values displayed on GUI by categories. It can be useful when these parameters are not relevant for the user but they generate high CPU load on the UIHandler component.

The most of the load is caused by the Traffic case statistics and the FSM statistics (if there are any). To decrease the load, first the refresh rates for these elements should be decreased by increasing the value of the tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_TCStats and tsp\_EPTF\_ExecCtrlClient\_UIHandler\_refreshRate\_FSMStats module parameters. These will increase the refresh period for traffic case statistics and FSM statistics.

To achieve more responsibility e.g. on the target CPS values the tsp\_EPTF\_ExecCtrl\_UIHandler\_refreshRate\_targetCPS should be decreased.

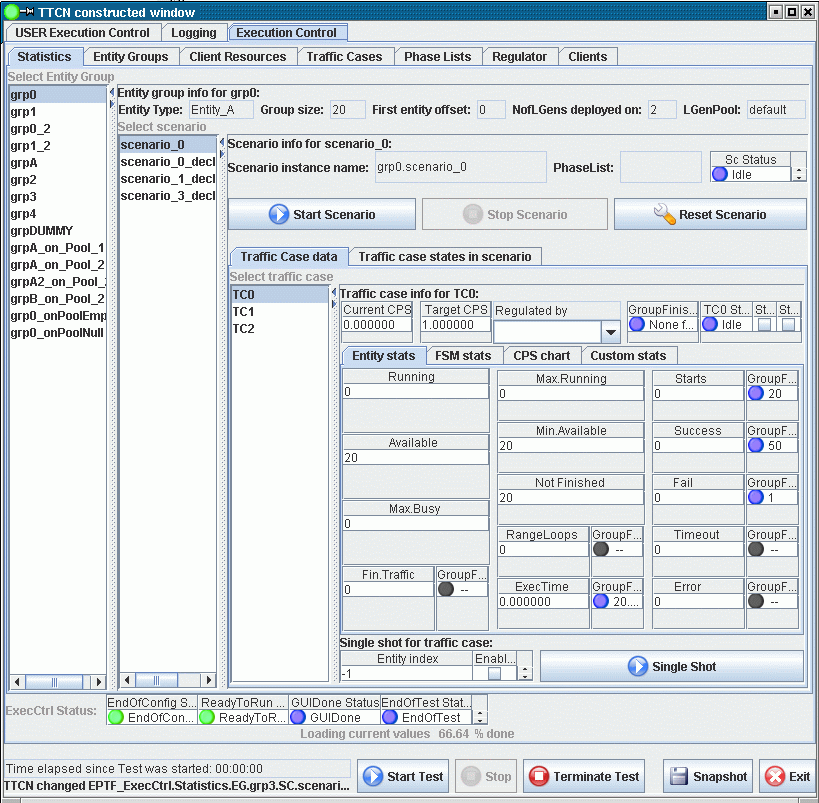
Please note that decreasing these parameter values not always shortens the refresh period on the GUI. This is the case when the source variable is calculated in a longer refresh period, e.g. for the traffic case statistics the source statistics is calculated by the LGenBase with the default refresh period.

## Using the GUI

The Execution Control GUI is created by the ExecCtrl\_UIHandler component. It is a tabpanel, which is placed to a tabbox widget id specified in the pl\_EPTF\_GUI\_Main\_Tabbox\_WidgetId argument of its init function f\_EPTF\_ExecCtrl\_UIHandler\_init\_CT.

The ExecCtrl GUI is shown on the picture below:

Picture 1: The ExecCtrl GUI

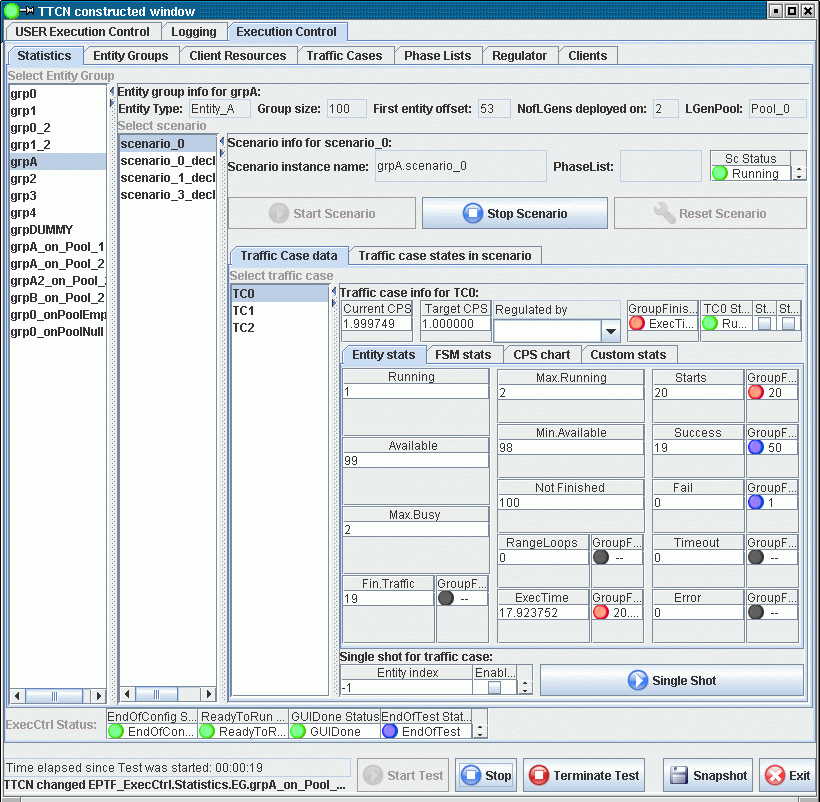


Execution Control tab

There are the following tabs under the **Execution Control** tabpanel: **Statistics**, **Entity Groups**, **Client Resources**, **Traffic Cases**, **Phase Lists**, **Regulator** and **Clients**. These tabs are described in detail in the coming sections.

### Statistics tab

The statistics panel shows the parameters, control buttons and statistics data of entity groups, scenarios and traffic cases. Picture 2 shows the **Statistics Panel**.



Picture 2: The Statistics tab for non-weighted scenarios

1

2

3

4

5

6

7

8

9

10

The description of the items on the GUI:

1. Select entity group, scenario and traffic case
2. Entity group and scenario info
3. Traffic case parameters and statistics
4. Scenario and traffic case control buttons
5. Single shot parameters and start button
6. Main control buttons
7. Group finish condition status LEDs and the condition criteria
8. Scenario, traffic case and group finish status LEDs
9. Runtime GUI status line
10. ExecCtrl StatusLEDs

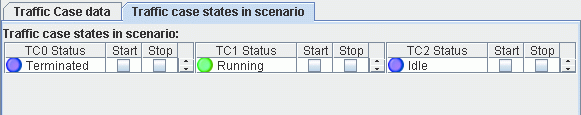
The traffic case can be selected by choosing the appropriate entity group, scenario and traffic case from the lists at (1). To increase the display area, the arrows on the top-right corner of the lists can be used to open and close the lists.

Basic information of the selected entity group, scenario and traffic case is displayed at (2). For entity groups the name of the entity group, the entity type, the number of entities in the group, the number of active entities in the entity group, the offset of the first entity in the group, the number of LGens the entity group is deployed on and the LGenPool the LGen is placed are displayed. If there is no LGen the entity group could be deployed on, the *Number of LGens deployed on* counter will show zero.

For scenarios the scenario instance name is displayed. This name, if had not been specified in the configuration file, is automatically generated from the entity group name and the scenario declarator name. The PhaseLists item shows the supported phase list.

An overview of the status of the traffic cases are displayed in the format of status LEDs on the **Traffic case states in scenario**.

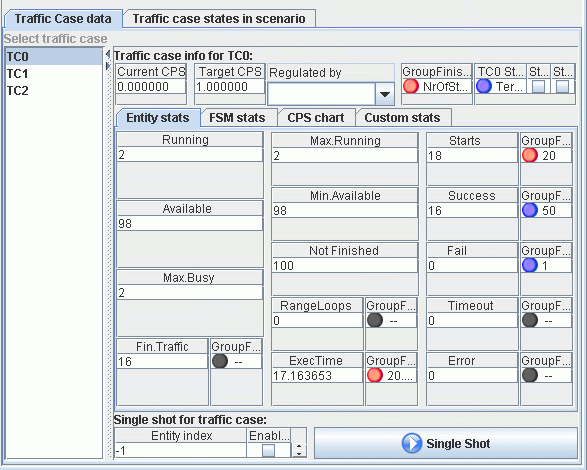
Picture 3a: Traffic case states in scenario detail



On the **Traffic Case data** tab the following traffic case parameters are displayed (in case of non-weighted scenario) at (3):

* Target CPS, which is the CPS set by the user (or by the regulator)
* Regulator that regulates the CPS (this item is only shown when regulators are used) If the scenario is in a scenario group with multiple execution phases, the regulator value displayed corresponds to the current phase. Changing the regulator value will change the regulator assigned to the current phase.

