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

EPTF CLL UIHandler, User Guide

Contents

[1 Introduction 2](#_Toc468710753)

[1.1 Revision history 2](#_Toc468710754)

[1.2 About this Document 2](#_Toc468710755)

[1.2.1 How to Read this Document 2](#_Toc468710756)

[1.2.2 Abbreviations 3](#_Toc468710757)

[1.2.3 Terminology 3](#_Toc468710758)

[1.2.4 System Requirements 3](#_Toc468710759)

[1.3 Description of Required External Features outside CLL 3](#_Toc468710760)

[2 UIHandler 4](#_Toc468710761)

[2.1 Overview 4](#_Toc468710762)

[2.2 Compilation dependencies 5](#_Toc468710763)

[2.3 Usage dependencies 6](#_Toc468710764)

[3 Workflow 7](#_Toc468710765)

[3.1 Installation 7](#_Toc468710766)

[3.2 Configuration 7](#_Toc468710767)

[3.3 Screen layout design 10](#_Toc468710768)

[3.3.1 Static design 10](#_Toc468710769)

[3.3.2 Dynamic design 10](#_Toc468710770)

[3.4 Runtime test configuration design (deployment) 11](#_Toc468710771)

[4 DataSource elements 11](#_Toc468710772)

[5 Simulation mode 11](#_Toc468710773)

[6 Examples 12](#_Toc468710774)

[6.1 Configuration file 12](#_Toc468710775)

[6.2 Demo Module 12](#_Toc468710776)

[7 References 13](#_Toc468710777)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-07 | PA1 | First draft version | ELSZSKU |
| 2007-12-04 | PA2 | Template update, headless mode | EZSOSZA |
| 2008-01-09 | PA3 | General update, server mode | EZSOSZA |
| 2008-02-01 | PA4 | Updated after review | EZSOSZA |
| 2008-02-06 | PA5 | VariableUI | EZSOSZA |
| 2008-02-14 | PA6 | CLL feature UIs | EZSOSZA |
| 2008-02-22 | PA7 | Shadow GUI support | EZSOSZA |
| 2009-04-23 | PB1 | New tsp to support Cleanup | EISTFAL |
| 2011-02-09 | PC1 | XTDP Authentication+DataSource+doc update | ETHJGI |
| 2011-05-20 | PD1 | Condition is added to DataSource | ETHJGI |
| 2011-08-22 | PE1 | DataSource removed | EBALLUG |
| 2012-05-25 | PF1 | Simulation mode added | EMIHMIK |
| 2012-07-26 | PG1 | ProgressBar dataSource added | ETHJGI |
| 2016-04-19 | PH1 | External dependencies added. | EZOLZSI |
| 2016-05-18 | PH3 | System Requirement added + 3PP product list updated. | ESZILSZ |
| 2016-06-09 | H | Updated for release. | ESZILSZ |
| 2016-08-22 | PJ1 | Updated with external dependencies | EIMRENA |
| 2016-10-12 | PJ2 | Html2Canvas added to 3PPs | EIMRENA |
| 2016-10-12 | PJ3 | Corrected Html2Canvas description | EIMRENA |
| 2016-10-15 | PJ4 | Minor edit | EVAURBA |
| 2016-10-17 | PJ5 | Editorial changes | EIMRENA |
| 2016-12-02 | J | Updated for release | EIMRENA |

## About this Document

### How to Read this Document

This is the User Guide for the UIHandler of the Ericsson Performance Test Framework (TitanSim), Core Library (CLL). EPTF CLL is developed for the TTCN-3 [1] Toolset with TITAN [2]. This document should be read together with the Function Description of the UIHandler feature [6]. For more information on the EPTF CLL please consult the Product Revision Information, the User’s Guide [3] and the Function Specification [4] of the TitanSim.

### Abbreviations

CLI Command Line Interface

CLL Core Library

EPTF Ericsson Performance Test Framework

TitanSim New synonym for the EPTF framework

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

### Terminology

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

### System Requirements

In order to use the UIHandler feature the system requirements listed in EPTF CLL User Guide [4] should be fulfilled.

