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

EPTF CLL UIHandler, Function Description

Contents

[1 Introduction 2](#_Toc391063703)

[1.1 Revision history 2](#_Toc391063704)

[1.2 How to Read this Document 3](#_Toc391063705)

[1.3 References 3](#_Toc391063706)

[1.4 Scope 3](#_Toc391063707)

[1.5 Recommended way of reading 4](#_Toc391063708)

[1.6 Typographical conventions 4](#_Toc391063709)

[1.7 Abbreviations 4](#_Toc391063710)

[1.8 Terminology 4](#_Toc391063711)

[2 General Description 5](#_Toc391063712)

[2.1 Overview 6](#_Toc391063713)

[2.2 Sharing EPTF Variables 9](#_Toc391063714)

[2.2.1 Sharing variables with VariableUI 10](#_Toc391063715)

[2.3 UI handling 11](#_Toc391063716)

[2.3.1 Runtime GUI 11](#_Toc391063717)

[2.3.2 CLI 13](#_Toc391063718)

[2.3.3 Web Browser GUI 13](#_Toc391063719)

[3 General functions 15](#_Toc391063720)

[3.1 Naming Conventions 15](#_Toc391063721)

[3.2 Using default UIHandler component 16](#_Toc391063722)

[3.3 Initialization 16](#_Toc391063723)

[4 Variable handling 17](#_Toc391063724)

[4.1 Order the UIHandler component to subscribe to an EPTF Variable 17](#_Toc391063725)

[4.2 Order the UIHandler component to subscribe to an EPTF Variable and connect it to a widget 18](#_Toc391063726)

[4.3 Using the VariableUI 19](#_Toc391063727)

[5 UI handling 20](#_Toc391063728)

[5.1 User interactions 20](#_Toc391063729)

[5.2 GUI handling 21](#_Toc391063730)

[5.2.1 Layout manipulation 21](#_Toc391063731)

[5.2.2 Progress information during startup 26](#_Toc391063732)

[5.2.3 Convenience functions in UIHandlerClient 27](#_Toc391063733)

[5.3 HostAdminUI Functions 28](#_Toc391063734)

[5.4 LoadRegulator UI Functions 28](#_Toc391063735)

[5.5 StatCaptureUI Functions 28](#_Toc391063736)

[5.6 XTDP operation modes of UIHandler 28](#_Toc391063737)

[5.6.1 XTDP-client mode 29](#_Toc391063738)

[5.6.2 XTDP-server mode 29](#_Toc391063739)

[5.7 Shadow GUI 30](#_Toc391063740)

[5.8 Headless mode 30](#_Toc391063741)

[5.9 CLI handling 30](#_Toc391063742)

[5.9.1 Data manipulation 30](#_Toc391063743)

[5.9.2 Monitor the values of variables 31](#_Toc391063744)

[5.9.3 Other commands 32](#_Toc391063745)

[6 Feature GUIs 33](#_Toc391063746)

[6.1 HostAdminUI 33](#_Toc391063747)

[7 Summary table of all public functions for UIHandler 33](#_Toc391063748)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2007-11-07 | PA1 | First draft version | ELSZSKU |
| 2007-12-04 | PA2 | Update after review, 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 feature | EZSOSZA |
| 2008-02-14 | PA6 | More CLL feature UIs | EZSOSZA |
| 2008-02-22 | PA7 | Shadow GUI support | EZSOSZA |
| 2009-02-25 | PB1 | Set focus feature, client convenience functions | EISTFAL |
| 2010-02-01 | PC1 | CLI: added LS commands, STOP stops the whole test execution. | EGBOTAT |
| 2010-04-23 | PD1 | Use-case diagram changed | ELSZSKU |
| 2010-07-16 | PE1 | Refresh rate | ETHJGI |
| 2011-01-18 | PF1 | XTDP Protocol Module changes | ETHJGI |
| 2011-02-09 | PF2 | XTDP Authentication update + DataSource | ETHJGI |
| 2011-05-20 | PG1 | Condition added to DataSource | ETHJGI |
| 2011-07-22 | PH1 | Built-in data elements added | EBALLUG,ETHJGI |
| 2011-08-22 | PH2 | DataSource removed | EBALLUG |
| 2012-04-13 | PJ1 | Externalvalue update | EKOVIST |
| 2012-05-25 | PJ2 | Simulation mode update | EMIHMIK |
| 2012-07-26 | PK1 | ProgressBar dataSource | ETHJGI |
| 2012-09-13 | PL1 | Web Browser GUI updated, htmlcode added | EMIHMIK |
| 2013-01-28 | PM1 | Widget ID restrictions | ETHJGI |
| 2013-01-29 | PM1 | Web Browser GUI configuration and Custom CSS handling | EKOVIST |
| 2013-06-11 | PM2 | Remove XTDP Test Port functions, added iterator change functionality | EJNOSVN |
| 2013-06-19 | M | Updated for release | EJNOSVN |
| 2013-12-17 | N | Updated for release | ENIKOTA |
| 2014-06-17 | S | Updated for release | ESZILSZ |

## How to Read this Document

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

## 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 113 200/4 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 109 21-CNL 113 512-19 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Product Revision Information
4. 155 17-CNL 113 512 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
5. EPTF Core Library for TTCN-3 toolset with TITAN, Reference Guide  
   http://ttcn.ericsson.se/products/libraries.shtml
6. 18/155 16-CNL 113 512  
   EPTF Core Library Variable, Function Description
7. 198 17-CNL 113 437 Uen.  
   Runtime GUI for TTCN-3 Toolset with TITAN, User's Guide
8. 155 17-CNL 113 320 Uen  
   Telnet Test Port for TTCN-3 Toolset with TITAN, Function Specification
9. 198 17-CNL 113 663 Uen  
   XTDP Protocol Module User Guide

## Scope

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

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of EPTF Core Library [4]. They should get familiar with the list of acronyms and the glossary in Section 1.7 and 1.8, respectively.

## Typographical conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context. Moreover, whenever a concept is mentioned that has a special meaning as described in the Glossary (Section 1.8) of this document, then these occurrences are marked with an initial arrow, e.g., *🡪 TitanSim Statistics*.

## Abbreviations

CLI Command Line Interface

CLL Core Load Library

EPTF Ericsson Load Test Framework, formerly TITAN Load Test Framework

GUI Graphical User Interface

PTC Parallel Test Component

TitanSim Ericsson Load Test Framework, formerly TITAN Load Test Framework

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

UDP User Datagram Protocol

UI User Interface

XTDP eXtensible Titan Display Protocol

XTDL eXtensible Titan Display Language, a derivative of XUL

XUL eXtensible User-interface Language

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

*UIHandler* is a main feature or an abstract layer of the TitanSim Core Library that is supposed to communicate with a User Interface e.g. Runtime GUI, or CLI.

*UIHandler component* means a component that extends the EPTF\_UIHandler\_CT component.

*UIHandlerClient or UIHandlerClient component* is a component that extends the EPTF\_UIHandlerClient\_CT.

*UIHandler CLI* means a component that extends the EPTF\_UIHandler\_CLI\_CT component.

*Headless mode* is a feature of the UIHandler that is useful, when the runtime GUI goes away. When running TitanSim applications with headless mode turned on, the UIHandler will be able to reconnect to the GUI.

*EPTF Variable* is a variable of EPTF\_Var\_CT component.

*Widget* is a screen element of the runtime GUI.

*Command TELNET terminal* is the terminal that connects to the v\_UIHandler\_CLI\_TELNETaspIf port of the EPTF\_UIHandler\_CLI\_CT component.

*Display TELNET terminal* is the terminal that connects to the v\_UIHandler\_CLI\_displayTELNETaspIf port of the EPTF\_UIHandler\_CLI\_CT component.

*VariableUI* is a feature of the UIHandler, that allows easier widget – variable connections and subscriptions.

# General Description

This document specifies the UIHandler feature of the EPTF Core Library.The main goal of the UIHandler feature is to hide the details of handling the Graphical User Interface (or shortly GUI), including both the communication with, and the layout-configuration of the GUI. Furthermore, it provides TELNET access [8] to TitanSim, which can be used as a Command Line Interface (CLI) substitute of the GUI.

## Overview

Users can display EPTF Variables [6], or can change the value of these variables using either the TitanSim Runtime GUI [7], a standard HTML web browser or a TELNET-client. These use-cases are collectively called as “data-manipulation” in this document and are shown in Figure 1.