Picture 3b: Traffic case entity information in detail



For each traffic case a couple of tabs provided to display useful information. The **Entity Stats** tab shows aggregated LGenBase statistics and group finish condition information in the form of status LEDs. On the **FSM Stats** tab the TC scope statistics declared in the FSM tables are automatically collected, if available. The **CPS chart** tab shows the current CPS and the target CPS of the traffic case (or scenario) graphically. The **Custom stats** tab is provided to place custom parameters to the GUI by the user. This tab is empty by default and has the widget id c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".TrafficCaseStats.CustomStats.tab".

#### Entity Statistics

The **Entity stats** tab shows the following statistics:

* Current CPS, which is the actual CPS generated by the LGens
* Running: this is the number of entities currently in running state
* Available: the available entities
* Max.Busy: the maximum of busy entities
* Fin.Traffic: the number of entities that finished the traffic
* Max.Running: maximum of running entities
* Min.Available: mimimum of available entities
* Not Finished: number of entities not finished
* RangeLoops: number of times the whole range of entities were used
* ExecTime: the execution time of the traffic case
* Starts: number of times the entities were started
* Success: number of times the entities reported success
* Fail: number of times the entities reported fail
* Timeout: number of times the entities reported timeout
* Error: number of times the entities reported error
* GoS: Grade of service. This gives Success/(Success+Fail+Timeout+Error) in percent.

For more details on these statistics see LGenBase documentation.

These values are sampled periodically. The period is the same as the synchronization period in the EPTF Variable feature. The LEDs next to the values display the group finish condtition state. The colour blue means that the condition is not fulfilled wheras the red colour indicates that the condition has fired. The numbers next to the LEDs show the value of the condition threshold. If a LED is black and there is no text, it means that the corresponding condition was not set for this traffic case.

The (4) are the control buttons of the scenario and the traffic cases in the scenario. They can be used to start/stop/reset the scenario or start/stop the traffic cases in the scenario.

Single shot traffic can be initiated by the *Single Shot* button at (5). This will start the traffic case on the specified entity in the entity group. The logging of the single shot traffic can be enabled by the *Enable Log* checkbox. The entity can be selected by specifying the *Entity index*. It has to fall into the entity group (from 0 to groupsize-1), or -1 is accepted for automatic entity selection. Single shot traffic can be initiated at any stage of traffic case execution.

The main control buttons can be found at the bottom of the panel at (6). These buttons can be used to start all enabled scenarios, stop all running scenarios, terminate all test case execution, generate snapshot, or exit from the application.

The group finish condition status information is displayed by the LEDs at (7). The LEDs are blue if the condition is set for the given item, and next to the LED the criteria is written. The LED colour changes to red if the condition is fired. If the LED is black and the text shows “—“, it means that the given condition is not monitored. Black LED with text “NA” on anywhere on the ExecCtrl GUI means that the value is not available.

The current status of the Scenario, the traffic cases in the scenario, the selected traffic case and the group finish condition overall status is displayed at (8). The group finish status LED colour follows the colours of LEDs at (7), but the text shows the condition that fired last. The traffic case and scenario status LEDs are the aggregated value of traffic case and scenario statuses in the LGens. If the status is different on the LGens (out-of-sync), the LED shows “Mixed” in colour yellow. If the LED is Black and shows “Off”, that means that there is no LGens suitable for executing the scenario and the traffic cases. It can be a mis-configuration, or the LGens exited by some reason.

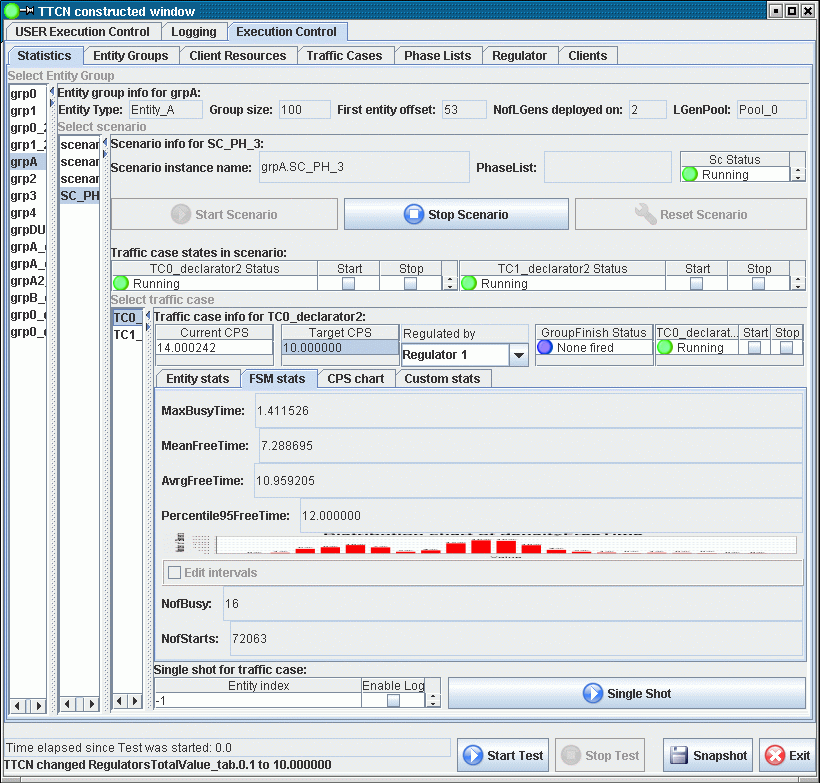
At number (9) some basic status information is show. One of them is the time elapsed since the test was started. This is the time elapsed in seconds since the execution was started if manual mode false, or since the *Start Test* button was pressed. This time is used by the time profiles also to determine the time-dependent CPS levels. At the bottom line the name of the last updated widget is shown.

At number (10) the status information of the ExecCtrl component is shown. The status is displayed in LED format for the EndOfConfig, ReadyToRun, GUIDone, EndOfTest events. If a certain event was not detected yet, the corresponding LED colour is blue. When the event happens the LED colour turns green and stays so from that on.

#### FSM Statistics

The **FSM stats** tab shows the aggregated value of traffic case level FSM statistics. These statistics are declared in FSMs on the LGens and ExecCtrl automatically collects and aggregates them and shows their value on this tab. The values are displayed as a {statName,statValue} pair. If the density chart display is not disabled by tsp\_EPTF\_ExecCtrl\_UIHandler\_noDensityChart, the density statistics are displayed graphically on a distribution chart, otherwise it is displayed as an integer list. The next picture shows a example FSM stats panel (the actual items displayed on this tab can vary depending on the FSM definition):

Picture 4: FSM statistics for a traffic case



If no FSM statistics is available, a gray text is displayed instead, stating that there is no available data to display.

The density chart on this tab displays how many elements of the measured data falls into a given interval. The various value intervals are displayed on the horizontal axis, while the number of data inside the intervals is displayed as a vertical column.

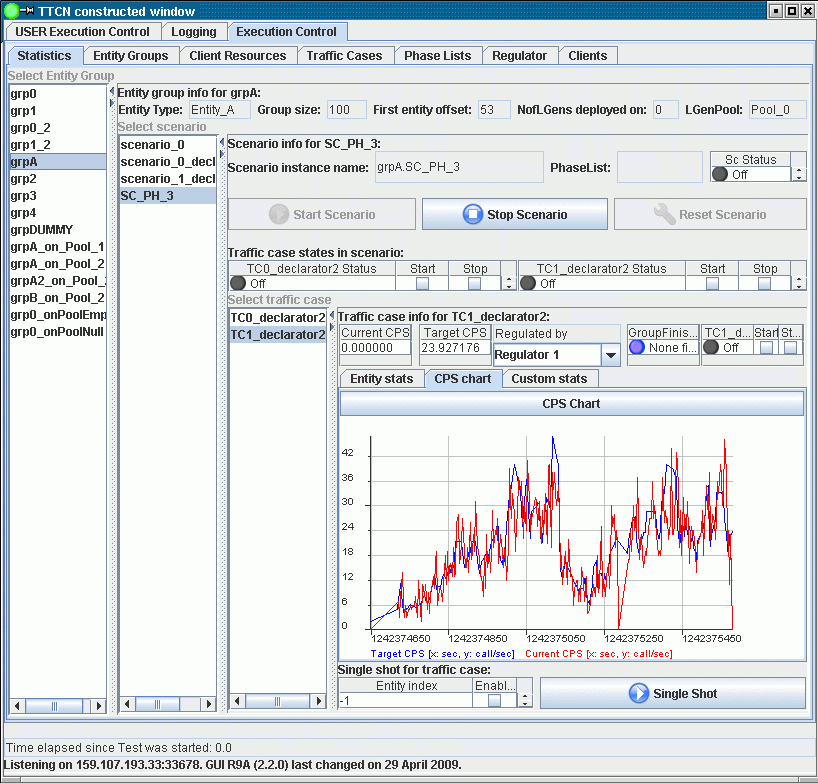
Zooming in and out is possible by selecting or inverse selecting a rectangle area on the chart while pressing and holding down the left mouse button.

