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

EPTF AppLib MBT for TitanSim, Function Specification

Contents

[1 Introduction 3](#_Toc272742449)

[1.1 Revision history 3](#_Toc272742450)

[1.2 How to Read this Document 3](#_Toc272742451)

[1.3 Scope 3](#_Toc272742452)

[1.4 References 3](#_Toc272742453)

[1.5 Typographical conventions 4](#_Toc272742454)

[1.6 Abbreviations 4](#_Toc272742455)

[1.7 Terminology 4](#_Toc272742456)

[2 General 4](#_Toc272742457)

[3 Functional specification 6](#_Toc272742458)

[3.1 Overview 6](#_Toc272742459)

[3.2 MBT Applib PDUs 7](#_Toc272742460)

[3.2.1 EPTF\_MBT\_ConfigRequest 7](#_Toc272742461)

[3.2.2 EPTF\_MBT\_ConfigResponse 7](#_Toc272742462)

[3.2.3 EPTF\_MBT\_TestStepRequest 8](#_Toc272742463)

[3.2.4 EPTF\_MBT\_TestStepResponse 8](#_Toc272742464)

[3.2.5 EPTF\_MBT\_UserRequest 8](#_Toc272742465)

[3.2.6 EPTF\_MBT\_CommandRequest 9](#_Toc272742466)

[3.2.7 EPTF\_MBT\_CommandResponse 9](#_Toc272742467)

[3.3 Behavior 9](#_Toc272742468)

[3.3.1 MBT Behavior 9](#_Toc272742469)

[3.4 TestSteps 10](#_Toc272742470)

[3.4.1 Invoke TestStep 10](#_Toc272742471)

[3.4.2 Invoke User Function 10](#_Toc272742472)

[3.4.3 Send User Response 10](#_Toc272742473)

[3.5 Events 11](#_Toc272742474)

[3.5.1 Incoming User Request 11](#_Toc272742475)

[3.5.2 Incoming Test Step Request 11](#_Toc272742476)

[3.6 Functions 11](#_Toc272742477)

[3.6.1 f\_EPTF\_MBT\_init() 11](#_Toc272742478)

[3.6.2 f\_EPTF\_MBT\_initLGenFsm() 12](#_Toc272742479)

[3.6.3 fcb\_EPTF\_MBT\_entityGroupCreated() 12](#_Toc272742480)

[3.6.4 fcb\_EPTF\_MBT\_customUserFunction() 12](#_Toc272742481)

[3.6.5 EPTF\_MBT\_LGen\_CT: The MBT Component type 12](#_Toc272742482)

[3.7 FSMs 13](#_Toc272742483)

[3.8 TTCN Scripter for Confromiq Qtronic 14](#_Toc272742484)

[3.8.1 Parameter: eptfEnable 14](#_Toc272742485)

[3.8.2 Parameter: extendsComponent 15](#_Toc272742486)

[3.8.3 Parameter: eptfBidirectionalPort 15](#_Toc272742487)

[3.8.4 Parameter: eptfInboundPort 15](#_Toc272742488)

[3.8.5 Parameter: eptfOutboundPort 15](#_Toc272742489)

# 

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2009-12-16 | PA1 | First draft version | EANTWUH |
| 2010-09-20 | A | Final revision | EANTWUH |
|  |  |  |  |
|  |  |  |  |

## How to Read this Document

This is the Function Specification for the MBT application library of the Ericsson Performance Test Framework (TitanSim). The modules are developed for the TTCN-3 Toolset ‎[1] with TITAN ‎[2]. This document should be read together with Product Revision Information ‎[3] of the application library. Additionally, the TitanSim Core Load Library ‎[4] should be consulted for understanding the core functionalities of the TitanSim used by the application library.

## Scope

The purpose of this document is to specify the content of the MBT Application Library.

