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

EPTF CLL DataSource, Function Description

Contents

[1 Introduction 2](#_Toc453599285)

[1.1 Revision history 2](#_Toc453599286)

[1.2 How to read this document 2](#_Toc453599287)

[1.3 Scope 2](#_Toc453599288)

[1.4 Recommended way of reading 2](#_Toc453599289)

[1.5 Typographical conventions 2](#_Toc453599290)

[1.6 Abbreviations 3](#_Toc453599291)

[1.7 Terminology 3](#_Toc453599292)

[2 General Description 3](#_Toc453599293)

[3 Detailed description 5](#_Toc453599294)

[3.1 The DataSource API 6](#_Toc453599295)

[3.1.1 Initialization 6](#_Toc453599296)

[3.1.2 Data source registration 6](#_Toc453599297)

[3.1.3 Getting data 8](#_Toc453599298)

[3.1.4 Checking the existence of data elements 9](#_Toc453599299)

[3.1.5 Getting conditions 10](#_Toc453599300)

[3.1.6 Getting data values 12](#_Toc453599301)

[3.1.7 Setting data values 13](#_Toc453599302)

[3.1.8 Non-blocking functions 14](#_Toc453599303)

[3.1.9 Function to process help information 15](#_Toc453599304)

[3.2 DataSource CLI 16](#_Toc453599305)

[3.2.1 Initialization 16](#_Toc453599306)

[3.2.2 DataSource CLI Commands 16](#_Toc453599307)

[4 References 19](#_Toc453599308)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2011-08-22 | PA1 | First draft version | EBALLUG |
| 2011-12-08 | PB1 | Added built-in condition: “dataElementPresent” | ETHJGI |
| 2012-07-03 | PC1 | CLI integration | ETHJGI |
| 2012-07-20 | PC2 | getData added for DataSourceClient | ETHJGI |
| 2013-01-14 | PD1 | Ds getlist added to CLI | EMIHMIK |
| 2015-03-26 | PE3 | getDataValue + help | ETHJGI |
| 2015-05-15 | PE4 | ds info command returning the help | ETHJGI |
| 2015-05-18 | PE5 | Documentation updated | ESZILSZ |
| 2015-05-20 | PE6 | Graphics replaced with new ones | ESZILSZ |
| 2015-06-19 | E | Updated for release and graphics changed back to the old ones. | ESZILSZ |
| 2016-05-18 | PF1 | 3.1.6 Chapter updated | ESZILSZ |
| 2016-06-09 | F | Updated for release | ESZILSZ |

## How to read this document

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

## Scope

This document is to specify the content and functionality of the DataSource feature of the EPTF CLL.

## Recommended way of reading

The readers are supposed to get familiar with the concept and functionalities of EPTF CLL [4]. They should get familiar with the list of acronyms and the glossary in sections 1.6 and 1.7, respectively.

## Typographical conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context. Moreover, whenever a concept is mentioned that has a special meaning as described in the Terminology (section 1.7) of this document, then these occurrences are marked with an initial arrow, for example, *🡪 TitanSim Statistics*.

## Abbreviations

CLI Command Line Interface

CLL Core Library

EPTF Ericsson Performance Test Framework

GUI Graphical User Interface

PTC Parallel Test Component

TitanSim New synonym for the EPTF Framework

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

## Terminology

*EPTF 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.

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

# General Description

This data source feature was developed for the flexible GUI configuration feature.

The feature can be used to share data between the data source client components and the data source main component. The DataSource clients can register data sources into the system, and the main DataSource component or another DataSource client component can access the data. The data source is identified with a feature name (like LGenBase), and a PTC name (like LGen1). When the main data source component is requested to get a certain data from the data source clients, this identification information should be provided together with the name of the data (like TrafficCaseCPS) and additional parameters (for example the name of the traffic case: TC1).

The main data source component provides external data, iterators and conditions.

For external data and iterators an EPTF Variable name is returned. This variable stores the value of the requested data on the component where the request was made. This variable is subscribed to the source variable on the component which provides the data (in this case the CPS of TC1 on LGen1) with the specified subscription mode and refresh rate.

The type of value of the EPTF Variable that stores the value of the data is not restricted. It can be anything, like float or CharstringList that the EPTF Variable supports. It depends on the data source client that provides the variable. This means that even values containing certain lists can also appear as data sources.