By right-clicking on the density chart a pop-up menu comes up through which the chart can be saved into a PNG image file, or several chart properties can be changed. These properties are lost when the GUI is restarted.

#### CPS Chart

For each traffic case a **CPS chart** is provided. This chart shows the target CPS and the current CPS of the traffic case. In case of weighted scenarios the targetCPS value shows the targetCPS value of the scenario. The CPS chart is shown on the next picture:

Picture 5: CPS chart for a traffic case



To zoom in the chart select the rectangle to zoom to by the mouse while holding down the left mouse button. To restore original zoom level press the **CPS Chart** button above the chart.

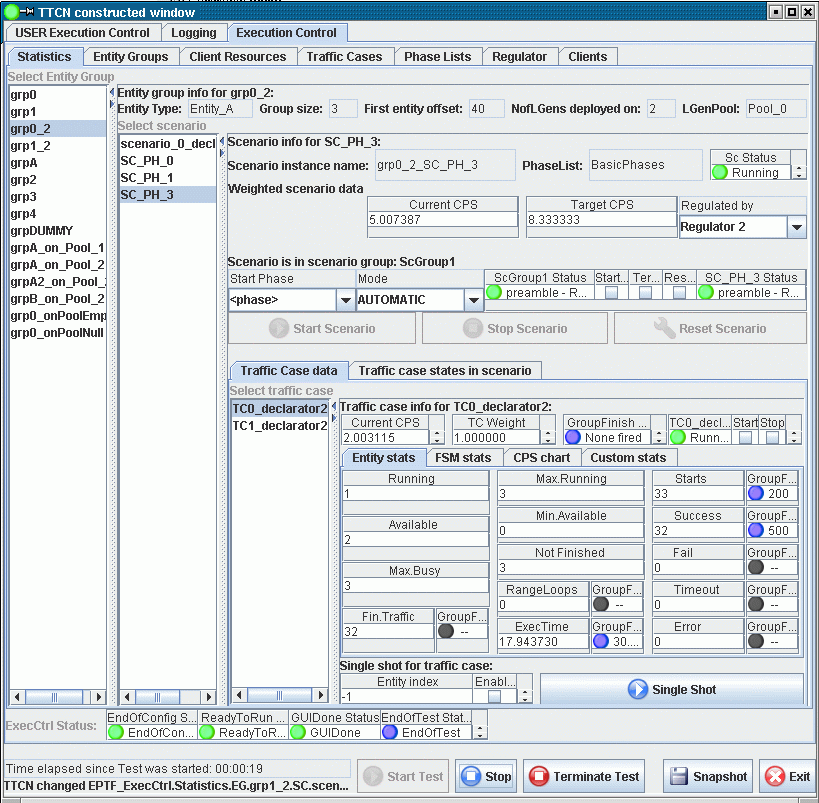
The chart can be saved into a PNG image the by right clicking on the chart and selecting **Save as image ..** from the menu.

This chart is not displayed if the tsp\_EPTF\_ExecCtrl\_UIHandler\_noCPSChart module parameter is set to true.

#### Weighted scenario data

Some additional information is displayed on the Statistics panel if the scenario is weighted or of the scenario belongs to a scenario group.

It is possible to add user defined custom statistics to the traffic case information panel. These will be displayed under the Custom charts tab. The widgetId to use: c\_EPTF\_ExecCtrl\_statisticsRoot&".EG."&vl\_eGrpName&".SC."&vl\_scTypeName&".TC."&vl\_tcName&".userSpecificTcData".



1

2

Picture 6: Statistics tab for weighted scenarios and/or for scenarios in scenario groups

3

1. Weighted scenario parameters (only displayed if scenario is weighted)
2. Scenario group information (only displayed if scenario is in a scenario group)
3. Traffic case weight

For weighted scenarios the Current CPS, Target CPS and the regulator is shown that regulates the CPS at (1). Notice, that at the traffic case parameters (3) the traffic case weight is displayed instead of the Target CPS.

If the scenario is in a scenario group (phase handling) some basic information and control buttons are available at (2). These include the start phase drop down list, the phase selection mode, the scenario group status LED displaying the current phase together with its state and the status of the selected scenario in the group. Checkboxes are provided for starting/stopping the current phase in the scenario group, stopping the scenario group execution and resetting the scenario group. Most of these information can be found on the *Phase Lists* tab also. The scenario control buttons are disabled if the scenario is in a scenario group. Note, that stopping the scenario group can be achieved by unchecking the *StartCurrentPhase* checkbox.

If the scenario group is not running (i.e. idle) the current phase can be changed by changing the *Start Phase* value. The regulator displayed corresponds to the regulator used in the current phase. Changing the regulator will change the regulation in the current phase only.

**Scenario Group control buttons**

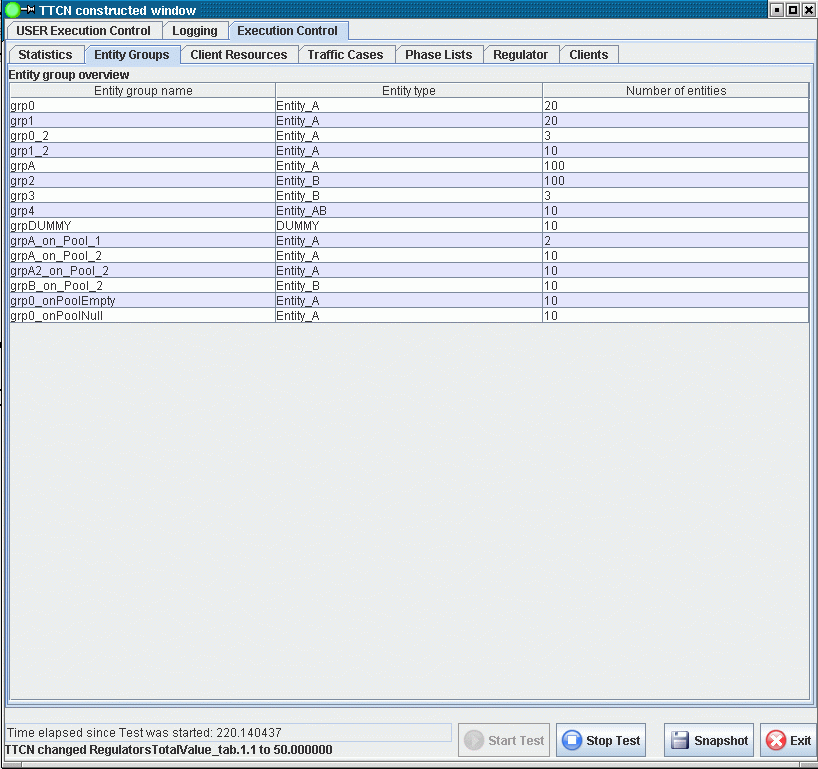
There are three check boxes that can be used to control the scenario group execution: **StartCurrentPhase**, **TerminateGroup** and **ResetGroup**.

**StartCurrentPhase**:  
Pressing this button will start the actual phase in the scenario group. Pressing it again (removing the tick mark) will stop the current phase and if the mode is AUTOMATIC it will start the next phase automatically.  
  
Example:  
Current phase is preamble: Pressing StartCurrentPhase will stat the preamble phase. If the mode is AUTOMATIC sooner or later the loadgen phase will be started. Unticking the  StartCurrentPhase checkbox will stop the loadgen phase and the postamble phase will begin.  
  
Use the StartCurrentPhase checkbox if you would like to stop the current phase and continue with the next phase automatically.  
  
**TerminateGroup**:  
This check box can be used to stop the execution of the scenario group entirely. After this, the scenario group can be started by StartCurrentPhase, but all data will be reset.  
  