Figure 1 Data manipulation use cases

Users can change layout of the GUI via adding, removing, enabling or disabling GUI elements. These use-cases are collectively called as “widget manipulation” in this document and are shown as in Figure 2.

Figure 2 GUI manipulation use cases

There are five logical roles realized as dedicated component types in the UIHandler CLL feature:

* The components extending EPTF\_UIHandlerClient\_CT manage EPTF Variables, describe how to handle them (in which widget they have to be displayed), and how to interpret their values. Their logical role is the “client(s)” of the UIHandler, they request the UIHandler to display their EPTF Variable data that they owns. For this relationship the UIHandler clients act as EPTF Variable “providers” [6].
* The component extending EPTF\_UIHandler\_CT connect EPTF Variables of its client components (i.e., those extending EPTF\_UIHandlerClient\_CT) to the GUI. The logical role of the UIHandler is a “server” the deals with the presentation of the EPTF Variable data of its clients. For this relationship the UIHandler acts as an EPTF Variable “subscriber” [6].
* The component extending EPTF\_UIHandler\_CLI\_CT is a special UIHandler that provides TELNET terminal access to remotely control those EPTF Variables that are subscribed by the UIHandler.
* The component extending EPTF\_UIHandler\_VariableUI\_CT is basically a EPTF\_UIHandlerClient\_CT(it also extends that) but has some functions, that can manage massive widget – variable subscriptions and putting them to the GUI.
* It contains UIs to selected CLL features. You may able to use the feature without or with UI support. When the former is used, the feature will add his own widgets, variables to the GUI.

At any given time only one UIHandler component can connect to the same GUI. There can be more than one UIHandlerClient components sharing EPTF Variables with the same UIHandler component. Figure 3 describes a typical TTCN-3 run-time configuration using UIHandler. Run-time components (PTC objects) are visualized as rectangles. If the component-type of such a PTC defined via (possibly recursive) extension of another component type(s), then it is depicted as embedded boxes according to the extension hierarchy.

runtime GUI

XTDL

TELNET terminal

TELNET

EPTF\_UIHandlerClient\_CT

EPTF\_Var\_CT

EPTF\_Var\_Mgmt\_PT

EPTF\_adminPort\_PT

PTC#m

PTC#n

EPTF\_Var\_CT

IPL4 transport

EPTF\_Var\_Mgmt\_PT

EPTF\_CLI\_TELNET\_PCO

EPTF\_adminPort\_PT

EPTF\_UIHandler\_CT

EPTF\_UIHandlerClient\_CT

EPTF\_Var\_CT

EPTF\_Var\_Mgmt\_PT

EPTF\_adminPort\_PT

PTC#p

EPTF\_UIHandler\_CLI\_CT

Web Browser

XTDL

IPL4 transport

Web Browser

XTDL

Figure 3 A common configuration of the components of the UIHandler feature

If the users want to display their data on different UIs, there can be more UIHandler components connected to different UIs.

The same variable can be shared with more than one UIHandler components.

Figure 4 describes a more complicated configuration, where there are two UIHandlers (with their associated GUI).

runtime GUI

TELNET terminal

EPTF\_UIHandlerClient\_CT

EPTF\_Var\_CT

EPTF\_Var\_Mgmt\_PT

EPTF\_adminPort\_PT

PTC#q

EPTF\_UIHandlerClient\_CT

EPTF\_Var\_CT

EPTF\_Var\_Mgmt\_PT

PTC#r

PTC#n

EPTF\_Var\_CT

EPTF\_UIHandler\_CT

EPTF\_UIHandler\_CLI\_CT

EPTF\_UIHandlerClient\_CT

EPTF\_Var\_CT

EPTF\_Var\_Mgmt\_PT

EPTF\_adminPort\_PT

PTC#p

PTC#m

EPTF\_Var\_CT

EPTF\_UIHandler\_CT

EPTF\_UIHandler\_CLI\_CT

runtime GUI

TELNET terminal

Web Browser

Web Browser

Web Browser

Web Browser

Figure 4 A more complicated configuration of the components of the UIHandler feature

In any actual run-time test configuration any component can extend the EPTF\_UIHandler\_CT, EPTF\_UIHandlerClient\_CT or the EPTF\_UIHandler\_CLI\_CT component, even all of them at the same time. It is quite possible to deploy all these logical roles on the same PTC.

## Sharing EPTF Variables

To share EPTF Variables UIHandler uses the subscription functionalities of the EPTF\_Var\_CT component.

The component (PTC#2) that extends the EPTF\_UIHandlerClient\_CT component and wants to allow its EPTF Variable to be changed/displayed via the GUI/CLI, must send a subscription note to the component that extends the EPTF\_UIHandler\_CT component (PTC#1). This subscription note describes

* the name of the EPTF Variable that the component wants to use with the GUI/CLI,
* the name of the EPTF Variable that the UIHandler will create as a subscriber of the above variable,
* the ID of the widget in which the subscriber variable will be displayed
* the subscription mode and.
* the refresh rate which specifies the rate of refresh for this subscription. The refresh period is calculated as the refreshRate multiplied by the minSyncInterval.

PTC#1 subscribes to the specified variable via the functionalities of the EPTF\_Var\_CT component. After a successful subscription it sends back an acknowledgment and if there was a widget ID in the message, refreshes the content of the given widget.

PTC#2

EPTF\_UIHandlerClient\_CT

PTC#1

EPTF\_UIHandler\_CT

subscribeMeNote

subscribe

subscribeResp

subscribeMeNoteAck

Figure 5 Sequence of the variable sharing.

### Sharing variables with VariableUI

When extending EPTF\_UIHandler\_VariableUI\_CT, it is a bit easier to manage variable subscriptions. Only widgetId(or widget) – variable name pairs, called a connection have to be provided, and they should automatically be added to the GUI, and connected with the variables.

There are three forms, how the feature can be used:

* In the simplest form, it is assumed, that all the widgets are on the GUI, so only their widgetIDs should be provided with the variable that connects to.
* If the widgets that would subscribe to a variable are not present or it is not know that is present on the GUI, the whole widget should be provided with its parent’s ID with the [appropriate](http://dict.sztaki.hu/dict_search.php?M=3&O=ENG&E=1&C=1&A=1&S=H&T=1&D=0&G=0&P=0&F=0&MR=100&orig_lang=ENG%3AHUN%3AEngHunDict&orig_mode=3&orig_word=megfelelo&flash=&sid=6253a1a162c7ba6278724dd8455f7b7c&L=ENG%3AHUN%3AEngHunDict&W=appropriate) function.
* In some cases, the first two possibilities not enough, also an other widget should be provided upon, called the holding widget. Further more, one holding widget could be used for many widgets – variable pairs, so there is a reference with the widget to a holding widget too, that could be provided with the appropriate function.

After providing all the pairs, it is possible to add the widgets and subscribe all of them in a single step.

## UI handling

### Runtime GUI

The UIHandler component communicates with the Runtime GUI through XTDP messages.

After the UIhandler and Runtime GUI is connected via TCP, the XTDP session will be initiated. It is established after a successful XTDP handshake and authentication. The authentication is optional and can be switched off by setting the tsp\_EPTF\_UIHandler\_Authmsg or the tsp\_EPTF\_UIHandler\_Authresp module parameter to empty string. The password that has to be entered on the Runtime GUI can be set in the tsp\_EPTF\_UIHandler\_Authresp parameter. The maximal time of authentication can be adjusted by the module parameter tsp\_EPTF\_UIHandler\_authMaxTimer. By default the UIHandler authentication is switched off.

#### GUI layout manipulation

Users can *add*, *remove*, *enable* or *disable* components of the runtime GUI. Both the UIHandler and the UIHandlerClient components provide means for GUI layout manipulation.

To initiate GUI layout changes from the UIHandlerClient component, the UIHandlerClient component sends messages to the UIHandler component via its EPTF\_adminPort\_PT port. The UIHandler component books the changes and sends the appropriate XTDP messages to the runtime GUI via TCP (using the CLL IPL4 Transport).

#### Data manipulation using the runtime GUI

Users can connect EPTF Variables to widgets by their names.

When the content of such a provider variable in one of the UIHandlerClients has been changed, then the EPTF Variable feature automatically updates the content of the related subscriber variable in the UIHandler component. Upon the detection of this change the UIHandler component sends an XTDP message to the runtime GUI and changes the content of the associated widget, as shown in Figure 6.