For conditions the value of the condition is requested from the data source client and its value is returned as a boolean value.

It is also possible to request the data value directly, without using EPTF Variables. In addition, the provider has the possibility to deliver data which is not based on EPTF Variables.

Additional details can be found in chapter 3.

# Detailed description

The DataSource feature contains the definition of the main component and the client. The clients register the data sources which can be accessed on the main component or another client later on:

Register data source

PTC#m

EPTF\_DataSourceClient\_CT

PTC#n

EPTF\_DataSourceClient\_CT

EPTF\_DataSource\_CT

Register data source

Get data

Data available here

Data value

Variable subscription

Data requested from main component

Register data source

PTC#m

EPTF\_DataSourceClient\_CT

PTC#n

EPTF\_DataSourceClient\_CT

EPTF\_DataSource\_CT

Register data source

Get data

Get data

Data available here

Data value

Data value

Variable subscription

Data requested from client component

## The DataSource API

### Initialization

Both the DataSource and the DataSourceClient component have to be initialized before other API functions are used.

The initialization is done by the

function f\_EPTF\_DataSource\_init\_CT(in charstring pl\_selfName) runs on EPTF\_DataSource\_CT

and

function f\_EPTF\_DataSourceClient\_init\_CT(in charstring pl\_selfName, in EPTF\_DataSource\_CT pl\_DataSourceCompRef) runs on EPTF\_DataSourceClient\_CT

functions respectively.

### Data source registration

The registration of data sources can be done on the DataSourceClient using the function:

f\_EPTF\_DataSourceClient\_registerData(

in charstring pl\_source,

in charstring pl\_ptcName,

in fcb\_EPTF\_DataSourceClient\_dataHandler pl\_dataHandler,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT

Here the pl\_dataHandler parameter is a callback function with the following signature:

type function fcb\_EPTF\_DataSourceClient\_dataHandler(

out charstring pl\_dataVarName,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_CharstringList pl\_params

) runs on self return integer;

This callback function is called when the data value is requested for a given data element with the specified parameters. The callback function has to return in its pl\_dataVarName argument the name of the EPTF\_Variable that contains the data value. If the parameters are incorrect, the function has to return a positive error code, otherwise zero. The parameters are ordered lexicographically according to their names.

The registration of conditions can be done on the DataSourceClient using the function:

f\_EPTF\_DataSourceClient\_registerCondition(

in charstring pl\_source,

in charstring pl\_ptcName,

in fcb\_EPTF\_DataSourceClient\_conditionHandler pl\_conditionHandler,

in charstring pl\_method := "",

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT

Here the pl\_conditionHandler parameter is a callback function with the following signature:

type function fcb\_EPTF\_DataSourceClient\_conditionHandler(

out boolean pl\_conditionValue,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_method,

in EPTF\_DataSource\_Params pl\_params

) runs on self return integer;

This callback function is called when the condition value is requested for the registered method with the source and ptc name specified in the registerCondition function. The callback function has to return in its pl\_conditionValue argument the value of the condition. If the parameters are incorrect, the function has to return a positive error code, otherwise zero. If the pl\_method argument is not specified in the registration, the "refers(ModuleName.functionName)" (=log2str(pl\_sourceCompRef)) is used as a default. The parameters are ordered lexicographically according to their names.

To register data sources that do not require EPTF Variables the function

f\_EPTF\_DataSourceClient\_registerDataValue(

in charstring pl\_source,

in charstring pl\_ptcName,

in fcb\_EPTF\_DataSourceClient\_dataValueHandler pl\_dataValueHandler,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT

can be used. The handler function given in the pl\_dataValueHandler argument has the signature:

type function fcb\_EPTF\_DataSourceClient\_dataValueHandler(

out EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params

) runs on self return integer;

This function returns the value of the requested data instead of an EPTF Variable name that stores the data. Since there is no EPTF Variable available for this kind of data, setting their value requires a new function to change the value.