Example: If you press the TerminateGroup checkbox in the loadgen phase, the scenario group will stop, and the next phase will not be started.  
After that if the StartCurrentPhase checkbox is pressed it will start the scenario group in the loadgen phase (since this is the phase it was stopped in). But because the database is reset after stop, it is possible that no traffic will be generated: Titansim will think that no users are registered. To be able to generate traffic after pressing TeminateGroup, the phase should be changed back to preamble.

### Entity Groups tab

The **Entity Groups** panel shows the overview of the entity groups declared in the system. It presents information on the entity type and the number of entities in the group (i.e. the group size). The following picture displays the **Entity Group** tab:

Picture 7: The Entity Groups tab



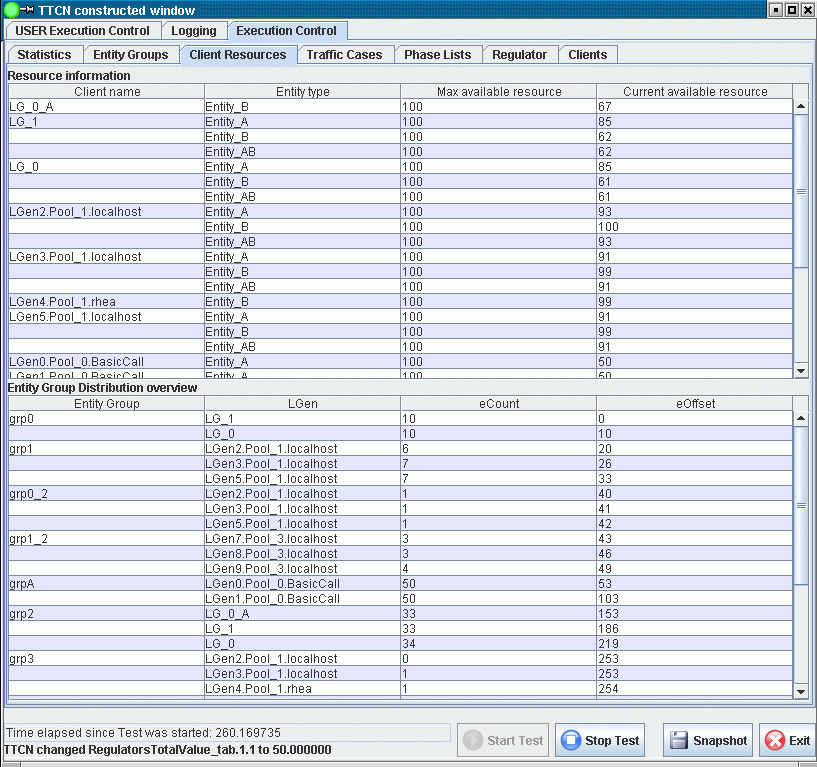
### Client Resources tab

The **Client Resources** panel shows the resource information of the LGens. For every LGen the supported entity types, the number of maximal available entities of a given entity type and the currently available number of entities are displayed on the *Resource information* section of this panel. The number zero is displayed if no entities available on the given LGen, or if the LGen terminated. The number -1 is shown when there is no upper limit on the entity number. If the line is empty for an LGen, that means that the LGen does not support any entity types.

Under the *Entity group distribution overview* the entity groups are listed according to their LGen distribution. For each entity group those LGens are listed on which they are distributed. For each distributed part the *eCount* shows its size on the LGen, i.e. how much entities it contains. The value of *eOffset* gives the global offset of the first entity of the distributed part. If an entity group was not distributed to any LGens, the corresponding columns are left empty.

The sum of the *eCount* values for the same entity group gives the size of the entity group (which is displayed on **Entity Groups** panel also); while the first *eOffset* value for an entity group gives the global offset of the entity group itself. The picture below shows this tab.

Picture 8: The Client Resources tab



## Traffic Cases panel

On the **Traffic Cases** panel the scenario and traffic case data is shown. It gives an overview of the distribution of scenarios on LGens and an overview of scenario and traffic case parameters. The states of the scenarios are displayed per LGen and also the overall scenario state is shown as a status LED. A short text next to the LED gives textual information on the current state. These states correspond to the states defined in LGenBase. The accumulated scenario status LEDs, which show the overall state of the scenario on all LGens, are updated if the states on all LGens are the same. Otherwise the “Mixed” text is displayed and the LED colour is set yellow. The following list describes the states not present in LGenBase:

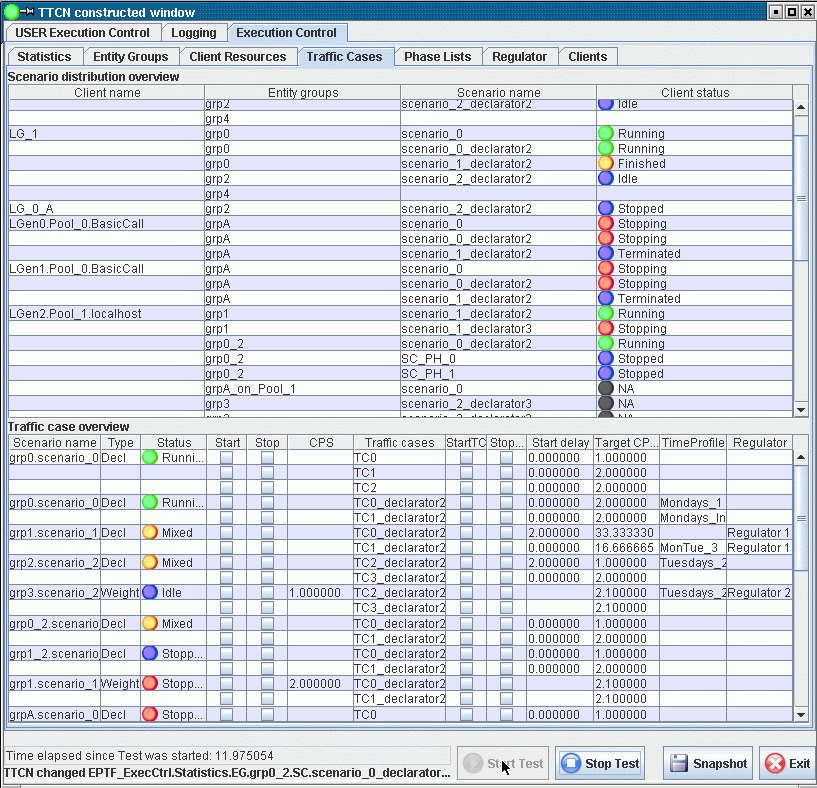
* Black – “NA”: The value is not available. The LGen is not able to execute that scenario, because entities are not available
* Black – “Off”: The corresponding LGen is terminated; information is not available any more.
* LED is missing: no scenario is assigned to that entity group or the LGen does not support any entity groups, i.e. no entity group is deployed on that LGen, or no scenario is assigned to that entity group.
* Yellow – “Mixed” : the state is out of sync, the scenario state is different on the LGens

The **Traffic Cases** panel is divided into two parts. The *Scenario distribution overview* and the *Traffic case overview*.

The *Scenario distribution overview* presents the scenario distribution information. For each LGen the entity groups deployed on them and the scenarios assigned to the entity groups are listed. The *Client status* column shows the status LEDs for the scenario.

The *Traffic case overview* collects the main traffic case parameters and control buttons. It also displays the synchronized scenario status. The *Type* column informs whether the type of the scenario is normal (*Decl*) or weighted (*Weight*). Next to the status LEDs are placed the start/stop checkboxes, which can be used to start and stop the scenario. These checkboxes can be disabled from the configuration file. If the scenario is weighted, the *CPS* column can be used to set the target CPS for the scenario. Then the traffic cases in the scenario are listed below each other. The *StartTC*/*StopTC* checkboxes are provided to start and stop the traffic cases. The *Start delay* column shows the time the start of the traffic case is delayed after the scenario was started. In the *TargetCPS/Weight* column the target CPS of the traffic case (normal scenario) or the traffic case weight (weighted scenario) can be adjusted. The *TimeProfile* and *Regulator* columns show if there is a time profile or a regulator configured to control the CPS value automatically. The Traffic Cases panel is shown below:

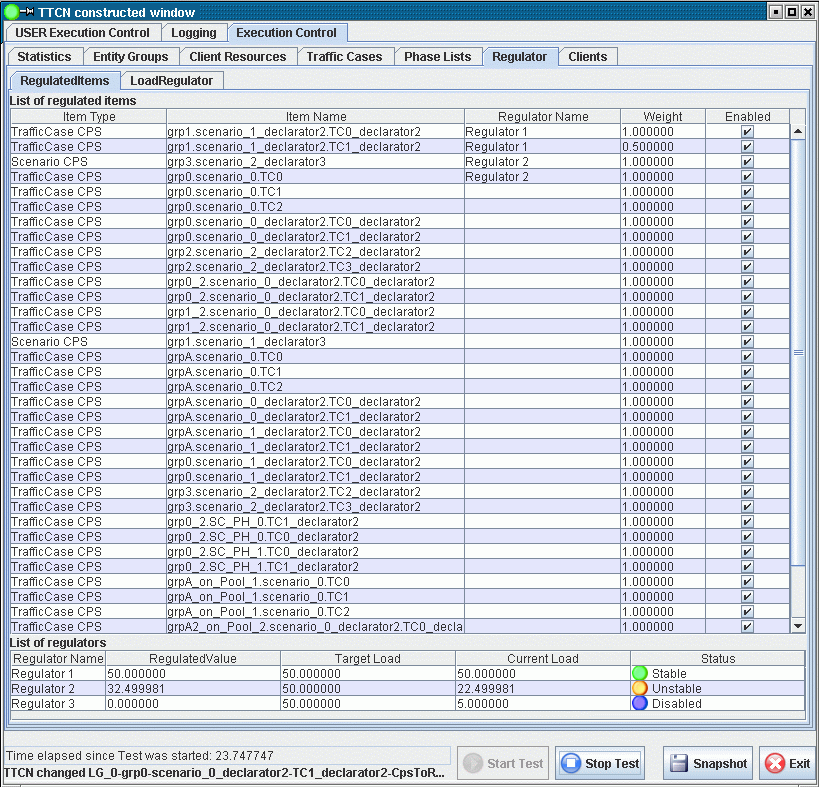
Picture 9: The Traffic Cases tab



## Regulator panel

This panel shows all CPS values that can be regulated by regulators. These include scenario CPS-es and traffic case CPS-es. The type is shown in the *Item Type* coloumn. Every traffic case CPS value is identified by the entity group name, scenario name and traffic case names separated by ‘.’. Scenario CPS-es do not contain the traffic case name of course. This information is shown on the *Item Name* column. The *Regulator Name* shows the name of the regulator that is set to regulate that item. This can be changed during runtime. It is possible to regulate more than one item with the same regulator. In this case the weight of the items determines what portion of the total CPS is used for the given item. Each item can be enabled in the *Enabled* column by checking the checkbox. The **Regulator** panel is shown below:

Picture 10: The Regulator tab

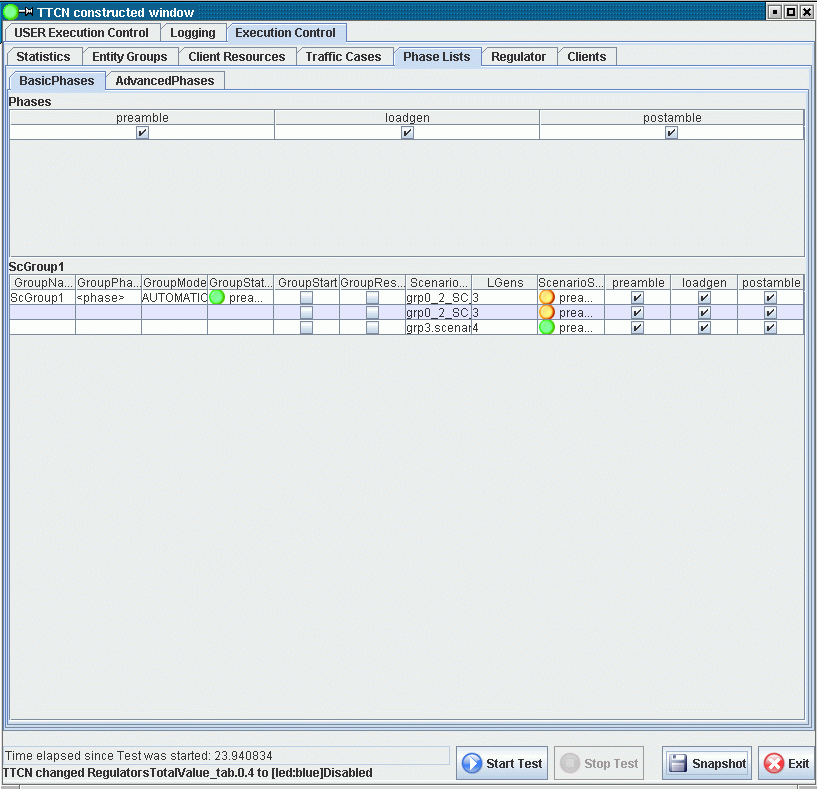


Below the list of regulated items the data of the regulators are listed. It is possible to adjust the target load here also. The required CPS that is needed to reach that target load is displayed in the *Regulated Value* column. This value is split based on the weights between the regulated items belonging to that regulator. The actual load is displayed in the column *Current Load*. The status of the regulators can be monitored via their status LED. Stable means, that the target load is reached, instable means that the regulator is adjusting the CPS to reach the target load level. Disabled means that all items regulated by the regulator belong to a scenario that is not running. If the regulator is not connected, the text Disconnected is shown. In this case the user is able to adjust the *Regulated Value* manually instead of the *Target Load*.

## Phase Lists panel

The **Phase Lists** panel shows the status and control buttons of execution phases for scenario groups. It is described in more detail in the ‎7.7 section.

Picture 11: The Phase Lists tab



# Execution Control Client

The Execution Control Client (ExecCtrlClient) is the client side interface towards the Execution Control component. These clients also referred to as LGens, since they should extend the LGenBase component, are created, configured and started by the Execution Control component. The clients execute the traffic cases.

Each client shall declare the supported behaviour types, entity types, FSMs. The ExecCtrl uses this information to deploy the entity groups on the clients, and executes traffic cases on the corresponding LGens.

The ExecCtrlClients can be organized into LGenPools in order execute traffic cases in parallel, which makes the system scaleable.

## Configuration

The following module parameters can be used to configure the ExecCtrlClient behaviour.

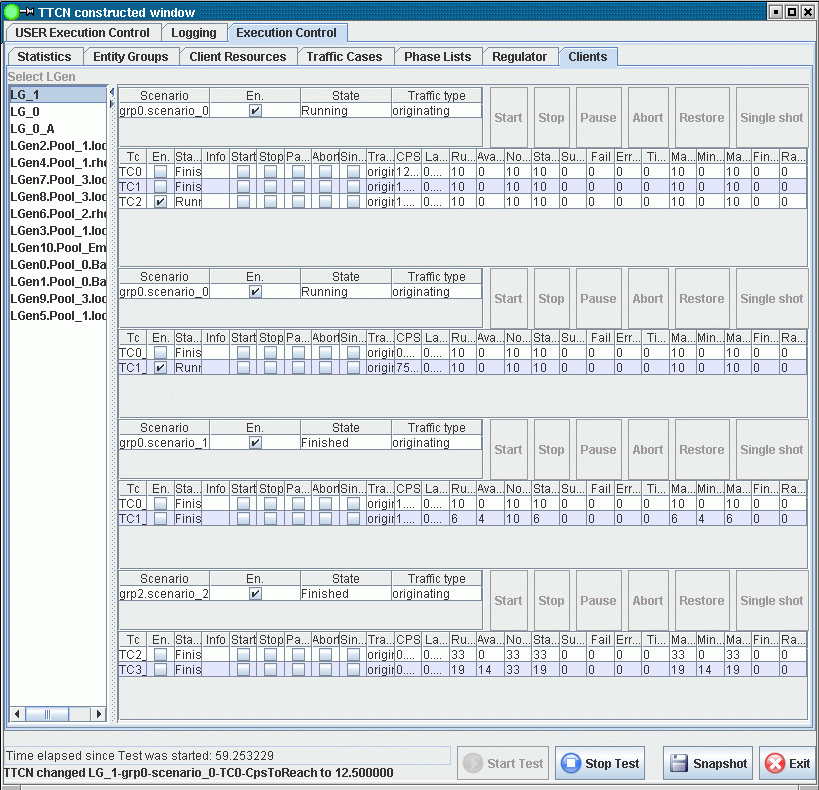
**tsp\_EPTF\_ExecCtrlClient\_terminateIfExecCtrlTerminates**

This boolean type module parameter can be used to enable/disable termination of the ExecCtrlClient component when ExecCtrl terminates. If enabled, all ExecCtrlClient components are shut down before ExecCtrl terminates. Otherwise they are terminated by the Base cleanup mechanism when the system terminates. Default value is: true.