PTC#m

EPTF\_UIHandlerClient\_CT

PTC#n

EPTF\_UIHandler\_CT

adjustContent

adjustContentResp

runtime GUI

sendTimeline

singleRqPutValueTo…

Figure 6 Displaying the content of a variable after its change

When a user changes the content of a widget – e.g. types into a tree cell, or presses a button – the runtime GUI sends a message to the UIHandler component. If there is an associated variable, the UIHandler component initiates a change of its content via the “adjust” method of the EPTF\_Variable [6] feature. Figure 7 describes this synchronization mechanism.

PTC#m

EPTF\_UIHandlerClient\_CT

PTC#n

EPTF\_UIHandler\_CT

adjustContent

adjustContentResp

runtime GUI

widgetModification

parseGuiParam

Figure 7 The synchronization mechanism

### CLI

After an UIHandler component creates EPTF Variables by a subscription, the created EPTF Variables can be accessed via a TELNET terminal application. Users can connect their TELNET terminal to the UIHandler component and change or display the values of the variables. The UIHandler component interprets the commands received on TELNET port and sends back the answers on the same port.

Figure 8 describes the collaboration of the components using the CLI functionality.

UIHandler

TELNET terminal

UIHandlerClient

setContent

adjustContent

adjustContentResp

Figure 8 Collaboration when the user sets the content of a variable

Users can monitor the content of the variables handled by UIHandler by ordering a periodic printout of their content. Such periodic printout will be directed to another TELNET terminal. The former is called the “command” TELNET terminal, the later is called the “display” TELNET terminal, respectively. The UIHandler component interprets the commands received on the command TELNET terminal and periodically sends the content of the required variables to the display TELNET terminal.

varContentAnswer

UIHandler

TELNET terminal

TELNET terminal for display

displayContent

Figure 9 Collaboration when the user monitors the content of a variable

### Web Browser GUI

The UIHandler can behave like a web server, if the module parameter tsp\_EPTF\_UIHandler\_enableBrowserGUI is set to true (by default it is). The UIHandler is waiting for requests on a hostname and port given by the module parameters:

tsp\_EPTF\_UIHandler\_Browser\_RemoteAddress (default: "127.0.0.1") and tsp\_EPTF\_UIHandler\_Browser\_RemotePort (default: 4000).

When a web browser (currently Mozilla/Chrome/Internet Explorer 7+ are supported) connects to the UIHandler, the content of the GUI is sent to the browser via an XML message, which looks similar to an XTDP message. The difference is mainly an xml-stylesheet tag. The module parameter tsp\_EPTF\_UIHandler\_Browser\_xsl2send (default: "EPTF\_LoadMain.xsl”) is used to convert the given XML message to a HTML compatible format.

When the browser gets the xml, it downloads the xsl, it makes the transformation into html. The browser downloads the images and the javascript files. The favicon.ico is determined by the module parameter : tsp\_EPTF\_UIHandler\_Browser\_favicon2send ( default: "favicon.ico")

There is an opportunity to show a welcome screen, while the application is not ready. To show/hide this screen, the following functions can be used: f\_EPTF\_UIHandler\_Browser\_enableWelcomeScreen(); f\_EPTF\_UIHandler\_Browser\_disableWelcomeScreen(). The welcome screen is defined in the tsp\_EPTF\_UIHandler\_Browser\_welcomeScreen module parameter.

All of the files requested by the browser has to be placed in the directory determined by the module parameter tsp\_EPTF\_UIHandler\_Browser\_directory ( default: "../BrowserGUI/"). If the directory does not exist, the UIhandler tries to find the files in the current directory. The default files ( .js, .png, .jpg, .xsl, .css) are located in the src/UIHandler/BrowserGUI directory, this directory should be linked to the correct place defined by the parameter above.

When the page is loaded, the values of the widgets are refreshed using JavaScript sending and receiving JSON strings ( widgetId:value pairs).

The Browser GUI can be configured with the following functions:

* f\_EPTF\_UIHandler\_Browser\_getRemoteHosts - This function returns the Host names and ports on which the BrowserGUI is activated.
* f\_EPTF\_UIHandler\_Browser\_addRemoteHosts - This function adds the given Host name and port pairs to the list on which the BrowserGUI is activated and activates them as well.
* f\_EPTF\_UIHandler\_Browser\_setRemoteHosts - This function sets and activates the given Host name and port pairs on which the BrowserGUI is listening.
* f\_EPTF\_UIHandler\_Browser\_clearRemoteHosts - This function clears the host name and ports on which the BrowserGUI is listening.
* f\_EPTF\_UIHandler\_Browser\_removeRemoteHosts - This function removes and deactivates the given Host name and port pairs on which the BrowserGUI is listening.
* f\_EPTF\_UIHandler\_Browser\_setBrowserDirectory - This function sets the BrowserGUI directory, where the .png, .css, .js and several other files are, needed by the BrowserGUI feature.
* f\_EPTF\_UIHandler\_Browser\_getBrowserDirectory - This function returns with the BrowserGUI directory string, where the .png, .css, .js and several other files should be, needed by the BrowserGUI feature.

Custom CSS styles can be setup using the following functions:

* f\_EPTF\_UIHandler\_Browser\_getCustomStyle - This function returns the active custom CSS style of the BrowserGUI.
* f\_EPTF\_UIHandler\_Browser\_addCustomStyle - This function adds the given CSS Style to the main.css from file.
* f\_EPTF\_UIHandler\_Browser\_setCustomStyle - This function sets the custom style (CSS) from file.
* f\_EPTF\_UIHandler\_Browser\_clearCustomStyle - This function clears the custom style CSS elements.
* f\_EPTF\_UIHandler\_Browser\_removeCustomStyle - This function removes the custom style (CSS) based on filename.
* f\_EPTF\_UIHandler\_Browser\_addCustomStyleStr - This function adds the given CSS Style to the main.css from string.
* f\_EPTF\_UIHandler\_Browser\_setCustomStyleStr - This function sets the custom style (CSS) from string.
* f\_EPTF\_UIHandler\_Browser\_removeCustomStyleStr - This function removes the custom style string (CSS) based in string.

# General functions

## Naming Conventions

All functions of the EPTF\_UIHandler\_CT component have the prefix f\_EPTF\_UIHandler\_.

All functions of the EPTF\_UIHandlerClient\_CT component have the prefix f\_EPTF\_UIHandlerClient\_.

Functions of the EPTF\_UIHandler\_CLI\_CT component have the prefix f\_EPTF\_UIHandler\_CLI\_.

## Using default UIHandler component

Although the UIHandlerClient components can use many UIHandler components in the executable, usually there is only one GUI; and consequently only one UIHandler. Therefore, the users can declare a default UIHandler component for a given UIHandlerClient and implicitly use it with all the UIHandler management functions.

To support this, all of the UIHandlerClient component API functions have two forms: one with an explicit UIHandler component reference parameter and one without it. Functions having an explicit UIHandler component parameter manage the specified UIHandler component, the functions without this parameter manage the default UIHandler component.

Users can specify the default UIHandler component in the f\_EPTF\_UIHandlerClient\_init\_CT initializer function.

## Initialization

Before using the functions of the UIHandler component, the

function f\_EPTF\_UIHandler\_init\_CT(  
 in charstring pl\_selfName,   
 in boolean pl\_connectGUI := true,   
 in 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 := ")

function should be called. The pl\_selfName parameter specifies the name of the component. The pl\_connecGUI parameter can be used to force the UIHandler to connect to the Runtime GUI during its initialization in XTDP-client mode. This connection means not only the establishment of the TCP connection, but the successful XTDP handshake and authentication as well. The maximal wait time for connection establishment can be controlled by the module parameter tsp\_EPTF\_UIHandler\_maxGUIWaitTime. If the TCP connection fails, but headless mode is enabled, this function will not wait for the XTDP session to be established and will exit before the maximal waiting time entering into headless mode.

A predefined initial layout can be specified in the pl\_windowLayout parameter. If this argument is given, the layout on the Runtime GUI will be replaced by the layout specified. The layout has to be given in a charstring value that can be decoded by the decode function f\_EPTF\_UIHandler\_XSD\_decodeXUL which is a wrapper for the decode function of the XTDP Protocol Module [9].

Before using the functions of UIHandlerClient component, the

function f\_EPTF\_UIHandlerClient\_init\_CT(  
 in charstring pl\_selfName,  
 in EPTF\_UIHandler\_CT pl\_DefaultUIHandler)

function should be called.