To set the value of data sources that do not require EPTF Variables a setter function has to be registered with the following function:

public function f\_EPTF\_DataSourceClient\_registerSetDataValue(

in charstring pl\_source,

in charstring pl\_ptcName := "",

in fcb\_EPTF\_DataSourceClient\_setDataValueHandler pl\_setDataValueHandler,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT

The setter function is given in the pl\_setDataValueHandler argument and has the signature:

type function fcb\_EPTF\_DataSourceClient\_setDataValueHandler(

inout EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_IntegerList pl\_indexList := {}

) runs on self return integer

The parameter pl\_dataValue contains the new value to be set. This is an inout parameter. It makes possible that the function could alter the value to be set if necessary and return the value that was actually set. The type of this parameter has the same type that the EPTF Variables can store.

### Getting data

After a data source is registered, the data values can be accessed. To get a certain data from the source, the following function has to be called on the DataSource main component:

function f\_EPTF\_DataSource\_getData(

out charstring pl\_dataVarName,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_CharstringList pl\_params := {},

in EPTF\_Var\_SubscriptionMode pl\_subscriptionMode := tsp\_EPTF\_DataSource\_subscriptionMode,

in integer pl\_refreshRate := tsp\_EPTF\_DataSource\_refreshRate

) runs on EPTF\_DataSource\_CT return integer

The corresponding function on the DataSourceClient component:

function f\_EPTF\_DataSourceClient\_getData(

out charstring pl\_dataVarName,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_Var\_SubscriptionMode pl\_subscriptionMode := tsp\_EPTF\_DataSource\_subscriptionMode,

in integer pl\_refreshRate := tsp\_EPTF\_DataSource\_refreshRate,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT return integer

The data source can be selected by the pl\_source and the pl\_ptcName arguments. The actual data element can be specified by the pl\_element and pl\_params arguments. These functions create a variable that is subscribed automatically in the background to the variable on the DataSourceClient component which published the data with the specified subscription mode and refresh rate. The functions return the name of this variable in the pl\_dataVarName argument. The variable contains the value of the data. In case the parameters are invalid or no data source exists, the functions return a nonzero return value. If the return value is zero, it means that the returned variable name is valid and the function call was successful.

On the DataSourceClient component the parameter pl\_sourceCompRef specifies the DataSource component which is used to request the data from. If only one DataSource component was used during the registration this parameter can be omitted: this component will be used to get the data by default.

These functions can be called with the same arguments more than once. In this case, the variable name is retrieved from a database.

#### Built-in data elements

The DataSource feature provides data elements, iterators and conditions. The source ID for these built-in data is “DataSource”.

DataSource provides the following built-in data elements:

* "Sources": returns the variable that contains the list of registered data sources. Usage: pl\_source:=”DataSource”, pl\_element:=”Sources” and no parameters should be specified in pl\_params. The DataSource itself is present in the list even if no other data sources are registered. Each time a new DataSource client registration is performed the new source is added to the list if it is not there already. An example value of the variable: {"DataSource","ExecCtrl"}
* "PTCs": Data source keeps track of registered PTCs for all source types. Usage: pl\_source:=”DataSource”, pl\_element:=”PTCs” and there has to be one parameter in pl\_params with name ”Source” and the requested source type (for example: “StatHandler”) in the parameter value. The returned variable contains a list of PTC names that have been registered with the specified data source type. If no PTCs were registered yet, the variable’s value will be an empty charstring list.

### Checking the existence of data elements

After a data source is registered, the data values can be accessed. To check if a data element is present, the following function can be used on the DataSource component:

function f\_EPTF\_DataSource\_checkData(

out charstring pl\_dataVarName,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {}

) runs on EPTF\_DataSource\_CT return integer

The corresponding function on the DataSourceClient component is:

function f\_EPTF\_DataSourceClient\_checkData(

out charstring pl\_dataVarName,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT return integer

The parameters of these functions are similar to the getData functions.

These functions check if the given data source was registered, the specified data element is valid and that the variable that stores its value exists on the owner component (DataSourceClient).

These functions do not create the variable on the local component, that is they do not perform a subscription to the owner of the data even if it exists. The functions return a non-zero error code if the dataSource is not found by some reason. Zero error code means that the dataSource is available.

### Getting conditions

After a condition is registered, the value of the condition can be accessed. To get a certain condition from the source, the following function has to be called on the DataSource main component:

function f\_EPTF\_DataSource\_getCondition(

out boolean pl\_conditionValue,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_method,

in EPTF\_DataSource\_Params pl\_params := {}

) runs on EPTF\_DataSource\_CT return integer

The condition can be selected by the pl\_source and the pl\_ptcName arguments. The actual condition can be specified by the pl\_method and pl\_params arguments.