# Execution Control Client UI Handler

The Execution Control Client UI Handler (ExecCtrlClient\_UIHandler) is the extension of the ExecCtrlClient that adds the functionality of displaying LGenBase statistics information on the GUI under the **Clients** tab. It is not possible to change the displayed settings by the user, since they are all controlled by ExecCtrl. This is why the items on the **Clients** panel are disabled. The data displayed on this tab for a given scenario is aggregated on the **Statistics** panel for all LGens running that scenario. The **Clients** panel is shown below:

Picture 12: The Clients tab



## Configuration

**tsp\_EPTF\_ExecCtrlClient\_UIHandler\_createClientTab**

This module parameter can be used to prevent the ExecCtrl clients to create their **Clients** tab on the ExecCtrl panel. If it is set to false the client statistics is not shown. This can speed up execution and shorten the start-up time. Default setting: true (i.e. the panel creation is enabled)

**tsp\_EPTF\_ExecCtrlClient\_UIHandler\_refreshRate\_FSMStats**

This parameter defines the refresh rate of the FSM statistics on the Traffic case data panel.

# Usage

The ExecCtrl or the ExecCtrl\_UIHandler component can be a stand-alone component, which means that after creation its behaviour function can be started on it. After that all ExecCtrlClient components should get the component reference of this component so that they can establish connection towards the ExecCtrl or the ExecCtrl UI Handler component. There can be only one ExecCtrl component in the test configuration. The ExecCtrlClient components can be used as an interface to the LGenBase feature. Note, that in the ExecCtrlClient component the initialization should follow this order:

1. Initialize the LGenBase
2. Declare entity types
3. Initialize the ExecCtrlClient
4. Declare more entity types
5. Report readyToStart towards ExecCtrl

## Using ExecCtrl

The ExecCtrl feature can be used without calling its behaviour function. In this case the ExecCtrl API can be used to initialize, configure, start and terminate it.

The ExecCtrl feature has to be initialized first by its init function. This init function loads the module parameters by default and starts the execution of the test cases automatically.

To be able to load additional configuration data, the loadConfig function can be used. This function has to be called before the operation of ExecCtrl is started. This means that the automatic calling of start function has to be disabled in the init function of ExecCtrl.

When all configuration data is loaded by the loadConfig function, the manual control mode has to be set. The automatic mode means, that when the operation of ExecCtrl is started, all enabled scenarios and scenario groups will be started automatically.

After the configuration of ExecCtrl is finished, the operation of ExecCtrl can be started by the start function. After that the code has to enter into an infinite alt loop. In this alt loop the default altsteps will handle the events and operate ExecCtrl. LGens belonging to LGenPool will be created by the start function, or can be created after the start function is called by the createLGens function. If there are LGens that should be created manually, they can be created after ExecCtrl is started. Regulator components, if they are used, have to be created also after ExecCtrl is started.

During execution, scenarios can be started and stopped; CPS can be adjusted, etc.

At the end of the test the exit function has to be called, or the ExecCtrl component has to be stopped using the Base feature. A final test report will be generated if configured to do so at exit.

## Events in ExecCtrl

The following events can be detected in ExecCtrl and ExecCtrl\_UIHandler:

* EndofConfig: happens when the all client configuration has finished. This event cannot be checked, and there is no callback function for this event in ExecCtrl yet.
* ReadyToRun: happens when all clients are reported ready to run. By the time this event happens, all variables had been created, and subscribes had been done for variables in all ExecCtrlClients. The state can be checked by f\_EPTF\_ExecCtrl\_checkReadyToRun.
* GuiDone: happens when ExecCtrl\_UIHandler finished creating its GUI and all ExecCtrl clients have reported the GUIDone event to ExecCtrl. This condition can be checked by the function: f\_EPTF\_ExecCtrl\_UIHandler\_checkGuiDone in ExecCtrl UIHandler.
* ScenarioGroupPhaseChange: happens when the phase of the scenario group changes. Callback functions that are called at this event can be registered by the function f\_EPTF\_ExecCtrl\_registerScenarioGroupPhaseChangedCallback
* ScenarioStateChange: happens when the state of the scenario on all LGens becomes the same (i.e. becomes syncronized). Callback functions that are called at this event can be registered by the function f\_EPTF\_ExecCtrl\_registerScenarioStateChangedCallback.
* TrafficCaseStateChange: happens when the state of the traffic case on all LGens becomes the same (i.e. becomes syncronized). Callback functions that are called at this event can be registered by f\_EPTF\_ExecCtrl\_registerTrafficCaseStateChangedCallback.
* OnGroupFinishCondition: happens when any of the onGroupFinish conditions becomes true, or when all set to false. Callback functions registered by the function f\_EPTF\_ExecCtrl\_registerOnGroupFinishCallbackFn are called when this event occurs.
* CPSChange: happens when a traffic case or a scenario target CPS changes. Callback functions registered by f\_EPTF\_ExecCtrl\_registerCPSChangedCallback\_TC for traffic cases and f\_EPTF\_ExecCtrl\_registerCPSChangedCallback\_SC for scenarios are called when the target CPS for the corresponding traffic case or scenario changes (i.e. when the f\_EPTF\_ExecCtrl\_setCps\_TC or f\_EPTF\_ExecCtrl\_setCps\_SC function was called).
* EndOfTest: happens when any LGen reports the EndOfTest event or when the Exit button is pressed. Functions registered by f\_EPTF\_ExecCtrl\_registerFinalTestReportGeneratorFn are called.

Events in ExecCtrlClient:

* ReadyToStart: To report this event to ExecCtrl the function f\_EPTF\_ExecCtrlClient\_readyToStart should be called, or it is called automatically in the client’s init function if enabled to do so. Call this function only after all entity types are declared. Entity types declared after calling this function will be invisible for ExecCtrl.
* ScenarioInitialized: When a scenario is created in the ExecCtrlClient, the function registered by f\_EPTF\_ExecCtrlClient\_set\_scenarioInitialized\_fcb is called.
* EndOfConfig: This event is detected when ExecCtrl finished configuring the ExecCtrlClient and the scenarioInitialized callback function finished its operation. Callback functions registered with f\_EPTF\_ExecCtrlClient\_registerEndOfConfigCallback will be called in the order they were registered when this event is detected. The event can be detected by the f\_EPTF\_ExecCtrlClient\_checkEndOfConfig function.
* GuiDone: This event occurs when the ExecCtrl reports this event to the Client (when the ExecCtrl GUI is ready) and all call-back functions registered by f\_EPTF\_ExecCtrlClient\_registerGuiDoneCallback have finished in the ExecCtrlClient. The event can be detected by the function f\_EPTF\_ExecCtrlClient\_checkGuiDone. Also, callback functions can be registered by f\_EPTF\_ExecCtrlClient\_registerGuiDoneCallback. The ExecCtrlClient (if not disabled in its init function) reports a GUIDone event to ExecCtrl when these call-back functions all return. The ExecCtrl synchronizes this reported GUIDone events. It is possible to disable the automatic reply and the GUIDone event can be reported to ExecCtrl later (e.g. after custom GUI has been created) by the f\_EPTF\_ExecCtrlClient\_reportGUIDone function.
* GUIDoneReported: This event happens when the ExecCtrlClient has reported the GUIDone event to ExecCtrl. This event can be checked by the f\_EPTF\_ExecCtrlClient\_checkGuiDoneReported function.
* EndOfTest: happens when any LGen reports the EndOfTest event or when the Exit button is pressed. Functions registered by f\_EPTF\_ExecCtrlClient\_registerFinalTestReportGeneratorFn are called.

## Using LGen Pools

The LGen pools can be useful to deploy the LGen components automatically by the ExecCtrl with the help of user defined LGen creator functions.

LGens created manually will be placed in the ‘default pool’. The number of manually created LGens should be specified in the init function of ExecCtrl or ExecCtrl\_UIHandler (f\_EPTF\_ExecCtrl\_init\_CT, or f\_EPTF\_ExecCtrl\_UIHandler\_init\_CT).

The LGen pool definition consists of a list of triplets containing the creator function, the hostname and the number of LGens to create on that host. Since the creator functions are implemented by the user, they have to be registered by the user into ExecCtrl so that they can be referred to from the configuration file as a string. This registration can be done by the f\_EPTF\_ExecCtrl\_loadConfig function using its pl\_EPTF\_ExecCtrl\_LGenFunction\_Entry\_List parameter. Note, that although it is possible to specify a custom name for the function, the following name can also be used in the tsp\_EPTF\_ExecCtrl\_LGenPool\_Declarators module parameter even if no custom name is given: <Modulename.functionName>.

The creator function should create the LGen (the LGen should extend the ExecCtrlClient\_CT component) on the specified host. It is not checked by ExecCtrl that the LGen created is actually running on the given host. The function should return null if the LGen could not be created, or if the function deploys non-LGen components, i.e. components that do not extend ExecCtrlClient\_CT, or if the creation failed. Otherwise it should return the component reference to the created LGen. The LGen creator function should run on a component that extends ExecCtrl\_CT because the creator functions are called by ExecCtrl.