Before using the functions of UIHandler\_CLI component, the

function f\_EPTF\_UIHandler\_CLI\_init\_CT(  
 in charstring pl\_selfName,  
 in boolean pl\_connectGUI := true)

function should be called.

The pl\_simulationMode parameter can be used to initialize UIHandler in simulation mode. The pl\_simulation\_buildDB\_file parameter definies the file, what will be used to save the database or load the database in simulation mode. More information about using these 2 parameters can be found in the UIHandler User Guide document.

# Variable handling

Since the UIHandler component subscribes to EPTF Variables of the UIHandlerClients, therefore both the UIHandlerClient and the UIHandler components use EPTF Variable API [6] functions to initiate such subscriptions and connect such variables to GUI widgets.

## Order the UIHandler component to subscribe to an EPTF Variable

Using the

function f\_EPTF\_UIHandlerClient\_subscribeVariableTo(  
 in EPTF\_UIHandler\_CT pl\_subscriber,  
 in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeline,

in integer pl\_refreshRate := -1)

function the UIHandlerClient component can order the UIHandler component to subscribe to its EPTF Variable.

The pl\_providedParamName specifies the name of the EPTF Variable. The UIHandler component will subscribe to this variable, and the created EPTF Variable will have the name specified in the pl\_destParamName parameter.

The pl\_refreshRate parameter specifies the rate of refresh for this subscription. The refresh period is calculated as the refreshRate multiplied by the minSyncInterval. If this argument is not specified (or <=0 value is given) the default refresh rate is used with period defined by the syncInterval parameter. The minSyncInterval and syncInterval parameters can be set by the functions:

f\_EPTF\_Var\_setSyncInterval and f\_EPTF\_Var\_setMinSyncInterval

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_subscribeVariable(  
 in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeline,

in integer pl\_refreshRate := -1)

## Order the UIHandler component to subscribe to an EPTF Variable and connect it to a widget

Using the

function f\_EPTF\_UIHandlerClient\_subscribeMeTo (  
 in EPTF\_UIHandler\_CT pl\_subscriber,  
 in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_UIHandler\_WidgetIdString pl\_widgetId := "",  
 in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeline,

in integer pl\_refreshRate := -1)

function the UIHandlerClient component can order the UIHandler component to subscribe to its EPTF Variable and eventually connect it to a widget on the runtime GUI.

The pl\_providedParamName specifies the name of the EPTF Variable. The UIHandler component will subscribe to this variable, and the created EPTF Variable will have the name specified in the pl\_destParamName parameter.

If the pl\_widgetId parameter is not an empty string, the UIHandler component will connect the variable to the specified widget. That means that when the content of the EPTF Variable changes, the UIHandler component changes the content of the associated widget. Similarly, when the content of the specified widget changes, the UIHandler component changes the content of the variable.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_subscribeMe (  
 in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_UIHandler\_WidgetIdString pl\_widgetId := "",  
 in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeline,

in integer pl\_refreshRate := -1)

It is possible to connect a GUI widget to a variable which is defined in an other component. This can be done by using the function

function f\_EPTF\_UIHandlerClient\_subscribeTo (  
 in EPTF\_UIHandler\_CT pl\_subscriber,  
 in EPTF\_Var\_CT pl\_remoteCompRef,

in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_UIHandler\_WidgetIdString pl\_widgetId := "",

in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeline,

in integer pl\_refreshRate := -1)

Here the parameter remoteCompRef is that component that the EPTF variable created. The pair of this function is

function f\_EPTF\_UIHandlerClient\_subscribe (  
 in EPTF\_Var\_CT pl\_remoteCompRef,

in charstring pl\_providedParamName,  
 in charstring pl\_destParamName := "",  
 in EPTF\_UIHandler\_WidgetIdString pl\_widgetId := "",

in EPTF\_Var\_SubscriptionMode pl\_subsMode := timeLine)

## Using the VariableUI

Initializing the component can be done with   
f\_EPTF\_UIHandler\_VariableUI\_init\_CT(in charstring pl\_selfName,

in EPTF\_UIHandler\_CT pl\_UIHandler)

Put means add the widget to the GUI and subscribe it to the variable. The following function put a variable given by a name from the database.

f\_EPTF\_UIHandler\_VariableUI\_putVarbyName(in charstring pl\_var)

Using the functions

f\_EPTF\_UIHandler\_VariableUI\_putconnections()

f\_EPTF\_UIHandler\_VariableUI\_putlastconnections()

it is possible to put all the variable given already, or just the last few, that are not processed since the last put.

With the function

f\_EPTF\_UIHandler\_VariableUI\_addsimpleconnection(in charstring pl\_varname, in charstring pl\_widgetid)

it is possible to fill the database with an existing widgetID and a variable name. It returns an index in the database.

The function

f\_EPTF\_UIHandler\_VariableUI\_addwidgetconnection(in charstring pl\_varname,

in charstring pl\_widgetid,

in XTDP\_XML\_Tag pl\_widget,

in charstring pl\_parentid)

can be used, to put a widget - variable par into the database, with the widget given in XUL format. This widget will be added to pl\_parentid, and variable will be connected to the widget with pl\_widgetid. It returns an index in the database.

A holder widget can be specified with the following function:

f\_EPTF\_UIHandler\_VariableUI\_addholdingwidget(in charstring pl\_name, in XTDP\_XML\_Tag pl\_widget, in charstring pl\_parentid)

It returns an index in the database.

Assigning a holder widget to a widget – variable pair can be done with the following functions:

f\_EPTF\_UIHandler\_VariableUI\_assignholdingwidget(in charstring pl\_varname, in charstring pl\_holdingname)

f\_EPTF\_UIHandler\_VariableUI\_assignholding(in integer pl\_connidx, in integer pl\_holdingidx)

A variable – widget pair can be removed from the database with the following functions:

f\_EPTF\_UIHandler\_VariableUI\_removeconnection(in charstring pl\_varname)

# UI handling

## User interactions

All the user interactions using the runtime GUI is mapped to EPTF Variable management. User input widgets can be connected to variables, and when the user changes them – e.g. types into a tree cell, or presses a button – the runtime GUI sends a message to the UIHandler component. Then the UIHandler component changes the content of the associated variable as described in 2.3.1.2.

Table 1 describes the permitted mapping of widgets to of EPTF Variable types.

Table 1 Widget types and EPTF Variable types that can be assigned to

|  |  |
| --- | --- |
| Widget type | EPTF Variable type |
| Button | intVal |
| Toolbarbutton | intVal |
| TreeCell of checkbox | boolVal |
| TreeCell of floatField | floatVal |
| TreeCell of pushButton | intVal |
| TreeCell of integerField | intVal |
| TreeCell of string | charstringVal |
| TreeCell of statusLEDOfText | charstringVal |
| TreeCell of statusLED | charstringVal |
| Chart | floatVal |

## GUI handling

### Layout manipulation

As described in Section 2.1 the UIHandlerClient components own their EPTF Variables and they might choose to associate them to widgets of the GUI. Therefore, they may have to change the GUI layout: add or remove, enable or disable those widgets, or set the focus to a particular widget. To support this, the UIHandler and UIHandlerClient components have widget adding, removing, enabling, disabling and set focus functions.

In order to prepare an initial GUI layout EPTF\_UIHandler\_CT component type has widget adding and removing functions. The widget enabling and disabling functionalities are related to handling the particular EPTF Variables, therefore such functions are provided by the UIHandlerClient components. The ‘set focus’ functionality is available by the UIHandlerClient components as well.

When adding, removing, manipulating elements on the GUI as described in this section, all widget IDs must conform to the restrictions in the XTDP protocol module XSD [9].

UIHandlerClient provides a group of convenience functions as well. Using them makes it easy to create specific widgets on the GUI, as button, text label, spacer, chart, tabbed box and horizontal box.

#### Add widgets

Using the

function f\_EPTF\_UIHandlerClient\_XSD\_addElementToGUITo(  
 in Widgets pl\_xul,   
 in EPTF\_UIHandler\_WidgetIdString pl\_parentWidgetId,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function users can add a widget or a tree of widgets under the parent widget pl\_parentWidgetId on the GUI connected to the specified UIHandler component.

The pl\_xul parameter describes the hierarchy of widgets to add to the GUI. See [7].