#### Built-in conditions

The DataSource feature itself provides built-in conditions. The source ID is the same as for the data elements. The method names are the followings (the name of the parameters does not matter):

* "==": Checks if the value of the two specified parameters are equal,
* "!=": returns true if the two parameters are not equal
* ">": returns true if the first of the two float parameters is greater than the first
* ">=": returns true if the first float parameter is greater or equal than the second
* "<": returns true if the second float parameter is greater than the first
* "<=": returns true if the second float parameter is greater or equal than the first
* "match": returns true if the if the first parameter matches with the second parameter pattern
* "not match": returns true if the if the first parameter does not match with the second parameter pattern
* "and": returns the logical ‘and’ of the two Boolean parameters
* "or": returns the logical ‘or’ of the two Boolean parameters
* "not": returns the logical ‘not’ of the parameter

**Note:** For these built-in conditions the parameters are not ordered lexicographically according to their names. The name of the parameter of these conditions is not relevant. However, there are conditions which need specific parameter names. These are the followings:

* "dataElementPresent": returns true if the specified data element is present somewhere in the system. This means that the given data source was registered, the specified data element is valid and the variable that stores its value exists. All these are checked by this condition and if any of them is not fulfilled the condition will be evaluated false. This condition does not make the data variable available on the DataSource component (i.e. the Variable subscription is not performed), but checks if the source variable is present on the owner (DataSourceClient) component.

Usage: pl\_source:=”DataSource”, pl\_method:=”dataElementPresent” and its parameters should be specified in pl\_params.

The pl\_params has to specify the following parameters (with parameter names):

* + "Source": the source of the data that is to be checked
  + "PTCName": optional parameter to specify the name of the PTC where the data source is located. If not given, the source component is located automatically (will be successful if only one PTC had registered the given source)
  + "Element": the element for the given source to be checked

If the data source to be checked has any parameters, they have to be specified as well. First the name of the parameter, then its value should be given in the pl\_params parameter in order as:

* + “ParamName”: the name of the parameter for the data source to be checked
  + “ParamValue”: the value of the parameter (with the name specified before) for the data source to be checked

Example:

The following condition checks if the "PTCs" element of the DataSource feature itself exists for the Source: "DataSource":

f\_EPTF\_DataSource\_getCondition(

pl\_conditionValue := vl\_conditionValue,

pl\_source := "DataSource",

pl\_ptcName := "",

pl\_method := "dataElementPresent",

pl\_params := {

{

paramName := "Source",

paramValue:= "DataSource" // the source to be checked

},

{

paramName := "Element",

paramValue:= "PTCs" // the element of the source to check

},

{

paramName := "ParamName",

paramValue:= "Source" // parameter name of the PTCs element

},

{

paramName := "ParamValue",

paramValue:= "DataSource" // the value of the Source param.

}

}

);

The same example in XML (the label is shown only if there are any PTCs with source ID "DataSource".The value of the label is the value of the data source of which presence was checked):

<condition element='dataElementPresent' id='conditionID' source='DataSource'>

<params>

<dataparam name='Source' value='DataSource' />

<dataparam name='Element' value='PTCs' />

<dataparam name='ParamName' value='Source' />

<dataparam name='ParamValue' value='DataSource' />

</params>

<insertif id='conditionID'>

<label value='' >

<externaldata element='PTCs' source='DataSource'>

<params>

<dataparam name='Source' value='DataSource' />

</params>

</externaldata>

</label>

</insertif>

</condition>

### Getting data values

It is possible to get only the value of a data element without using an EPTF Variable. In this case no Variable is created on the local component, no Variable subscription is done and even no source Variable is needed to provide data by the data source.