The minimum required web browser version for using the Browser GUI is Mozilla Firefox 38.5.2 Extended Support Release (module name in AFS: firefox/38.5.2esr).

## Description of Required External Features outside CLL

|  |  |  |
| --- | --- | --- |
| **Product Number** | **Product Name** | **Product Description** |
| 1/CAX 105 6334 | RapidXml 1.13 | RapidXml is required by the current web based GUI component. It is used to generate web pages based on templates/descriptions stored in XML files |
| 33/CAX 105 3414 R1A | jQuery 2.2.3 | It provides a rich function set including, but not limited to HTML document traversal and manipulation, event handling, animation, and AJAX calls. Other 3rd parties depend on this library as well. |
| 2/CAX 105 6945 | Flot 0.8.3 | Flot is used by BrowserGUI and WebGUI features to display charts (e.g.: CPS charts). It replaces a similar 3rd party component that was removed from TitanSim due to licensing problems. |
| 1/CAX 105 8146 | Html2Canvas 0.4.1 | Html2Canvas is used to convert any html element to canvas, which then can be saved as an image file. Used by the UIHandler (BrowserGUI) and WebGUI features. |

# UIHandler

## Overview

Users of TitanSim may want to interactively display their data on a user interface or they want to control the behaviour of their tests through such interface. The UIHandler feature provides support for this.

The EPTF\_CLL\_UIHandlerClient\_CT component may have EPTF Variables and it can connect these variables to widgets on a runtime GUI connected to an UIHandler. When the variable changes, the content of the connected widget will be changed automatically. When the user changes the content of a connected widget, its new value will be automatically forwarded to the associated variable.

The EPTF\_CLL\_UIHandler\_VariableUI\_CT can be used just like EPTF\_CLL\_UIHandlerClient\_CT, but it provides some functions that can be used to handle masses of variable – widget subscriptions.

It also provides functions to change the layout of the GUI.

Usage in steps:

* Create an UIHandler component, and map the necessary ports.
* Initialize the UIHandler component.
* Create your component that extends EPTF\_CLL\_UIProviderClient\_CT and implicitly the EPTF\_Var\_CT.  
  Initialize the UIHandlerClient component.
* Create and/or modify the runtime GUI configuration where you want to display your data.
* Create your variables.
* Order the UIHandler via functions of the UIHandlerClient to subscribe to your variables, and connect these variables to widgets. If you want to connect your variables to more than one widget, you can do it.
* Run your test logic.

The feature can operate in either as a client and a server too, to make connection with the Runtime GUI.  
The UIhandler, if used with headless mode activated, is capable of seamlessly continue operation even if the connection to the runtime GUI is lost. Alternately you can use a web browser to view the GUI. For more details see [3].

## Compilation dependencies

The EPTF CLL UIHandler API includes the following files:

* UIHandler
  + EPTF\_CLL\_UIHandlerCLI\_CommandDefinitions.ttcn  
    Contains the CLI command patterns.
  + EPTF\_CLL\_UIHandlerCLI\_Definitions.ttcn  
    Type definitions for the CLI feature of the UIHandler.
  + EPTF\_CLL\_UIHandlerCLI\_Functions.ttcn  
    Function definitions for the CLL feature of the UIHandler.
  + EPTF\_CLL\_UIHandlerClient\_Definitions.ttcn  
    Definitions for client layer of the UIHandler.
  + EPTF\_CLL\_UIHandlerClient\_Functions.ttcn  
    Private and public function definitions for client layer of the UIHandler.
  + EPTF\_CLL\_UIHandlerVariableUI\_Definitions.ttcn  
    Definitions for the VariableUI.
  + EPTF\_CLL\_UIHandlerVariableUI\_Functions.ttcn  
    Function definitions for the VariableUI.
  + EPTF\_CLL\_UIHandlerBrowser\_Functions.ttcn  
    These functions realizes the web browser GUI functionalities.
  + EPTF\_CLL\_UIHandler\_CLIClientDefinitions.ttcn  
    Definitions for client layer of the UIHandlerCLI.
  + EPTF\_CLL\_UIHandler\_ConvertFunctions.ttcn  
    These functions are used in data display converting data.
  + EPTF\_CLL\_UIHandler\_Definitions.ttcn  
    Type definitions for the EPTF\_CLL\_UIHandler\_CT component.
  + EPTF\_CLL\_UIHandler\_MsgDefinitions.ttcn  
    Message structures and data types.
  + EPTF\_CLL\_UIHandler\_PrivateFunctions.ttcn  
    Internally used functions of EPTF\_CLL\_UIHandler\_CT component.
  + EPTF\_CLL\_UIHandler\_WidgetFunctions.ttcn  
    Functions of the EPTF\_CLL\_UIHandler\_CT component to manipulate runtime GUI.
  + EPTF\_CLL\_UIHandler\_XSD\_XTDPTemplateDefinitions.ttcn  
    Message types and data structures used in the communication with the runtime GUI using XSD.
  + EPTF\_CLL\_UIHandler\_XTDPTemplateDefinitions.ttcn  
    Message types and data structures used in the communication with the runtime GUI using XTDP Test Port.