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_XSD\_addElementToGUI(  
 in Widgets pl\_xul,   
 in EPTF\_UIHandler\_WidgetIdString pl\_parentWidgetId)

In the UIHandler component the corresponding function is:

function f\_EPTF\_UIHandler\_XSD\_addElementToGui(  
 in Widgets pl\_xul,   
 in EPTF\_UIHandler\_WidgetIdString pl\_parentWidgetID := "",  
 inout boolean pl\_widgetExists,  
 in boolean pl\_updateDb := true)

The pl\_updateDb parameter can be used to disable the internal database update in UIhandler.

#### Remove widgets

Using the

function f\_EPTF\_UIHandlerClient\_removeElementFromGUITo(  
 in EPTF\_UIHandler\_WidgetIdString pl\_Widget2Remove,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function users can remove a widget and its children from the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_removeElementFromGUI(  
 in EPTF\_UIHandler\_WidgetIdString pl\_Widget2Remove)

In the UIHandler component the similar function is:

function f\_EPTF\_UIHandler\_removeElementFromGui(  
 in EPTF\_UIHandler\_WidgetIdString pl\_widgetId)

#### Enable widgets

Using the

function f\_EPTF\_UIHandlerClient\_enableGUIElementTo(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function users can enable a widget on the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_enableGUIElement(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID)

#### Disable widgets

Using the

function f\_EPTF\_UIHandlerClient\_disableGUIElementTo (  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function users can disable a widget on the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_disableGUIElement(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID)

#### Setting the focused widget

Using the

function f\_EPTF\_UIHandlerClient\_setFocusToGUIElementTo (  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function users can set the focus to a widget on the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_disableGUIElement(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID)

There is another function specialized for Tabboxes. The

function f\_EPTF\_UIHandlerClient\_setFocusToTabTo (  
 in EPTF\_UIHandler\_WidgetIdString pl\_TabboxID,  
 in integer pl\_TabIndex,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)

function allows the user to select a particular tab of tabbox on the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_setFocusTo Tab(  
 in EPTF\_UIHandler\_WidgetIdString pl\_TabboxID,  
 in integer pl\_TabIndex)

#### Checking existence of widgets

function f\_EPTF\_UIHandlerClient\_widgetExistsOn(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID,  
 in EPTF\_UIHandler\_CT pl\_UIHandler)  
returns true, if the widget already exists on the GUI connected to the specified UIHandler component.

The pair of this function without UIHandler component parameter is

function f\_EPTF\_UIHandlerClient\_widgetExists(  
 in EPTF\_UIHandler\_WidgetIdString pl\_WidgetID)

#### Clear the GUI and create a new window

function f\_EPTF\_UIHandler\_clearGUI() removes all the widgets from the GUI, even the window widget.

The f\_EPTF\_UIHandler\_addWindow() function creates a new window widget on an empty GUI. The id, the height, width and the title of the new window widget can be defined with the tsp\_EPTF\_GUI\_Main\_Window\_WidgetId, tsp\_EPTF\_GUI\_Main\_Window\_Height, tsp\_EPTF\_GUI\_Main\_Window\_Width and tsp\_EPTF\_GUI\_Main\_Window\_Title module parameters.

#### Get the description of the current GUI, and save it into a file

The f\_EPTF\_UIHandler\_snapshot() function returns the XUL representation of the current GUI in a charstring. This description can be stored into a file with the f\_EPTF\_UIHandler\_saveLayout function.

#### Create custom GUI from XML

It is possible to load widgets from XML to the GUI. The XML is decoded into the Widgets type defined in the XTDP Protocol Module. The following function puts the widgets in XML under the given parent widget id:

public function f\_EPTF\_UIHandler\_createGUI (

in charstring pl\_xul,

in charstring pl\_parentWidgetId := "" )

runs on EPTF\_UIHandler\_Private\_CT

return boolean

The XML has to be in a format that can be decoded by f\_EPTF\_UIHandler\_XSD\_decodeXUL function. If the XML contains iterators, external values, external data and conditions, they are processed by the UIHandler which uses the DataSource feature to determine the value of the external data. This means that it is possible to put content to the GUI for which the value is determined dynamically. For example without knowing how many traffic cases are defined, it is possible to define an XML that puts the CPS value for all traffic cases to the GUI.

The enabled/disabled state of the widget is determined automatically from the subsCanAdjust flag of the EPTF Variable containing the data for the data source element. See the relevant section of [6].

Here is a simple XML that puts some variables to the GUI:

<window xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' height='386.000000' id='EPTF\_Main\_Window' orientation='vertical' title='TTCN constructed window' width='820.000000'>

<hbox id='EPTF\_Main\_hbox' orientation='vertical'>

<hbox id='Params' orientation='vertical'>

<label disabled='false' flex='0.000000' value='Available variables' />

<textbox id='Var.AllVars.value' readonly='true' value=''>

<externaldata element='VarValue' source='VarProvider'>

<params>

<dataparam name='VarValue' value='allVar'></dataparam>

</params>

</externaldata>

</textbox>

<iterator element='VarList' id='VarList' source='VarProvider'>

<label disabled='false' flex='0.000000' value='Value of %VarList% :'/>

<textbox id='Var.%VarList%.value' readonly='true' value=''>

<externaldata element='VarValue' source='VarProvider'>

<params>

<dataparam name='VarValue' value='%VarList%'></dataparam>

</params>

</externaldata>

</textbox>

</iterator>

</hbox>

</hbox>

</window>

In the example above there is a data source registered with the dataSource name “VarProvider”. This data source supports the following elements: “VarValue” and “VarList”. The “VarValue” element returns the value of a given variable of which the name is given in the parameters. The name of the parameter is also called “VarValue”.   
 In this XML in the first hbox the value of the “allVar” variable is shown. Then there is a “VarList” iterator (element is “VarList”) with id “VarList”. This iterator iterates through all the variables in “VarProvider”. The value of the iterator is accessible by the iterator id written between “%”s like %VarList%. In the example this is given to the external data “VarValue” as the parameter.   
 So the example creates textboxes for all variables in “VarProvider” and puts their values into them. A label is placed before the textbox with the name of the variable. The value shown in the textbox will change dynamically if the value of the variable changes.

There is also the possibility to access the index of the iterator by writing “::idx” added to the iterator id between “%”s like %VarList::idx%. It can be used for example in widget ids where the value of the iterator cannot be used (because of illegal characters or any other reason).   
 Writing “::count” added to the iterator id between “%”s like %VarList::count% means the number of iterated elements in the iterator. This can be used for example in conditions where if this number is lower than a specific number then put some widget onto the GUI, else do something else.

The externalvalue element works as an iterator on a single value. The attributes and parameters are the same as in case of the iterators. The %ID% reference returns the given variable value in charstring format. The %ID::ref% returns the variable name. Can be used in all places, where the iterators can.

The values of iterators and externalvalues can change during running. If this happens, the gui is re-organized according to the change, so widgets can appear or disappear.

There is a special widget type called “htmlcode”. This widget won’t appear in RuntimeGUI, only in the browser. It is possible to insert any html code to this value of this widget. The written html code has to be escaped, because the supported TITAN version does not allow the unescaped characters. Here is an example of using the html code widget:

<window xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' height='386.000000' id='EPTF\_Main\_Window' orientation='vertical' title='TTCN constructed window' width='820.000000'>

<hbox id='EPTF\_Main\_hbox' orientation='vertical'>

<htmlcode>

<value>&lt;a href=”http://ttcn.ericsson.se”&gt;LINK&lt;/a&gt;</value>

</htmlcode>

</hbox>

</window>

It is possible to use externaldata in “htmlcode” widget. The value of the html code will be replaced by the value of the externaldata.

#### Converting Widgets to/from charstring in XML format

The Widget type defined in the XTDP protocol module can be converted to charstring in XML format by the function:

public function f\_EPTF\_UIHandler\_XSD\_encodeXUL(in Widgets pl\_widgets)

return charstring{

var octetstring vl\_ret := enc\_Widgets(pl\_widgets);

return oct2char(vl\_ret)

}

To decode the XML into the TTCN-3 Widgets type the following function can be used:

public function f\_EPTF\_UIHandler\_XSD\_decodeXUL(in charstring pl\_widgets)

return Widgets{

var Widgets vl\_ret;

var integer vl\_dummy;

vl\_dummy := dec\_Widgets(char2oct(pl\_widgets), vl\_ret);

return vl\_ret;

}

### Progress information during startup

UIHandler provides dataSources to display progress information during startup. The “progressBar” dataSource can be changed from any component in case of a new progress information should be displayed.