## References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. Part 1: Core Language
2. 1/198 17-CRL 113 200 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 109 21-CNL 113 659-1 Uen  
   EPTF MBT Applib for TitanSim, Product Revision Information
4. 109 21-CNL 113 512-2 Uen   
   TitanSim CLL for TTCN-3 toolset with TITAN, Product Revision Information
5. EPTF MBT Application Library for TTCN-3 Toolset with TITAN, Reference Guide  
   <http://ttcn.ericsson.se/products/libraries.shtml>  
   Alternatively, please consult the doc/apidoc directory of the released application library.

## 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 (‎1.7) of this document, then these occurrences are marked with an initial arrow.

## Abbreviations

Applib Application Library

CLL Core Load Library

EPTF Ericsson Performance Test Framework

FSM Finite State Machine

MBT Modell Based Testing

TTCN-3 Test and Test Control Notation version 3

SUT System Under Test

TitanSim New synonym for the EPTF framework

## Terminology

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

*TitanSim Appliaction Library* is the application-specific part of the TitanSim software.It provides load generation functionalities belonging to specific protocols or products.

# General

Model-based testing is software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (SUT).

Model-based testing is the automation of black-box test design. A model based testing tool uses various test generation algorithms and strategies to generate tests from a behavioral model of the SUT.



Figure 1 Model Based Testing

The model is usually an abstract, partial presentation of the system under test's desired behavior. The model must be concise and precise: concise so that it does not take too long to write and so that it is easy to validate with respect to the requirements but precise enough to describe the behavior that is to be tested.

Test cases (including test data and oracles) can be automatically generated from the model using a model-based testing tool. The test engineer can also control the tool to focus the testing effort and manage the number of tests that are generated. The test cases derived from this model are functional tests on the same level of abstraction as the model. These test cases are collectively known as the abstract test suite. The abstract test suite cannot be directly executed against the system under test because it is on the wrong level of abstraction.

The tests produced from the model are abstract tests, so they must be transformed into executable tests. Therefore an executable test suite must be derived from the abstract test suite that can communicate with the system under test. This is done by mapping the abstract test cases to concrete test cases suitable for execution. This also requires some input from the test engineer, but most model-based testing tools provide assistance with this process.

Since developing a test harness is expensive one might think it would be a good idea to build a generic test harness, which can then be reused for several project. The development of such a generic test harness should be started by defining the higher-level interface of the test harness and the structure of the abstract messages. Since this interface heavily depends on the model, which in turn is very dependent on the test purpose the model was developed for, it is very hard to come up with a good interface that can be reused with as few limitations as possible.

One solution can be to re-use the application libraries that were developed for the TitanSim framework. These libraries have high level abstract interfaces. They can be extended with user code and they implement a lot of protocol specific functionality. In the following section we describe the MBT application library with which it is possible to use other TitanSim application libraries as test harness for the MBT approach.

# Functional specification

## Overview

The architecture of a TitanSim based test harness can be seen in Figure 2. The applib provides an external “MBT Applib Interface” on which it is possible to control and communicate with the test harness. This interface carries primitives that makes it possible to invoke and execute test steps of the application libraries. The events reported back by the applictions are also sent on this interface back to the MSC Executor.

The applib can be extended so that a user defined port with user defined messages can be handled with the help of an applib. This “User mapping” code is responsible to map the incoming messages to either TitanSim functions/test steps or user defined functions. The applib events can be also caught and mapped to user defined messages, which can be sent back to the Tester component.



Figure 2 TitanSim based Test Harness

The MBT applib provides a simple FSM that can catch the reported events from the applibs and can execute all desired test steps registered by the applibs. The behavior of the FSM can be controlled via functions in the “User mapping” code, or via MBT Applib PDUs on the “MBT Applib Interface”

The user glue code is where the applibs are initialized and put together. This part can also contain some user written support functions to further extend the test harness.

There are two main approaches to use this test harness arrangement:

1. The test harness can be controlled via the “MBT Applib Interface”. In this case, the Tester must communicate via MBT Applib PDUs, that is the model must be built so that it communicates with the environment using MBT applib PDUs.
2. The other alternative is that the model is using its own PDUs to interact with the environment. But then, the test harness must be extended with “User mapping” code that can handle these messages and can translate to MBT Applib primitives.

## MBT Applib PDUs

The MBT Applib is using the following PDUs.