## Usage dependencies

The UIHandler feature is part of the EPTF Core Library (CLL). It relies on several features of the CLL. To use the UIHandler the user has to obtain the respective files from the following features:

* Base
* Common
* FreeBusyQueue
* HashMap
* Logging
* Semaphore
* Transport
* Variable

UIHandler also requires files from TestPorts, ProtocolModules

* XTDPasp\_CNL113494
* Common\_Components/Abstract\_Socket\_CNL113384
* Common\_Components/Socket\_API\_CNL113686
* XTDP\_CNL113663
* XSDASN\_CNL113474
* TELNETasp\_CNL113320
* IPL4asp\_CNL113531
* TCCUsefulFunctions\_CNL113472

# Workflow

## Installation

Since EPTF\_CLL\_UIHandler 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 noted in the respective sections, the following are needed to use EPTF\_CLL\_UIHandler:

* Copy the files listed in Section [2.3] to the directory of the test suite or create symbolic links to them.
* Import the UIHandler demo or write your own application using UIHandler
* Create Makefile or modify the existing one. For more details see the relevant section of [2].
* Edit the configuration file according to your needs, see following section [3.2].

## 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].

This UIHandler feature defines TTCN-3 module parameters as defined in [2] clause 4. Actual values of these parameters – when no default value or a different from the default actual value wished to be used – shall be given in the [MODULE\_PARAMETERS] section of the configuration file.

This UIHandler feature defines the following module parameters:

**tsp\_EPTF\_GUI\_Main\_Window\_WidgetId,  
tsp\_EPTF\_GUI\_Main\_Window\_Width,  
tsp\_EPTF\_GUI\_Main\_Window\_Height,  
tsp\_EPTF\_GUI\_Main\_Window\_Title,  
tsp\_EPTF\_GUI\_Main\_Toolbar\_WidgetId,  
tsp\_EPTF\_GUI\_Main\_Tabbox\_WidgetId**These are the main Window and Tabbox parameters and the default toolbar’s ID, all optional.

**tsp\_xtdp\_listen\_addr:="127.0.0.1"  
tsp\_xtdp\_listen\_port := 11420;**The component will connect to the runtime GUI listening on tsp\_xtdp\_listen\_addr address and tsp\_xtdp\_listen\_port port.

**tsp\_EPTF\_savedGUIPrefix := "GUI"**The f\_EPTF\_UIHandler\_saveLayout function saves the layout to a configuration file with the given prefix. The name of the file consists of the value of this module parameter, plus the date, month, year, hours and minute.

**tsp\_EPTF\_GUI\_DefaultWindow := ""**See [3.3.1.2]

**tsp\_EPTF\_UIHandler\_headlessmode := true**Switch it to true if you want the UIHandler to be able to operate in headless mode.

**tsp\_EPTF\_UIHandler\_udp\_port := 3333;**When UIHandler is in client mode, and headless mode is on, it tries to reconnect to the server GUI. In order to do this, it first pings the GUI at the specified port. Only after receiving proper answers to these pings it attempts to reopen the lost XTDP connection.

**tsp\_EPTF\_UIHandler\_servermode := false;**Starts the UIHandler in XTDP-server mode if true, so it waits for a GUI to connect.