The Entity Groups can be associated to LGen pools with the help of the tsp\_EPTF\_ExecCtrl\_EntityGroup2LGenPool\_List module parameter. One Entity Group can be assigned only to one LGen pool, while one LGen pool can support more than one Entity Groups. The Entity Groups not associated to any LGen pool will be declared on the default pool.

When using LGen pools, the following should be taken care of:

* The creator functions are registered, otherwise the LGens of the pools cannot be created, because the function names in the configuration file are not recognized.
* Assure that there are LGens in the LGen pool with entity types suitable to support the entity groups associated to the LGen pool.
* The LGens have enough entities to support the Entity Groups.
* At least one Scenario is added to the Entity Group.

Note, that the LGen creator functions shall have a runs on clause with a component that extends ExecCtrl. Otherwise a compilation error is shown stating a ‘runs on cause mismatch’. If the created LGens need the UIHandler component reference (for example for creating their own GUI) the LGen creator function shall run on a component that extends ExecCtrl and UIHandlerClient. The UIHandler component reference can be accessed via the following variable in the UIHandlerClient: v\_UIHandlerClient\_DefaultUIHandler.

## Limited Traffic Execution

Limited traffic execution means that the traffic case is executed only on a part of the entire entity group. In this smaller part of the entity group are the active entities which can execute the traffic. The size of the active part of the entity group can be changed during runtime. If the traffic case is in a running state when the number of active entities is changed, all statistics values of the traffic cases will be reset and the CPS values will be redistributed on the LGens.

The number of active entities has to be between zero and the number of allocated entities. If the value is out of this range, the value will be adjusted to the smallest or largest permitted value.

The initial value of the number of active entities is the number of allocated entities. The traffic case is executed with different number of active entities in the same way as if all the entities were active, except that the number of entities that are allowed to execute the traffic is different. This means that traffic finish actions are executed.

# Multiple Execution Phase Support

## Function specification

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 are synchronized for a Scenario group.
* The Execution Control GUI provides the means to control (start/stop) the scenario groups and their state information is also displayed.

## Overview

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. Let’s see an example. There are three main phases: preamble, load generation, postamble. 

Figure 1 Phase example

When the preamble phase is started the registration and the re-registration traffic cases are started. In the registration traffic case the entities are enabled by default, in the re-registration they are disabled. When an entity makes a successful registration the same entity is enabled in the re-registration traffic case (therefore there is an arrow in the figure), thus this entity will be kept registered. When the registration traffic case was finished (all the entities tried to register in) the execution is going to the next phase. The load generator reports that the phase was finished successfully to the ExecCtrl that synchronizes the phase execution. And the next phase will be launched.

In the load generation phase, the main traffic case is started, this is where the actual call generation takes place (basiCallOrig TC). Initially in this traffic case the entities are disabled, but those who managed to register in were made available by the registration traffic case. As the basicCallOrig is finished it reports its phase finish and as soon as all the scenarios reported this phase was finished the final phase is started.

In the postamble phase the re-registration traffic case is stopped (disabled) and the de-registration traffic case is started. As all the entities are de-registered, the phase is reported as finished.

## Declarations

### Phase Lists

A named phase list can be declared using the tsp\_EPTF\_ExecCtrl\_PhaseList\_Declarators module parameter. It consists of the phase definitions and an enable/disable flag:

tsp\_EPTF\_ExecCtrl\_PhaseList\_Declarators :=

{

{

name := "GeneralPhaseList",

phases :=

{

{

name := "preamble",

enabled := true

},

{

name := "loadgeneration",

enabled := true

},

{

name := "postamble",

enabled := true

}

}

}

//, …

}

### Scenario Groups

A scenario group can be built from scenario instances. These scenario instances will be synchronized inside the group. A declared phase list must be assigned to the scenario group also, the phases of the phase list will be executed on the scenarios. A scenario can be added into the scenario group only if it can handle the given phase list. For each scenario group there is a mode filed, which tells the ExecCtrl whether the group should be executed automatically with no human interaction or manually, where the phases must be executed individually by the user.

A named scenario group can be defined using the tsp\_EPTF\_ExecCtrl\_ScenarioGroup\_Declarators module parameter:

tsp\_EPTF\_ExecCtrl\_ScenarioGroup\_Declarators :=  
{

{

name := "basicCallGroup",

execMode := AUTOMATIC,

scenarioNames := { "orig", "term" },

phaseListName := “GeneralPhaseList”

}  
 //, …  
}

## Databases

There are two main databases

* One for the phase list declarators
* And an other for the scenario groups

Both databases are using FBQ (so removing scenario groups and phase lists is possible to do in the future) and hashmaps to make the lookups faster.

The phase list declarators are stored as is, but from the scenario group declarators we create scenario group instances. A scenario group instance for the above declarator (‎7.3.2) can be seen here

template EPTF\_ExecCtrl\_ScenarioGroup\_Instance t\_myScenarioGroup\_Instance\_storeSG :=

{

name := “basicCallGroup”,

execMode := AUTOMATIC,

scenarios :=

{

{

name := "orig",

idx := -1,

lgenIndexes := {0,1},

lgenPhaseLists := {t\_myPhaseList\_Instance\_1, t\_myPhaseList\_Instance\_1}

},

{

name := "term",

idx := -1,

lgenIndexes := {0,1},

lgenPhaseLists := {t\_myPhaseList\_Instance\_1, t\_myPhaseList\_Instance\_1}

}

},

scenarioPhaseLists :=

{

t\_myPhaseList\_Instance\_1,

t\_myPhaseList\_Instance\_1

},

groupPhaseList := t\_myPhaseList\_Instance\_1,

phaseFinishAction := null,

groupFinishAction := null,

isRunningVarIdx := -1,

internalAdjust := false

}  
  
 template EPTF\_ExecCtrl\_PhaseList\_Instance t\_myPhaseList\_Instance\_1 :=

{

name := “GeneralPhaseList”,

phases :=

{

{

name := "preamble",

enabled := true,

state := IDLE

},

{

name := "loadgeneration",

enabled := true,

state := IDLE

},

{

name := "postamble",

enabled := true,

state := IDLE

}

},

actualPhaseIndex := 0

}

For each scenario a phase list instance is stored in the scenarioPhaseList field, where state of the actual phase is stored. For each LGen a scenario is running on there is also a phase list instance in the lgenPhaseLists field. In addition to these there is an overall phase list instance for the whole scenario group, where we keep track of the overall state of the phase execution.

The phases for each scenario can be adjusted individually.

The phase states are the following:

* IDLE
* RUNNING
* SKIPPED
* STOPPING
* FINISHED

## Execution

Only that scenario group can be started, where none of the scenarios are running.

Pseudo code:

1. startScenarioGroup(in ScenarioGroupName)
   * + - 1. for each phase in phase list

for each scenario

for each LGen the scenario is running on

send startPhase(name, enabled) request via ExecCtrl i/f to ExecCtrlClient to each LGen

for each scenario

for each LGen the scenario is running on

wait for phaseFinished response

set the phase state accordingly

set the main phase state in the main phase list

go to next phase

## ExecCtrl/ExecCtrlClient Phase PDUs

This section describes the PDUs that are sent between the ExecCtrl and ExecCtrl client.

### Phase start request

Direction: ExecCtrl -> ExecCtrlClient

Purpose: With this request a certain phase can be started on a given Scenario instance on a specific load generator.

Contents:

* phaseName - \*charstring\* - phase to be started
* scenarioInstanceName - \*charstring\* - name of the scenario instance
* state - <EPTF\_ExecCtrl\_Phase\_States> - the action that must be performed can be either RUNNING or SKIPPING
* msgID - <EPTF\_IntegerList> - msgID contains the following indices respectively: scenario group idx, scenario idx, lgen idx, phase idx

### Phase stop request

Direction: ExecCtrl -> ExecCtrlClient

Purpose: With this request a certain phase can be stopped on a given Scenario instance on a specific load generator. After the current phase is stopped, the next phase will start (in AUTOMATIC mode).

Contents:

* phaseName - \*charstring\* - phase to be stopped
* scenarioInstanceName - \*charstring\* - name of the scenario instance
* state - <EPTF\_ExecCtrl\_Phase\_States> - the action that must be performed can be only STOPPING
* msgID - <EPTF\_IntegerList> - msgID contains the following indices respectively: scenario group idx, scenario idx, lgen idx, phase idx

### Phase stop test request

Direction: ExecCtrl -> ExecCtrlClient

Purpose: With this request a certain phase can be stopped on a given Scenario instance on a specific load generator. After the current phase is stopped, the next phase will not start (even in AUTOMATIC mode).