### EPTF\_MBT\_ConfigRequest

Direction: Tester → Test Harness

Parameters:

* entityGroupName: charstring  
  Name of the entity group to be created
* noEntities: integer  
  Number of entities to be created in the group
* behaviors: EPTF\_CharstringList  
  List of behavior that should be allocate to the group.
* fsmName: charstring  
  Name of the FSM to be activated for each member of the entity group.

Description:

The PDU is will declare an entity type in the test harness with the behaviors given as parameter. Then an entity group is created of this entity type. The group will contain noEntites number of entities. For each entity an FSM is activated. The type of FSM is specified in the fsmName parameter. Finally the vf\_EPTF\_MBT\_entityGroupCreate call back function is invoked. This call back function must be implemented by the user and its main purpose is to init the behavior contexts and other databases as the user wish.

### EPTF\_MBT\_ConfigResponse

Direction: Test Harness → Tester

Parameters: -

Description:

This PDU indicates that the recently received EPTF\_MBT\_ConfigRequest PDU was processed and executed.

### EPTF\_MBT\_TestStepRequest

Direction: Test Harness → Tester

Parameters:

* stepName: charstring  
  Name of the requested test step
* stepArgs: EPTF\_Integer\_List  
  Container of the test step arguments
* addr: FsmAddr  
  Address of the FSM which will execute the test step.

Description:

The purpose of this PDU is to execute a test step on an FSM. The stepName and stepArgs parameters define the test step to be invoked, while the addr record selects an FSM of an entity in an entity group.

### EPTF\_MBT\_TestStepResponse

Direction: Tester → Test Harness

Parameters:

* bName: charstring  
  Name of the behavior
* iName: charstring  
  Name of the event

Description:

This PDU indicates that an event was received by one of the FSMs.

### EPTF\_MBT\_UserRequest

Direction: Test Harness → Tester

Parameters:

* functionName: charstring  
  Name of the requested test step
* functionArgs: EPTF\_Integer\_List  
  Container of the test step arguments

Description:

The purpose of this PDU is to execute a function that was registered into the CLL’s LGenBase. The functionName and functionArgs parameters define the test step to be invoked.

### EPTF\_MBT\_CommandRequest

Direction: Tester → Test Harness

Parameters:

* quit: QuitCommand  
  An empty record indicating that the test harness should start cleaning up and then exit.

Description:

This PDU makes the Test Harness to execute one of its management functions. Currently there is only one implements: the quit command, which will start the clean up process and finally the test harness exits.

### EPTF\_MBT\_CommandResponse

Direction: Test Harness → Tester

Parameters:

* ready: ReadyRespone  
  An empty record indicating that the test harness finished with the initialization and is waiting for requests.
* quitAck: QuitAckResponse  
  An empty record indicating that the test harness has received the quit command and will start executing it.

Description:

The PDU contains indication on the management interface of the test harness. Two fields can be chosen: either the ready indication or the quitAck reponse.

## Behavior

### MBT Behavior

Constant: c\_EPTF\_MBT\_myBName

Name: “MBT\_behavior”

Description:

This is the main MBT applib behavior which will create the required context space and will be used to declare events.

## TestSteps

### Invoke TestStep

Constant: c\_EPTF\_MBT\_stepName\_invokeTestStep

Name: “EPTF\_MBT\_stepName\_invokeTestStep”

Function: f\_EPTF\_MBT\_TestStep\_invokeTestStep

Arguments:

Description:

This test step fetches a EPTF\_MBT\_TestStepRequest PDU from the v\_EPTF\_TestStepRequest component variable looks up the test step instance based on the requested name and the executes it with the parameters described in the PDU.

### Invoke User Function

Constant: c\_EPTF\_MBT\_stepName\_invokeUserFunction

Name: “EPTF\_MBT\_stepName\_invokeUserFunction”

Function: f\_EPTF\_MBT\_TestStep\_invokeUserFunction

Arguments: -

Description:

This test step fetches a UserRequest PDU from the v\_EPTF\_UserRequest component variable looks up the function pointer based on the requested name and the executes it with the parameters described in the PDU.