**tsp\_EPTF\_UIHandler\_Authmsg := "Enter password";  
tsp\_EPTF\_UIHandler\_Authresp := "";**When these parameters are specified as non-empty strings, the GUI needs to authenticate itself.  
UIHandler will send the first message to GUI, what waits for an input from the user. In case the “password” from the user is the same as the second module parameter’s value, then the connection is established. Otherwise the UIHandler closes the communication connection.

**tsp\_EPTF\_UIHandler\_authMaxTimer := 1.0;**This module parameter defines the maximal time of the authentication. If the password is not received during this time, the XTDP session is closed by the UIHandler.

**tsp\_EPTF\_UIHandler\_handshakeMaxTimer := 1.0  
tsp\_EPTF\_UIHandler\_byeMaxTimer := 1.0**Timers used for the XTDP protocol.

**tsp\_EPTF\_UIHandler\_maxWaitTimer := 5.0**Max wait time for responses.

**tsp\_EPTF\_UIHandler\_maxGUIWaitTime := 30.0**The max wait time in the init function of the UIhandler to wait until Runtime GUI is connected.

**tsp\_EPTF\_UIHandler\_bufferedXTDP := 0.5;**By default, UIHandler sends XTDP Request messages in a buffered way, with 0.5 second timeouts. This timeout can be configured with this module parameter. The value 0.0 disables this feature.

**ts\_CLI\_refreshRate := 5.0;**The refreshing time of the CLI.

**tsp\_EPTF\_UIHandler\_CLIPrefix := ""**Common prefix for the UIHandler Command-Line Interface commands

**tsp\_EPTF\_UIHandlerClient\_maxKillTime** **:= 30.0;**  
This tsp defines in seconds how long shall wait the UIHandler\_Client component for every pending message to arrive during the cleanup. After this timeout the component exists even if not all pending messages arrived.

Note, that the XTDP test port must be configured to use connection ASPs in the test-port parameters section in order to be able to use the UIHandler functionality.

**tsp\_debug\_EPTF\_UIHandlerClient\_Functions := false**   
**tsp\_debug\_EPTF\_UIHandler\_VariableUI := false  
tsp\_debug\_EPTF\_UIHandler\_CLI\_Functions := false  
tsp\_debug\_EPTF\_UIHandler\_Functions := false**Enables debug printouts in UIHandlerClient,VariableUI,CLI,UIHandler

**tsp\_schemaLocation := "XTDP-Message.xsd"**  
Location of the schema

**tsp\_EPTF\_runtimeGuiExitButtonWidgetId := "EPTF\_exit\_ttcn\_button"**  
The widget ID of the exit button.

**tsp\_EPTF\_UIHandler\_enableBrowserGUI := true**

Enables the BrowserGUI feature.

**tsp\_EPTF\_UIHandler\_Browser\_RemoteAddress := "127.0.0.1"**

The BrowserGUI feature will listen on this adress.

**tsp\_EPTF\_UIHandler\_Browser\_RemotePort := 4000**

The BrowserGUI feature will listen on this port.

**tsp\_EPTF\_UIHandler\_Browser\_xsl2send := "EPTF\_LoadMain.xsl"**

This xsl file will be used for the main page.

**tsp\_EPTF\_UIHandler\_Browser\_xsl4Tabpage := "EPTF\_LoadTabpage.xsl"**

This xsl file will be used for the pages when changing between different tabpages.

**tsp\_EPTF\_UIHandler\_Browser\_favicon2send := "favicon.ico"**

This file will be used as favicon icon.

**tsp\_EPTF\_UIHandler\_Browser\_welcomeScreen := "welcome.html"**

This file will be used as welcome screen, displayed when nothing is on screen.

**tsp\_EPTF\_UIHandler\_Browser\_directory := "../BrowserGUI/"**

Location of the BrowserGUI files : .png, .js, .jpg, .xsl

## Screen layout design

### Static design

#### XUL file

The runtime GUI can be initialized with a static XUL file. The initialization XUL files can be set by a command line parameter when the runtime GUI starts. See [9].

The XUL files can be edited by text editors.