On the UIHandler component the function

public function f\_EPTF\_UIHandler\_updateProgress(in charstring pl\_progressInfo, in float vl\_progressPercent := -1.0) runs on EPTF\_UIHandler\_Private\_CT

can be used to update the progress information.

All changes to the “progressBar” dataSource are accumulated into the “progressBarHistory” dataSource, with each value is written into a separate line. This is done automatically by the UIHandler. The “progressBarHistory” dataSource is read-only.

It is possible to switch on/off the UIHandler’s own progress information by calling the function

public function f\_EPTF\_UIHandler\_enableOwnProgress(in boolean pl\_enableOwnProgress) runs on EPTF\_UIHandler\_Private\_CT

after the UIHandler is initialized. If own progress is disabled, progress information of UIHandler will not show up in the progressBar history, whereas changes set by other components will.

### Convenience functions in UIHandlerClient

These functions allow an easy way to create instances of the most commonly used widget types on the GUI. The following functions are provided by UIHandlerClient component:

The f\_EPTF\_UIHandlerClient\_XUL\_addButton() function allows to put a new button to the GUI.

f\_EPTF\_UIHandlerClient\_XUL\_addSpacer() function adds a new spacer to a parent widget on the GUI.

f\_EPTF\_UIHandlerClient\_XUL\_addLabel() function creates a new text label.

f\_EPTF\_UIHandlerClient\_XUL\_addEmptyHbox() function creates a new horizontal box.

f\_EPTF\_UIHandlerClient\_XUL\_addEmptyTabbox() function creates a new tabbed box.

f\_EPTF\_UIHandlerClient\_XUL\_addTabpanel() function creates a new tab panel.

f\_EPTF\_UIHandlerClient\_XUL\_addTab() function creates a new tab on a tabbed box.

f\_EPTF\_UIHandlerClient\_XUL\_addEmptyChart() function creates a new chart.

f\_EPTF\_UIHandlerClient\_XUL\_addTrace2Chart() function adds a new trace to a chart.

## HostAdminUI Functions

The HostAdminUI component named EPTF\_HostAdminUI\_CT can be used as the original HostAdmin component, but it provides an extra function, that can be used to start the component with

Starting the HostAdminUI component with the function   
f\_EPTF\_HostAdminUI\_behavior(in charstring pl\_selfname, in charstring pl\_hostname, in EPTF\_UIHandler\_CT pl\_UIHandler)

will result variables put by the component and measure the CPU load.

## LoadRegulator UI Functions

The LoadRegulatorUI component named EPTF\_LoadRegulatorUI\_CT can be used as the original LoadRegulator component, but it provides an extra function, that can be used to put the Regulator to the GUI.

The function f\_EPTF\_LoadRegulatorUI\_putUI(in charstring pl\_parentid) can be used to put its own widgets to the GUI. This widget is a simple box, it could be placed anywhere. Therefore a parent Id should be specified.

## StatCaptureUI Functions

The StatCaptureUI component named EPTF\_StatCaptureUI\_CT can be used as the original StatCapture component, but it provides an extra function, that can be used to put the Capturer to the GUI. It is only capable to display its configuration.

The function f\_EPTF\_StatCaptureUI\_putUI(inEPTF\_StatCapture\_CaptureGroupsConfig pl\_groups)   
can be used to put its own widgets to the GUI. This widget is a tab with a tabpanel. The parent widget’s Id could be configured thru a module parameter, also the configuration parameter to the function should be the proper module parameter of the StatCapture feature.

## XTDP operation modes of UIHandler

While the XTDP protocol [7] is symmetrical protocol between the GUI and its handler component, the user shall decide the mode for the XTDP connection initialization. The party who opens the XTDP-listener port and awaits for incoming connections is called the XTDP server. The party who connects to this XTDP-listener port is called the XTDP-client.

The UIHandler component can act as either an XTDP-server, or as an XTDP-client. The two modes are different from the point of view of handling XTDP-authentication on one hand and XTDP-connection loss from the other hand.

After the XTDP ports are connected, the XTDP session is always started from the Runtime GUI. Runtime GUI sends the XTDP-HandshakeRequest which is then answered by the UIHandler. After that if enabled, the XTDP-authentication takes place. UIHandler sends the XTDP-Authentication Request to the Runtime GUI with a challenge question configured in tsp\_EPTF\_UIHandler\_Authmsg. The response can be entered on the Runtime GUI and is sent back to the UIHandler which checks if it matches with the configured value given in tsp\_EPTF\_UIHandler\_Authresp. After successful authentication the XTDP session is up. The authentication can be switched off in the configuration file by setting tsp\_EPTF\_UIHandler\_Authresp to empty string.

### XTDP-client mode

The XTDP-client mode is the default behavior. In this mode the Runtime GUI opens the XTDP-listen port and waits for incoming connection.

The TitanSim executable initiates the connection (via the UIHandler). Upon XTDP connection-loss — which can happen if the GUI has been shut-down by the user, for example — it enters *headless mode*, see Section 5.8. During the headless mode it periodically attempts to reconnect.

In order to avoid long timeouts in XTDP connection attempts, UIHandler first “pings” the server via UDP. It attempts to initiate an XTDP-connect only after the GUI starts to answer to these “pings”.

On incoming XTDP connection XTDP handshake and if enabled, authentication is performed: the UIHandler will prompt (challenge string) the user for a password (response string) via the GUI.

Both the challenge and the response strings are read from modules parameters, given in the configuration file.

By default the XTDP authentication is turned off.

### XTDP-server mode

In server mode of the UIHandler, the XTDP port will be configured to act like a server, so it will listen on the specified port and wait for a Runtime GUI to connect. Since it might be quite possible that several TitanSim application is running on the same network concurrently, it might happen that a user connects its GUI to a wrong TitanSim application.

To minimize the impact of such misconfiguration problems, an authentication is possible: the UIHandler will prompt (challenge string) the user for a password (response string) via the GUI.

Both the challenge and the response strings are read from modules parameters, given in the configuration file.

By default XTDP-server mode is turned off.

## Shadow GUI

In server mode, it is possible to connect multiple Runtime GUIs to the TTCN application. The first connected GUI will be a Master GUI, which can control the execution. UIHandler will accept XTDP massages only from the Master GUI, but will send out its messages to every GUI connected. This way, the “shadowed” GUIs are only able to monitor the execution. In case the Master GUI disappears, the next connected GUI becomes the Master GUI for UIHandler.

## Headless mode

Headless mode is a feature of the UIHandler functionality, what allows the user to continue work or tests even when the Runtime GUI disconnected from the test executable. If headless mode is disabled, the UHandler will start up only if it can establish connection and XTDP authentication with the Runtime GUI during initialization. If the Runtime GUI is closed during test execution UIhandler will enter into headless mode automatically regardless of the headless mode setting (tsp\_EPTF\_UIHandler\_headlessmode). In headless mode when the GUI is not present, the UIHandler will not perform any XTDP message passing, but will keep track of the changes that are supposed to affect the GUI. Thus, when the GUI (re-)connected to the running test executable again, it will show the actual layout and data on the GUI.

If the UIHandler is configured to be an XTDP-client to the GUI, auto reconnection will be performed if the GUI is present again. By default it is turned on.

## CLI handling

Users can handle the values of the shared EPTF Variables of the UIHandlerClient components via TELNET terminal. The names of the EPTF Variables in these commands are the names with which the UIHandler component created its subscribing EPTF Variables. (see pl\_destParamName parameter of the subscription ordering functions 4.1in Section 4.1)

The commands are case independent.

### Data manipulation

#### Set the value of a variable

**Command:** SET

**Syntax:** SET variable\_name := value

**Purpose:** sets the value of a variable.

#### Get the value of a variable

**Command:** GET

**Syntax:** GET variable\_name

**Purpose:** gets the value of a variable and displays it on the command TELNET terminal, to which the user typed the command in.

#### List all variables and their values

**Command:** LS

**Syntax:** LS

**Purpose:** lists all GUI variables and their current values.

#### List read-only variables

**Command:** LSREADONLY

**Syntax:** LSREADONLY

**Purpose:** lists read-only GUI variables and their current values.

#### List writable variables

**Command:** LSWRITABLE

**Syntax:** LSWRITABLE

**Purpose:** lists GUI variables that can be set using the SET command and their current values.

### Monitor the values of variables

UIHandler CLI component periodically sends the value of the selected EPTF Variables to the display TELNET terminal in the form of:

variable\_name := value

Users can add EPTF Variables to the list of EPTF Variables to display, and remove from it.