### Send User Response

Constant: c\_EPTF\_MBT\_stepName\_sendUserResponse

Name: “EPTF\_MBT\_stepName\_sendUserResponse”

Function: f\_EPTF\_MBT\_TestStep\_sendUserResponse

Arguments: -

Description:

This test first examines whether the vf\_EPTF\_MBT\_createUserResponse call back function pointer is set to a valid function. If it is, then it will call that function. The purpose of this callback function is to be able to create user mapping code that can transform any applib events to user defined PDU response.

If this vf\_EPTF\_MBT\_createUserResponse pointer is set to null, the test step will create an EPTF\_MBT\_TestStepReponse PDU automatically based on the caught LGenBase/Applib event.

## Events

### Incoming User Request

Constant: c\_EPTF\_MBT\_inputName\_incomingUserRequest

Name: “EPTF\_MBT\_inputName\_incomingUserRequest”

Behavior: c\_EPTF\_MBT\_myBName

Index: 0

Description:

Whenever an EPTF\_MBT\_UserRequest PDU is received the MBT Applib will report this event to the FSM which is selected in the PDU’s addr parameter.

### Incoming Test Step Request

Constant: c\_EPTF\_MBT\_inputName\_incomingTestStepRequest

Name: “EPTF\_MBT\_inputName\_incomingTestStepRequest”

Behavior: c\_EPTF\_MBT\_myBName

Index: 1

Description:

Whenever an EPTF\_MBT\_TestStepRequest PDU is received the MBT Applib will report this event to the FSM which is selected in the PDU’s addr parameter.

## Functions

### f\_EPTF\_MBT\_init()

Parameters:

* pl\_selfName: chartstring  
  The name of the component instance
* pl\_selfId: integer  
  The unique id for the component type’s instance
* pl\_entityNamePrefix: charstring  
  The name of all entity groups will get this prefix

Description:

The function first initializes the extended component types then declares the MBT behavior type with the corresponding test steps and MBT applib events. It also sets up the MBT logging masks. Activates the MBT PDU handlers and finally it maps the MBT port.

### f\_EPTF\_MBT\_initLGenFsm()

Parameters:

* p\_userFunc: fcb\_EPTF\_MBT\_customUserFunction
* p\_entityGroupCreatedFunc: fbc\_EPTG\_MBT\_entityGroupCreated

Description:

This function sets the call back function pointers in the MBT component instance and declares the MBT FSM.

### fcb\_EPTF\_MBT\_entityGroupCreated()

Parameters:

* p\_req: EPTF\_MBT\_ConfigRequest

Description:

This is a call back function signature. The function that implements this signature can be used to fill in the context databases after an entity group was created with the help of an EPTF\_MBT\_ConfigRequest PDU.

### fcb\_EPTF\_MBT\_customUserFunction()

Parameters:

* pl\_ptr: EPTF\_LGenBase\_TestStepArgs

Description:

The purpose of this callback function is to be able to create user mapping code that can transform any applib events to user defined PDU response.

### EPTF\_MBT\_LGen\_CT: The MBT Component type

Ports:

* EPTF\_MBT\_PCO: EPTF\_MBT\_PortType

Variables:

* vf\_EPTF\_MBT\_entityGroupCreated: fcb\_EPTF\_MBT\_entityGroupCreated  
  Function poninter pointing to a function which will be called after an entity group was created.
* vf\_EPTF\_MBT\_createUserResponse: fcb\_EPTF\_MBT\_customUserFunction  
  Function pointer pointing to a function which will be called when an event was reported and a response must be generated back to the Tester component.
* v\_EPTF\_MBT\_TestStepRequest: EPTF\_MBT\_TestStepRequest  
  The last received EPTF\_MBT\_TestStepRequest PDU.
* v\_EPTF\_MBT\_ConfigRequest: EPTF\_MBT\_ConfigRequest  
  The last received EPTF\_MBT\_ConfigRequest PDU.
* v\_EPTF\_MBT\_CommandRequest: EPTF\_MBT\_CommandRequest  
  The last received EPTF\_MBT\_CommandRequest PDU.
* v\_EPTF\_MBT\_UserRequest : EPTF\_MBT\_UserRequest  
  The last received EPTF\_MBT\_UserRequest PDU.