#### Module parameter

The tsp\_EPTF\_GUI\_DefaultWindow module parameter can describe an initial GUI layout. If this module parameter was set, the UIHandler component clears the actual runtime GUI layout and creates a new one within the f\_EPTF\_UIHandler\_init\_CT function described by the module parameter.

The structure of this module parameter is the same as the structure of a XUL file except that the XML header is missing, and the quotation marks (“), the escape characters (\) and the new line characters must be preceded by an escape character.

The f\_EPTF\_UIHandler\_snapshot function returns the content of the current runtime GUI, and the f\_EPTF\_UIHandler\_saveLayout(f\_EPTF\_UIHandler\_snapshot()) creates a configuration file containing the runtime GUI layout.

#### Initialization parameter

If the pl\_windowLayout parameter of the f\_EPTF\_UIHandler\_init\_CT function is present, then it can describe an initial runtime GUI layout. The syntax is the same as the syntax of the tsp\_EPTF\_GUI\_DefaultWindow module parameter.

### Dynamic design

The UIHandler component provides functions to add widgets to the runtime GUI, remove widgets, or clear the GUI. UIHandlerClient component provides functions to add widgets to the runtime GUI, remove widgets too.

## Runtime test configuration design (deployment)

Any component in the runtime configuration can extend the EPTF\_UIHandler\_CT, the EPTF\_UIHandlerClient\_CT or the EPTF\_UIHandler\_CLI\_CT components. Even a single component can extend all the three component types at the same time.

# DataSource elements

The UIHandler can be used as DataSource Client. It has a data element, which can be connected to a button widget on the GUI. The application used data elements can be saved to a file via pressing this button.

There is another data source that can be used to display progress information during startup. The value of this dataSource should be changed to write new progress information into this variable. All changes of this dataSource are appended to each other into the progressBarHistory dataSource. The value of the progressBar dataElement can be changed from the UIHandler component by the f\_EPTF\_UIHandler\_updateProgress API function call. From other PTCs, the dataSource should be requested from the DataSourceClient feature and the returned variable can be adjusted to change the progress information.

To display progress information during custom GUI creation, a simple XML containing the UIHandler progressBar or progressBarHistory dataSources should be loaded before the large custom XML. Also the progressBar dataSource can be added to the custom XML also, which makes it possible to see the progress information after the custom XML is loaded.

For more information on dataSources in UIhandler see the natural documentation and the next chapter.

# Simulation mode

There is a extended functionality in UIHandler, it can be started in simulation mode with a silmulation database file. The main purpose of the simulation mode is to be able to see the custom gui layout without knowing the data behind it. If the data doesn’t available at all (not as dataelement, neither in presaved database), random data will show in the GUI to simulate its layout.

There are several behaviours, depending on the initial parameters:

public function f\_EPTF\_UIHandler\_init\_CT(

in charstring pl\_selfName,

in boolean pl\_connectGUI := true,

in template charstring pl\_windowLayout := "",

in EPTF\_CLI\_CT pl\_CLI\_compRef := null,

in charstring pl\_prefix := tsp\_EPTF\_UIHandler\_CLIPrefix,

in boolean pl\_simulationMode := false,

in charstring pl\_simulation\_buildDB\_file := "")

For this functionality the last 2 parameters are important.

1. If the pl\_simulationMode is false, and the pl\_simulation\_buildDB\_file is empty string, the UIHandler works the usual way.
2. If the pl\_simulationMode is false and the pl\_simulation\_buildDB\_file is given, that means the user can use a button to save the used dataelements to the given file. This file is can be used as database file in simulation mode. Only use this mode, if you want to save the database to file, otherwise building the database has performance cost.
3. If the pl\_simulationMode is true and the pl\_simulation\_buildDB\_file is given, that means that the UIhandler will show the GUI layout with all dataelement. If the dataelement can not be handled by DataSource server, the UIhandler will use the given database. If the database does not contain it either, the UIHandler will use random data.

# Examples

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

* cfg.cfg
* UI\_Handler\_Test.ttcn

## Configuration file

The used configuration file (cfg.cfg) is for the UIHandler example is placed in the demo directory.