#### Display periodically

**Command:** DISPLAY

**Syntax:** DISPLAY variable\_name

**Purpose:** Periodically displays the value of the variable on the display TELNET terminal.

#### Stop displaying

**Command:** HIDE

**Syntax:** HIDE variable\_name

**Purpose:** Stops periodical display the value of the variable on the display TELNET terminal.

**Command:** HIDE

**Syntax:** HIDE

**Purpose:** Stops displaying the values of all the EPTF Variables.

### Other commands

#### Help

**Command:** HELP

**Syntax:** HELP

**Purpose:** Displays a short help sting on the command TELNET terminal.

#### Quit

**Command:** QUIT

**Syntax:** QUIT

**Purpose:** Disconnects the command TELNET terminal.

#### Stop

**Command:** STOP

**Syntax:** STOP

**Purpose:** Stops the test execution and disconnects the command TELNET terminal.

# Feature GUIs

## HostAdminUI

It is possible, to use HostAdmin CLL feature with GUI support. When building the test executable with EPTF\_HostAdmin\_CT it acts exactly like the HostAdmin feature, but an extra parameter should be added at the usage: the UIHandler component reference. It will automatically put it’s own tab under the given parent, that can be specified via a tsp. On that tab, it will put out a table, and each HostAdmin in the system, will put out his own row to the table, that shows the name of the host, it is executed on and the average CPU load on that host.

See the documentation of the HostAdmin feature, and the reference in this document.