After a data source is registered, the data values can be accessed. All data can be requested this way, even the values of conditions. To get the value of a certain data from a source, the following function has to be called on the DataSource main component:

function f\_EPTF\_DataSource\_getDataValue(

out EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_CharstringList pl\_params := {}

in EPTF\_DataSource\_Filter pl\_filter := c\_EPTF\_DataSource\_Filter\_empty

) runs on EPTF\_DataSource\_CT return integer

The corresponding function on the DataSourceClient component:

function f\_EPTF\_DataSourceClient\_getDataValue(

out EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_DataSource\_Filter pl\_filter := c\_EPTF\_DataSource\_Filter\_empty,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT return integer

These functions work the same way as the getData functions, but can only be used when the value of the data is needed. These functions support data elements registered with f\_EPTF\_DataSourceClient\_registerData, f\_EPTF\_DataSourceClient\_registerCondition or f\_EPTF\_DataSourceClient\_registerDataValue functions. For conditions the method name has to be passed into the pl\_element parameter.

Data elements registered by f\_EPTF\_DataSourceClient\_registerDataValue can only be accessed by the getDataValue functions.

#### Built-in data elements which have only value

The DataSource feature provides data elements that have value only (no EPTF Variable exists that stores the value). The source ID for these built-in data is “DataSource”.

DataSource provides the following built-in data value elements:

* "help": returns the help information about any data source registered. Usage: pl\_source:=”DataSource”, pl\_element:=”help”. All parameters are optional. The parameter “format” defines the format of the output. It can be either “TEXT”, which means human readable format, or “JSON”, default: “JSON”. The parameter “Source” can be used to get help for a given source only. If not specified, help information for all available sources will be returned. The parameter “Element” can be used to get help information on a given element only. If not specified, help information of all elements are returned. If specified together with the “Source” parameter, only the help information of the corresponding elements of the specified source will be returned.
* “sizeOf”: returns the number of elements of another data source if its value is a list. Returns 1 for non-list type data sources. The data source can be specified via parameters.

### Setting data values

To set the value of a given data element, the following function is provided:

function f\_EPTF\_DataSource\_setDataValue(

inout EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_CharstringList pl\_params := {},

in EPTF\_IntegerList pl\_indexList := {}

) runs on EPTF\_DataSource\_CT return integer

The corresponding function on the DataSourceClient component:

function f\_EPTF\_DataSourceClient\_setDataValue(

inout EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_IntegerList pl\_indexList := {},

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT return integer

These functions can be used for all data elements that were registered by some source with the function f\_EPTF\_DataSourceClient\_registerData (that is when an EPTF Variable stores the data). However, the value of data elements registered by f\_EPTF\_DataSourceClient\_registerCondition or f\_EPTF\_DataSourceClient\_registerDataValue functions and data elements that have not been registered at all (that is all data elements for which no EPTF Variable exists) cannot be changed unless a value-setter handler function registered with f\_EPTF\_DataSourceClient\_registerSetDataValue makes it possible to change its value.

For data elements that have a list-type value (for example CharstringList), the parameter pl\_indexList can be used to filter out those elements from the list that should be changed directly. In this case, the number of elements of pl\_indexList should be the same as the number of elements of pl\_dataValue. For example if pl\_dataValue := {charstringlistVal := {”abc”,”def”}} and pl\_indexList := {2,5}, this means that the 2nd item in the list of the value of the data element will be changed to ”abc” and the 5th to ”def”. If pl\_indexList is not specified, the value of the whole list will be replaced with the value given in pl\_dataValue. But it is the responsibility of the setDataValue handler to support this kind of behavior.

### Non-blocking functions

Several DataSource API function has a non-blocking version on the main and the client component as well. These differ from the originals (that is the blocking versions) that instead of returning a result after the function call in an out parameter, a handler function has to be provided that will be called when the return value is available. The value requested is passed to the handler function together with the details of the data elements and the data provided by the user.

For example the non-blocking version of f\_EPTF\_DataSourceClient\_getData is:

public function f\_EPTF\_DataSourceClient\_getData\_nonBlocking(

in charstring pl\_source,

in charstring pl\_ptcName := "",

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params := {},

in EPTF\_DataSource\_GetDataHandler pl\_getDataHandler := cg\_EPTF\_DataSource\_GetDataHandler\_null,

in EPTF\_DataSource\_CT pl\_sourceCompRef := null

) runs on EPTF\_DataSourceClient\_CT return integer

The argument pl\_getDataHandler specifes the handler to be called back when the value is available. Its type is defined as:

type record EPTF\_DataSource\_GetDataHandler {

fcb\_EPTF\_DataSource\_getDataHandler getDataHandler,

EPTF\_IntegerList userData

}

The userData field is a list of integers provided by the user. These values will be passed to the handler function given in getDataHandler in the pl\_userData argument when invoked.