## FSMs

The main FSM of the MBT applib is described here in the following Figure 3.



Figure 3 MBT FSM

The FSM is very simple: it consists of only one state. In this main state it is listening to events that are generated by the MBT applib when an MBT PDU is received and the FSM calls the corresponding handler test steps.

Any other event must come from other applibs. These events must be reported back to the Tester component either by generating an EPTF\_MBT\_TestStepResponse PDU or by creating a user defined PDU. It is the f\_EPTF\_MBT\_sendUserResponse test step which is responsible for this task.

## TTCN Scripter for Confromiq Qtronic

The MBT Applib has its own specialized TTCN Scripter for the Conformiq Qtronic MBT tool. It is based on the scripter that is shipped with Conformiq Qtronic 2.1.1, but it is extended with some additional parameters

The sources can be found in the src/Qtronic/TTCNScripter directory

The compiled scripter’s JAR file is in /demo/TTCNScripter/TTCNScripter.jar

The MBT aplpib specific parameters are collected on a separate page (see Figure 4)

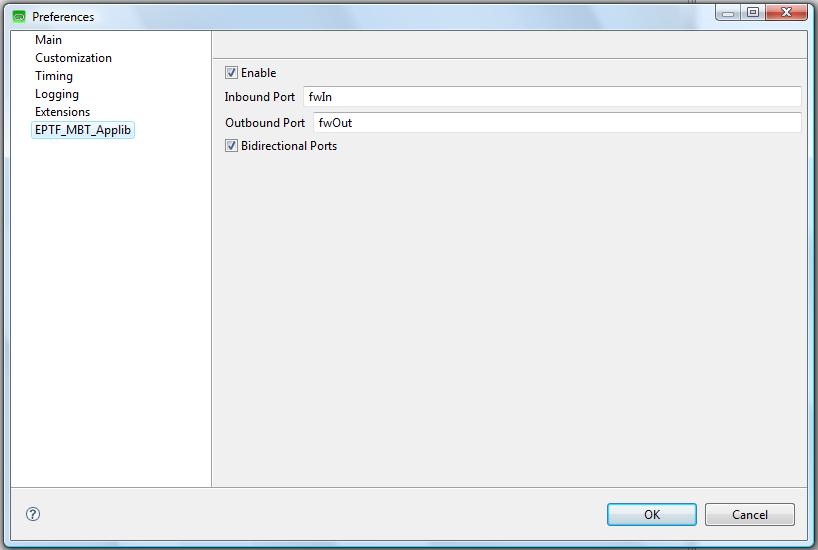


Figure 4 MBT Applib settings for Qtronic TTCN Scripter

### Parameter: eptfEnable

Type: Boolean

Effect:

When this check box is turned on it changes the a generated code in order to be able to seamlessly compile TitanSim’s MBT applib and the generated code together:

* In the generated test harness skeleton the EPTF\_MBT\_PCO.send and EPTF\_MBT\_PCO.receive statements are used for sending and receiving.
* The generated main module will import the generated test harness skeleton
* Port definition is not generated for the Applibs MBT port, it can be imported instead.
* Type definitions are not generated for EPTF types. (All type definitions that start with “EPTF\_” are skipped during code generation)
* Port mapping/unmapping code is not generated.

### Parameter: extendsComponent

Type: String

Effect:

When this string is not empty, the generated tester component will extend the component type which is given here.

### Parameter: eptfBidirectionalPort

Type: Boolean

Effect:

During code generation all types in a port will have be “inout”s. That is, there will be no difference between inbound and outbound messages. The same port definition can be used to send and receive that given message types.

### Parameter: eptfInboundPort

Type: String

Effect:

Here it can be specified which port is used as inbound for the test harness in the model.

### Parameter: eptfOutboundPort

Type: String

Effect:

Here it can be specified which port is used as outbound for the test harness in the model.