# Summary table of all public functions for UIHandler

Table 2 Summary of UIHandler functions

|  |  |
| --- | --- |
| Function name | Description |
| f\_EPTF\_UIHandler\_init\_CT | Initialises the EPTF\_UIHandler\_CT |
| f\_EPTF\_UIHandlerClient\_init\_CT | Initialises the EPTF\_UIHandlerClient\_CT component |
| f\_EPTF\_UIHandler\_CLI\_init\_CT | The initialization function of the EPTF\_UIHandler\_CLI\_CT component.. |
| f\_EPTF\_UIHandler\_clearGUI | Sends a message to user interface via XTDP port to remove all the elements. |
| f\_EPTF\_UIHandler\_addWindow | Adds a new window widget to the RuntimeGUI |
| f\_EPTF\_UIHandler\_addMainTabbox | Tries to add the main tabbox to the RuntimeGUI |
| f\_EPTF\_UIHandler\_addMainTabpages | Tries to add the TabPages to the RuntimeGUI |
| f\_EPTF\_UIHandler\_XSD\_addElementToGui | if the element to add to the Runtime Gui proves to be valid (will not destroy consistency), function sends out singleAddElementToGUI XTDP addRequest to the Gui and also adds this new element to GUI item list. |
| f\_EPTF\_UIHandler\_XSD\_encodeXUL | Encodes XTDP widgets the string |
| f\_EPTF\_UIHandler\_XSD\_decodeXUL | Decodes XTDP widgets from string |
| f\_EPTF\_UIHandler\_removeElementFromGui | Removes an element from the GUI. |
| f\_EPTF\_UIHandler\_snapshot | Makes a snapshot of the GUI as a charstring. |
| f\_EPTF\_UIHandler\_snapshotWindow | Makes a snapshot of the GUI's root widget (windowelem) as a charstring. |
| f\_EPTF\_UIHandler\_XSD\_encodeWindowXUL | Encodes a XUL Window record to its charstring representation |
| f\_EPTF\_UIHandler\_XSD\_encodeXULEm | Encodes an Embedded XUL record to its charstring representation |
| f\_EPTF\_UIHandler\_preprocessAliasXUL | Preprocess alias items in XUL charstring |
| f\_EPTF\_UIHandler\_XSD\_decodeXULEm | Decodes a XUL record to its charstring representation |
| f\_EPTF\_UIHandler\_saveLayout | Saves the parameter charstring as a window layout. |
| f\_EPTF\_UIHandler\_convertLayout | Converts the charstring retreived by the f\_EPTF\_UIHandler\_snapshot function to a storeable format. |
| f\_EPTF\_UIHandler\_revertLayout | Restores the charstring converted by the f\_EPTF\_UIHandler\_convertLayout to a charstring that can be given to |
| f\_EPTF\_UIHandler\_requestLayout | Sends a layoutRequest message to the RuntimeGUI and builds the database of the widgets by the response |
| f\_EPTF\_UIHandler\_getGUIAddrPorts | function to get address, port and reconnect port of the GUI to connect. |
| f\_EPTF\_UIHandler\_setGUIAddrPorts | function to set address , port and reconnect port of the GUI to connect. |
| f\_EPTF\_UIHandler\_setGuiReconnectPort | function to set the port of the GUI to reconnect. |
| f\_EPTF\_UIHandler\_setGuiHostPort | function to set address of the GUI to connect. |
| f\_EPTF\_UIHandler\_logAll | Logs all internal variables in EPTF\_UIHandler\_CT. |
| f\_EPTF\_UIHandler\_logWidgetDB | Logs all widgets database in EPTF\_UIHandler\_CT. |
| f\_EPTF\_UIHandler\_log2strWidgetRec | Logs Widget Record |
| f\_EPTF\_UIHandler\_GUIConnected | Returns whether the GUI is connected (authenticated) |
| f\_EPTF\_UIHandler\_initialized | Returns whether the GUI is initialized |
| f\_EPTF\_UIHandler\_exitButtonPressed | Handles the pressing of the exit button. Only for TESTING purposes! |
| f\_EPTF\_UIHandler\_str2Integer | Converts a string containing non-digit char separated numbers into a record of integer |
| f\_EPTF\_UIHandler\_str2IntegerList | Converts a string containing non-digit char separated numbers into a record of integer |
| f\_EPTF\_UIHandler\_str2FloatList | Converts a string containing non-digit char separated numbers into a record of floats |
| f\_EPTF\_UIHandler\_str2bool | function to convert a string value to a boolean value |
| f\_UIHandler\_setHandshakeMaxTime | Sets the max time for the XTDP handshake |
| f\_UIHandler\_setByeMaxTime | Sets max time for XTDP bye |
| f\_UIHandler\_setAuthMaxTime | Sets max time for XTDP authentication |
| f\_EPTF\_UIHandler\_enableOwnProgress | Enable/disable UIHandler’s own progress updates to the “progressBar” dataSource |
| f\_EPTF\_UIHandler\_updateOwnProgress | Update the value of the dataSource containing the progress information with the UIHandler's own progress info |
| f\_EPTF\_UIHandler\_updateProgress | Update the “progressBar” dataSource on the UIHandler component |
| f\_EPTF\_UIHandlerClient\_subscribeMe | Orders the default UIHandler to subscribe to a specified variable and optionally connect to a widget |
| f\_EPTF\_UIHandlerClient\_subscribeMeTo | Orders an UIHandler to subscribe to a specified variable and optionally connect to a widget |
| f\_EPTF\_UIHandlerClient\_subscribe | Orders the default UIHandler to subscribe to a specified variable on an other component and optionally connect it to a widget |
| f\_EPTF\_UIHandlerClient\_subscribeTo | Orders an UIHandler to subscribe to a specified variable on an other component and optionally connect it to a widget |
| f\_EPTF\_UIHandlerClient\_subscribeVariable | Orders the default UIHandler to subscribe to a specified variable |
| f\_EPTF\_UIHandlerClient\_subscribeVariableTo | Orders an UIHandler to subscribe to a specified variable |
| f\_EPTF\_UIHandlerClient\_removeElementFromGUI | Orders the default UIHandler to remove a widget from the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_removeElementFromGUITo | Orders an UIHandler to remove a widget from the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_enableGUIElement | Orders the default UIHandler to enable a widget on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_enableGUIElementTo | Orders an UIHandler to enable a widget on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_disableGUIElement | Orders the defalut UIHandler to disable a widget on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_disableGUIElementTo | Orders an UIHandler to disable a widget on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_setFocusToGUIElement | Orders the defalut UIHandler to set a widget being focused on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_setFocusToGUIElementTo | Orders an UIHandler to set a widget being focused on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_setFocusToTab | Orders the defalut UIHandler to select a particular tab of a tabbox on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_setFocusToTabTo | Orders an UIHandler to select a particular tab of a tabbox on the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_widgetExists | Checks whether the widget exists on the runtime GUI connected to the default UIHandler |
| f\_EPTF\_UIHandlerClient\_widgetExistsOn | Checks whether the widget exists on the runtime GUI connected to the specified UIHandler |
| f\_EPTF\_UIHandlerClient\_defaultUIHandler | Returns the default UIHandler component |
| f\_EPTF\_UIHandlerClient\_logAll | Logs all internal variable in EPTF\_UIHandlerClient\_CT |
| f\_EPTF\_CLI\_Client\_split | Splits a string using a given separator |
| f\_EPTF\_UIHandlerClient\_XSD\_addElementToGUI | Orders the default UIHandler to add a widget to the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_XSD\_addElementToGUITo | Orders an UIHandler to add a widget to the runtime GUI connected to the UIHandler |
| f\_EPTF\_UIHandlerClient\_XUL\_addWidget | Adds a widget to the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_XSD\_addEmptyTabbox | Adds a new TabBox to the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_XSD\_addEmptyTabpages | Adds a new TabPage to the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_XSD\_addWidget | Adds a widget to the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_addButton | Puts a new button to the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_addSpacer | Adds a new spacer to a parent widget on the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_addLabel | Creates a new text label |
| f\_EPTF\_UIHandlerClient\_XUL\_addEmptyHbox | Creates a new horizontal box |
| f\_EPTF\_UIHandlerClient\_XUL\_addEmptyTabbox | Creates a new tabbed box |
| f\_EPTF\_UIHandlerClient\_XUL\_addTabpanel | Creates a new tab on a tabbed box |
| f\_EPTF\_UIHandlerClient\_XUL\_addTab | Creates a new tab on a tabbed box |
| f\_EPTF\_UIHandlerClient\_XUL\_addEmptyChart | Creates a new chart on the GUI |
| f\_EPTF\_UIHandlerClient\_XUL\_addTrace2Chart | Adds a new trace to a chart |
| f\_EPTF\_UIHandler\_VariableUI\_init\_CT | Initializes the VariableUI component |
| f\_EPTF\_UIHandler\_VariableUI\_putVarbyName | Puts a variable to the GUI given by name |
| f\_EPTF\_UIHandler\_VariableUI\_putconnections | Puts all the variables to the GUI |
| f\_EPTF\_UIHandler\_VariableUI\_putlastconnections | Put the last given variables to the GUI. |
| f\_EPTF\_UIHandler\_VariableUI\_addsimpleconnection | Adds a simple connection(variable – widget pair) |
| f\_EPTF\_UIHandler\_VariableUI\_addwidgetconnection | Adds a connection with a widget |
| f\_EPTF\_UIHandler\_VariableUI\_addholdingwidget | Adds a holding widget to the database |
| f\_EPTF\_UIHandler\_VariableUI\_assignholdingwidget | Assigns a holding widget to a connection |
| f\_EPTF\_UIHandler\_VariableUI\_assignholding | Assigns a holding widget to a connection |
| f\_EPTF\_UIHandler\_VariableUI\_removeconnection | Removes a connection from the database |
| f\_EPTF\_UIHandlerXUL\_init\_CT | Initialises the EPTF\_UIHandler\_CT |
| f\_EPTF\_UIHandler\_Browser\_init\_CT | Initialize the host information to browser GUI |
| f\_EPTF\_UIHandler\_Browser\_enableWelcomeScreen | Enable welcome screen, when the application starts by setting the v\_EPTF\_UIHandler\_Browser\_enableWelcomeScreen component variable to true. |
| f\_EPTF\_UIHandler\_Browser\_disableWelcomeScreen | Disable welcome screen, when the application starts by setting the v\_EPTF\_UIHandler\_Browser\_enableWelcomeScreen component variable to false. |
| f\_EPTF\_UIHandler\_Browser\_getRemoteHosts | This function returns the Host names and ports on which the BrowserGUI is activated. |
| f\_EPTF\_UIHandler\_Browser\_addRemoteHosts | This function adds the given Host name and port pairs to the list on which the BrowserGUI is activated and activates them as well. |
| f\_EPTF\_UIHandler\_Browser\_setRemoteHosts | This function sets and activates the given Host name and port pairs on which the BrowserGUI is listening. |
| f\_EPTF\_UIHandler\_Browser\_clearRemoteHosts | This function clears the host name and ports on which the BrowserGUI is listening. |
| f\_EPTF\_UIHandler\_Browser\_removeRemoteHosts | This function removes and deactivates the given Host name and port pairs on which the BrowserGUI is listening. |
| f\_EPTF\_UIHandler\_Browser\_getCustomStyle | This function returns the active custom CSS style of the BrowserGUI. |
| f\_EPTF\_UIHandler\_Browser\_addCustomStyle | This function adds the given CSS Style to the main.css from file. |
| f\_EPTF\_UIHandler\_Browser\_setCustomStyle | This function sets the custom style (CSS) from file. |
| f\_EPTF\_UIHandler\_Browser\_clearCustomStyle | This function clears the custom style CSS elements. |
| f\_EPTF\_UIHandler\_Browser\_removeCustomStyle | This function removes the custom style (CSS) based on filename. |
| f\_EPTF\_UIHandler\_Browser\_addCustomStyleStr | This function adds the given CSS Style to the main.css from string. |
| f\_EPTF\_UIHandler\_Browser\_setCustomStyleStr | This function sets the custom style (CSS) from string. |
| f\_EPTF\_UIHandler\_Browser\_removeCustomStyleStr | This function removes the custom style string (CSS) based in string. |
| f\_EPTF\_UIHandler\_Browser\_setBrowserDirectory | This function sets the BrowserGUI directory, where the .png, .css, .js and several other files are, needed by the BrowserGUI feature. |
| f\_EPTF\_UIHandler\_Browser\_getBrowserDirectory | This function returns with the BrowserGUI directory string, where the .png, .css, .js and several other files should be, needed by the BrowserGUI feature. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_init | Function to initialise an empty <EPTF\_BASE\_TYPE##RingBuffer>. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_size | Function to ask the size of the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_capacity | Function to ask the capacity of the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_empty | Checks whether the buffer is empty. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_front | Function to get the first (eldest) element from the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_back | Function to get the last (latest) element from the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_push\_back | Function to store a new data item at end of buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_push\_front | Function to store a new data item at beginning of the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_pop\_front | Function to drop the first data item. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_pop\_back | Function to drop the last data item. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_clear | Function to remove all data from the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_get | Function to perform an unchecked random access to the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_at | Function to perform a secure access to the buffer. |
| f\_EPTF\_UIHandler\_ChartDataType\_RB\_dump | Function to dump the content of the buffer into a list. |
| f\_EPTF\_UIHandler\_CLI\_init\_CT | The initialization function. |
| f\_EPTF\_UIHandler\_CLI\_sendUserMessage | Send a custom message to be displayed to the user. |
| f\_EPTF\_UIHandler\_CLI\_logAll | Log all internal variables in EPTF\_UIHandler\_CLI\_CT component. |
| f\_EPTF\_UIHandler\_simulation\_createDBFromFile |  |
| f\_EPTF\_UIHandler\_simulation\_encode\_directContentDB | Encodes a EPTF\_UIHandler\_simulation\_directContentDB to its charstring representation |
| f\_EPTF\_UIHandler\_simulation\_decode\_directContentDB | Decode a charstring to EPTF\_UIHandler\_simulation\_directContentDB |
| f\_EPTF\_UIHandler\_XULWidgetId2charstr | Converts a widget ID to a charstring |
| f\_EPTF\_UIHandler\_CLIClient\_init\_CT | The initialization function. |
| f\_EPTF\_UIHandler\_CLIClient\_logAll | Log all internal variables in EPTF\_UIHandler\_Private\_CT component. |
| f\_EPTF\_UIHandler\_CLIClient\_sendUserMessage | Sends a custom message to the display terminal |
| f\_EPTF\_UIHandler\_readFileToCharstring | Loads a XUL describing a window layout and creates the described GUI. |
| f\_EPTF\_HostAdminUI\_behavior | Puts out a statistic row to its table and starts measuring |
| f\_EPTF\_LoadRegulatorUI\_putUI | Puts out the widgets of the LoadRegulatorUI |
| f\_EPTF\_StatCaptureUI\_putUI | Puts out the widgets of the StatCaptureUI |