The signature of the handler function is:

type function fcb\_EPTF\_DataSource\_getDataHandler(

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params,

// reponse parameters:

in integer pl\_errorCode,

in charstring pl\_remoteDataVarName,

in EPTF\_Var\_CT pl\_ownerCompRef,

in integer pl\_localDataVarId,

in EPTF\_Var\_DirectContent pl\_dataValue,

in EPTF\_IntegerList pl\_userData

) runs on self;

### Function to process help information

On the DataSource main component the help information can be accessed by the following functions:

public function f\_EPTF\_DataSource\_getHelpTEXT(

in charstring pl\_source := "",

in charstring pl\_element := ""

) runs on EPTF\_DataSource\_CT return charstring

public function f\_EPTF\_DataSource\_getHelpJSON(

in charstring pl\_source := "",

in charstring pl\_element := ""

) runs on EPTF\_DataSource\_CT return octetstring

They return the help information in human readable or JSON format.

The following function is provided to generate the help information for user given data sources:

public function f\_EPTF\_DataSource\_handleHelp(

out EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in EPTF\_DataSource\_Params pl\_params,

in EPTF\_DataSource\_Help\_DataElementChildren pl\_dataElementHelp

) return integer

This function can be used in data value handlers that would like to provide help information for the “help” data element. An example data value handler that handles only the “help” data element might look like this:

function f\_USER\_DSProcessDataValue(out EPTF\_Var\_DirectContent pl\_dataValue,

in charstring pl\_source,

in charstring pl\_ptcName,

in charstring pl\_element,

in EPTF\_DataSource\_Params pl\_params)

runs on USER\_CT return integer{

var integer vl\_errorCode := -1;

pl\_dataValue := {unknownVal := {omit}}; // set it to invalid

select( pl\_element )

{

case(c\_EPTF\_DataSource\_dataElement\_Help) {

vl\_errorCode := f\_EPTF\_DataSource\_handleHelp(pl\_dataValue,pl\_source,pl\_params,c\_EPTF\_USER\_help);

}

case else

{

}

}

return vl\_errorCode;

}

In this example the constant c\_EPTF\_USER\_help defines the help information for all data elements supported by the user defined data source.

## DataSource CLI

The DataSource CLI is an extension of the DataSource that makes it possible to execute DataSource commands from the Command Line Interface. It acts as a CLI Client and registers the DataSource commands into the main CLI component.

### Initialization

The DataSource\_CLI can be initialized by the

function f\_EPTF\_DataSource\_CLI\_init\_CT(

in charstring pl\_selfName,

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

) runs on EPTF\_DataSource\_CLI\_CT

The parameter pl\_CLI\_compRef specifies the component reference of the main CLI component.

After initialization all DataSource commands are available on the telnet terminal that belongs to the given CLI component.

### DataSource CLI Commands

The main DataSource command name that is available in the telnet command line interface is ‘ds’. Its parameters are described below.

#### help: To get help

To get help about the DataSource CLI commands the following command can be used in the telnet terminal:

ds help

This lists the available options together with a short description and shows basic command examples and syntax.

#### get: To get the value of dataElements

The DataSource CLI provides a command to get the value of registered external dataElements such as iterators, externaldata and externalvalues with syntax similar to the custom GUI XML.

The following command returns the list of all DataSources registered to the DataSource feature. It uses the dataElement ‘Sources’ of the DataSource ‘DataSource’ for the ‘ds get’ command:

ds get <datadescription xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' element='Sources' source='DataSource'> </datadescription>

Notice that the dataElement has to be given in XML notation as a <datadescription> element. This element comes from the XSD of the XTDP protocol module used for the custom GUI XML. This is why the xmlns attribute has to be given as in the example.

This command works for iterators, externaldata and extrernalvalues as well.

The command, if successful, prints out the value of the dataElement. In case of any error a nonzero error code is shown together with a description of the problem.

#### getCondition: To get the value of conditions

Similarly to the external values, the value of conditions can also be retrieved. Instead of ‘ds get’ the command ‘ds getCondition’ can be used for conditions.