Contents:

* phaseName - \*charstring\* - phase to be stopped
* scenarioInstanceName - \*charstring\* - name of the scenario instance
* state - <EPTF\_ExecCtrl\_Phase\_States> - the action that must be performed can be only IDLE
* msgID - <EPTF\_IntegerList> - msgID contains the following indices respectively: scenario group idx, scenario idx, lgen idx, phase idx

### Phase status response

Direction: ExecCtrlClient -> ExecCtrl

Purpose: With this response the ExecCtrlClient can report the state of a certain scenario in a certain phase.

Contents:

* phaseName - \*charstring\* - phase to be stopped
* scenarioInstanceName - \*charstring\* - name of the scenario instance
* state - <EPTF\_ExecCtrl\_Phase\_States> - the state of the scenario in given phase.
* msgID - <EPTF\_IntegerList> - msgID contains the following indices respectively: scenario group idx, scenario idx, lgen idx, phase idx

## GUI for ScenarioGroups and PhaseLists

In the Execution Control->Phase Lists tab there separate tab panels for each declared phase list. For an example screen see Figure 2.

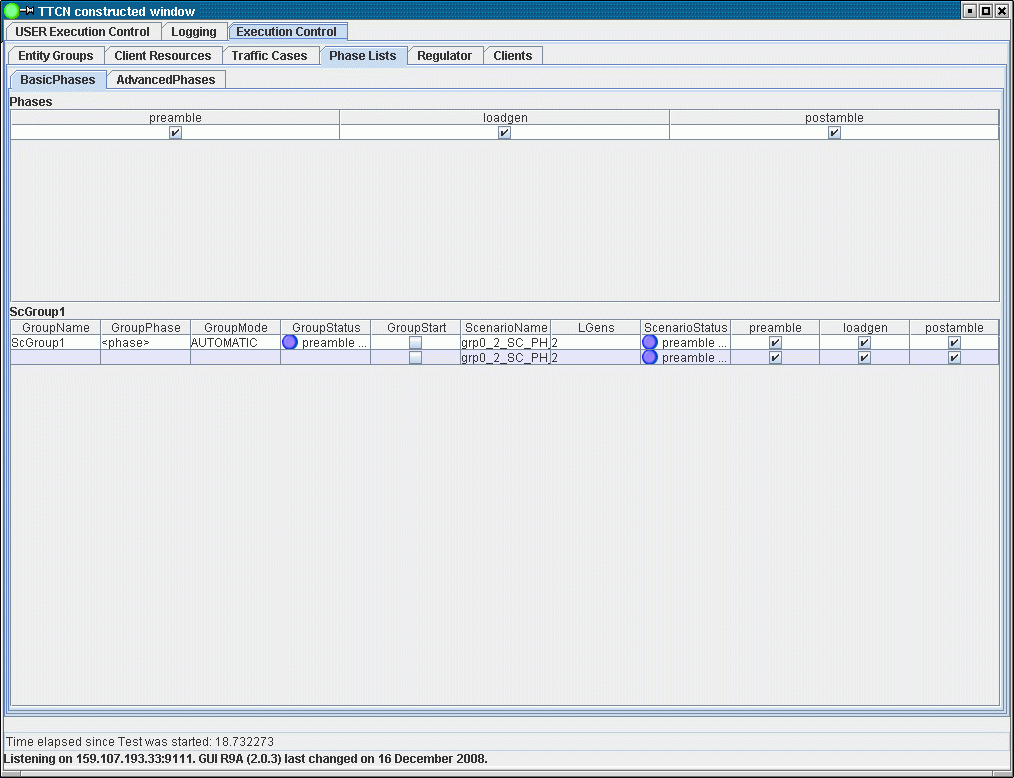


Figure 2 Phase list tab pane

On the phase lists tab those declared scenario groups can be seen that are using the given phase list.

* GroupName: Name of the scenario group
* GroupPhase: The desired start phase can be set here. The name of the phase must be typed in correctly. The phase can be modified only if the scenario is on IDLE state.
* GroupMode: The mode of the execution can be set here. The name of the mode must be typed in correctly. Available choices: “AUTOMATIC”, “automatic”, “MANUAL”, “manual”. The phase can be modified only if the scenario is on IDLE state.
  + In AUTOMATIC mode the phases will be executed consecutively without human intervention
  + In MANUAL mode the execution will be stopped after a phase finished. To start the next phase the user has to start the scenario group again.
* GroupStatus: This led provides you the actual phase and the actual state of the phase in the scenario group.
* GroupStart: By checking in the checkbox in the first row of the scenario group, the whole group can be started. By checking out, the scenario group is stopped.
* ScenarioName: Each row contains one scenario member of the group.
* ScenarioStatus: This led provides you the actual phase and the actual state of the phase regarding one scenario.

The following check boxes are for setting if a certain phase must be executed or skipped in case of a scenario.

# Error messages

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

**unknown entityType: <entity\_type\_name>**

No such entity type.

# Warning messages

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

**outOfResources when creating initial entity group: <group\_name> <entity\_type\_name> <entities\_created> <entities\_left>**

Cannot create the full entity group due to insufficient resources, i.e. the group may contain less element then originally desired.

# Examples

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

* execctrld.cfg
* ExecCtrl\_Demo.ttcn
* AppLib\_A\_Definitions.ttcn
* AppLib\_A\_Functions.ttcn
* ExecCtrlUIHandler\_Demo.ttcn
* ExecCtrlLGenPool\_Demo.ttcn
* ExecCtrlLGenPool\_Demo.cfg

## Configuration file

The used configuration file (execctrld.cfg) is for the ExecCtrl example is placed in the demo directory. You may change Runtime GUI related parameters, like:

EPTF\_CLL\_UIHandler\_Widget\_Functions.tsp\_xtdp\_listen\_addr := "127.0.0.1"; // RuntimeGUI is running on localhost

EPTF\_CLL\_UIHandler\_Widget\_Functions.tsp\_xtdp\_listen\_port := 55678;

The configuration file ExecCtrlLGenPool\_Demo.cfg is a simple setup to demonstrate how to configure ExecCtrl when LGen pools are used.

## Demo Module

The demo module ExecCtrl\_Demo.ttcn) illustrates a typical usage of the execution control feature, while ExecCtrlUIHandler\_Demo.ttcn demonstrates the usage of the execution control UI handler feature. The module ExecCtrlLGenPool\_Demo.ttcn shows how to register the LGen creator function and how to use LGen pools. It is also demonstrated how to create a component that is not an LGen.

Example Load Generator component definition using feature “A” with ExecCtrlClient:

**type component** ExecCtrl\_Demo\_CT **extends** EPTF\_LGenBase\_CT, EPTF\_ExecCtrlClient\_CT, LGen\_A\_CT {

**var integer** v\_myBIdx;

}

The initialization of ExecCtrlClient feature should called only after all the extended application library initializations (incl. virtual application libraries) in f\_LGen\_Demo\_init():

f\_EPTF\_ExecCtrlClient\_init\_CT(pl\_selfName, pl\_ExecCtrlRef);

The main altstep of the feature should put into the main event handler altstep, or, alternatively, activate as default altstep:

**alt** {

[] t.**timeout** {}

}

MTC should extend and initialize UIHandler in order to provide user interface for interactive features (limited functionality in the current drop):

type component myMTC\_CT extends EPTF\_Base\_CT, EPTF\_CLL\_UIHandler\_CT {}

//…

**map**(self:EPTF\_guiPortIf,system:EPTF\_guiPortIf);

f\_EPTF\_CLL\_UIHandler\_init\_CT("ExecCtrlDemo\_UIHandler", **true**, ""/\*pl\_wndLayout\*/);

f\_EPTF\_CLL\_UIHandler\_clearGUI();

f\_EPTF\_CLL\_UIHandler\_addWindow();

f\_ExecCtrl\_Demo\_initGui();

Starting ExecCtrl from MTC:

**var** EPTF\_ExecCtrl\_CT ExtCtrl := EPTF\_ExecCtrl\_CT.**create**;

ExtCtrl.**start**(f\_EPTF\_ExecCtrl\_behavior("ExecCtrl", tsp\_nrOfLGens));

# 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. 198 17-CNL 113 512 Uen  
   EPTF Core Library for TTCN-3 toolset with TITAN, User Guide
5. 19/155 16-CNL 113 512  
   EPTF Core Library Execution Control, Function Description
6. 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)
7. 7/155 16-CNL 113 512  
   EPTF Core Library LGenBase, Function Description