## Demo Module

In this section we will look at the tc\_Messages testcase in the demo directory.

The task is simply making two textboxes, that one is writeable, and the seconds update itself automatically to the first’s data. This is done by two variable and two subscribing method.

After installing and setting up the dependencies, we need to design the layout as described in the previous section. It is done by a simple TTCN code, what sends the widgets (XTDP templates) directly to GUI.

var EPTF\_UIHandlerClient\_SubsRet ret := f\_EPTF\_UIHandlerClient\_addElementToGUI(c\_MainHBox, tsp\_EPTF\_GUI\_Main\_Window\_WidgetId);

ret := f\_EPTF\_UIHandlerClient\_addElementToGUI(c\_MainTabBox, tsp\_EPTF\_GUI\_Main\_Window\_WidgetId);

ret := f\_EPTF\_UIHandlerClient\_addElementToGUI(c\_MainWidgets, "bubuTab");

f\_EPTF\_UIHandlerClient\_removeElementFromGUI("EPTF\_snapshot\_button");

f\_EPTF\_UIHandlerClient\_enableGUIElement(c\_EPTF\_runtimeGuiExitButtonWidgetId);

var XTDP\_XML\_Tag vl\_xul;

vl\_xul :={

xtdp\_textbox := {

disabled := false,

flex := omit,

id := "bubuTextBox",

multiline := false,

readonly := false,

rows := 1.0,

textvalue := "bubu",

widgetType := "console",

wrap := false

}

};

ret := f\_EPTF\_UIHandlerClient\_addElementToGUI(vl\_xul, "EPTF\_Main\_hbox\_Textboxes");

vl\_xul.xtdp\_textbox.id := "tb2";

vl\_xul.xtdp\_textbox.textvalue := "tb2";

ret := f\_EPTF\_UIHandlerClient\_addElementToGUI(vl\_xul, "EPTF\_Main\_hbox\_Textboxes");

We will be using the UIHandlerClient component as the type of our MTC and create a PTC with type UIHandler. This component will be the subscriber to the variables. Creating them and make a subscription is done by the following code:

f\_EPTF\_Var\_newInt("intVarProv",10, vl\_intVarIdx);

f\_EPTF\_Var\_newFloat("floatVar",9.7, vl\_floatVarIdx);

f\_EPTF\_UIHandlerClient\_subscribeMe("intVarProv", "intVar\_Subscriber","bubuTextBox");

f\_EPTF\_UIHandlerClient\_subscribeMe("intVarProv","","tb2");

we may deploy two load measuring components:

var EPTF\_HostAdminUI\_CT ct\_hostadmin := EPTF\_HostAdminUI\_CT.create;

ct\_hostadmin.start(f\_EPTF\_HostAdminUI\_behavior("LocalHostAdmin","Localhost",ct\_subscriber));

var EPTF\_HostAdminUI\_CT ct\_hostadmin2 := EPTF\_HostAdminUI\_CT.create(-,”192.168.0.2”);

ct\_hostadmin2.start(f\_EPTF\_HostAdminUI\_behavior("HostAdmin2","Host2",ct\_subscriber));

After that, we only need to implement a main loop and the behaviours that describe the actions to our events. Only one exists in this case, the pressing of the Exit button.

Note, that the widgets that can be manipulated, or can control execution like buttons, should be connected to a variable, and only the variable’s changes should be handled, not the widget’s. In case it is not connected, but used as a controlling action, the shadowed GUIs won’t be updated, and actions from them may affect the execution.

The whole source code can be found in the tc\_Messages testcase of the demo ttcn file.

# References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. [Part 1: Core Language](http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.02.01_60/es_20187301v030201p.pdf)
2. 1/198 17-CRL 113 200/5 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 155 17-CNL 113 512 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
4. 198 17-CNL 113 512 Uen  
   EPTF Core Library for TTCN-3 toolset with TITAN, User Guide
5. 2/155 16-CNL 113 512  
   EPTF CLL UIHandler, Function Description
6. EPTF CLL 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. 198 17-CNL 113 437 Uen.  
   Runtime GUI for TTCN-3 Toolset with TITAN, User's Guide