For example, matching the string “10” against the template “1\*” will look like this:

ds getCondition <datadescription xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' element='match' source='DataSource'> <params> <dataparam name='p1' value='10'/> <dataparam name='p2' value='1\*'/> </params> </datadescription>

This will return “true” since the pattern matches with the given string.

The command ‘ds getCondition’ returns either “true” or “false” depending on the value of the condition. In case of any problem the command prints out an error code together with the description of the problem.

**Note:** The same XML format can be used for externalValues and conditions, whereas in the custom GUI XML or in the DataSource API functions the name ‘method’ is used instead of the ‘element’ for conditions.

#### set: Set the value of dataElements

The value of the dataElements can be also set through the CLI.

The following command sets the value of the ‘MyData’ element of ‘DummySource’ source to the charstringList value {"A","B"} (this is its type). This dataElement has two parameters (‘p1’ and ‘p2’):

ds set <datadescription xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' element='MyData' source='DummySource'> <params> <dataparam name='p1' value='2.0'/> <dataparam name='p2' value='1.0'/> </params> </datadescription> {"A","B"}

**Note:** The value to be set has to be in a format identical to the one the corresponding ‘ds get’ command returns for the same dataElement. The value has to be placed after the <datadescription> parameter.

The command prints out the new value if it was successful, otherwise an error code is shown together with the description of the problem.

#### getVarName: Get the Name of the EPTF Variable that stores the value of the dataElement

The name of the variable that stores the value of the dataElements can be displayed by the ‘ds getVarName’ command. This command does not work for conditions, because there are no variables for conditions.

The following command prints out the name of the EPTF Variable that contains the list of PTCs that were registered as ‘StatManager’ DataSources. The command uses the ‘PTCs’ dataElement of the DataSource.

ds getVarName <datadescription xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl' element='PTCs' source='DataSource'> <params> <dataparam name='Source' value='StatManager'/> </params> </datadescription>

In case of error the command prints out a nonzero error code and the description of the problem.

#### getlist: Command to get the values of Datalements in a list.

The command has one datadescriptionlist parameter. The parameter has xml format, and it can contain any number of datadescription. The format of the datadescription can be seen in section [3.2.2.2](#_get:_To_get).

Here is an example to use getlist with datadescriptionlist parameter. This command returns the number of the started and the succeeded calls (tagged with new line characters for better visibility):

ds getlist <datadescriptionlist xmlns='<http://ttcn.ericsson.se/protocolModules/xtdp/xtdl>'>  
  <datadescription element='TcStat' source='ExecCtrl'>   
    <params>   
      <dataparam name="EntityGroup" value="DefaultEGrp" />  
      <dataparam name="Scenario" value="DefaultSc" />  
      <dataparam name="TrafficCase" value="DefaultTC1" />  
      <dataparam name="Statistic" value="Starts" />   
    </params>   
  </datadescription>  
  <datadescription element='TcStat' source='ExecCtrl'>   
    <params>   
      <dataparam name="EntityGroup" value="DefaultEGrp" />  
      <dataparam name="Scenario" value="DefaultSc" />  
      <dataparam name="TrafficCase" value="DefaultTC1" />  
      <dataparam name="Statistic" value="Success" />   
    </params>   
  </datadescription>  
</datadescriptionlist>

#### info: Request help information about dataElements

To get the help information of data elements present in the system, the following command can be used in the telnet terminal: ds info

This command collects the help information defined for all available data elements in the system.

However, it is possible to get the information for a specific source or a specific data element only. The following command will return the description of all data elements for the specified source, in this case for the source “SourceName”:

ds info S=SourceName

To get help information for a given data element for all sources that support it, the following command can be used. It will return the description of the specified element for any source that implements it:

ds info E=ElementName

If the help information is needed for a given element of a given source, it is possible to specify both parameters at the same time (in any order):

ds info E=ElementName S=SourceName

If the element name or the source name contains spaces, the name should be enclosed in apostrophes:

ds info S=’Source Name’ E=’Element Name’

**Note:** these commands can return help information that is defined according to the section 3.1.9. To get the list of sources that defines that help information for its elements, the following command can be used:

ds info E=help

# 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 CLL for TTCN-3 toolset with TITAN, Function Specification
4. 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)
5. 18/155 16-CNL 113 512  
   EPTF CLL Variable, Function Description