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

Contents

[1. Introduction 2](#_Toc468719302)

[1.1 Revision History 2](#_Toc468719303)

[1.2 How to Read this Document 2](#_Toc468719304)

[1.3 Scope 2](#_Toc468719305)

[1.4 Recommended Way of Reading 2](#_Toc468719306)

[1.5 Typographical Conventions 2](#_Toc468719307)

[1.6 Abbreviations 2](#_Toc468719308)

[1.7 Terminology 3](#_Toc468719309)

[2 General Description 3](#_Toc468719310)

[3 Communicating with the Data Source 3](#_Toc468719311)

[3.1 General HTTP server functions 4](#_Toc468719312)

[3.2 Data Source specific calls 4](#_Toc468719313)

[3.2.1 The getData request 4](#_Toc468719314)

[3.2.2 The setData request 5](#_Toc468719315)

[3.2.3 The members of the JSON requests 6](#_Toc468719316)

[3.2.4 The response 7](#_Toc468719317)

[4 APIs 8](#_Toc468719318)

[4.1 JavaScript 8](#_Toc468719319)

[4.2 Python 9](#_Toc468719320)

[4.3 TTCN 10](#_Toc468719321)

[5 Functional Interface 10](#_Toc468719322)

[5.1 Naming Conventions 10](#_Toc468719323)

[5.2 Public Functions 10](#_Toc468719324)

[5.2.1 Initialization of the DsRestAPI Feature 10](#_Toc468719325)

[5.2.2 Starting the DSServer, the HTTP server and creating symlinks 10](#_Toc468719326)

[5.2.3 Stopping the server 11](#_Toc468719327)

[5.2.4 Initialization of DSServer 11](#_Toc468719328)

[5.2.5 Processing JSON requests 12](#_Toc468719329)

[5.2.6 Collecting timelines 13](#_Toc468719330)

[6 References 13](#_Toc468719331)

# Introduction

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Rev** | **Characteristics** | **Prepared** |
| 2016-09-16 | PA1 | Renaming UIModel to DsRestAPI | EIMRENA |
| 2016-09-16 | PA2 | Corrected typos. | EIMRENA |
| 2016-10-13 | PA3 | Updated with Timeline sub-feature. | EIMRENA |
| 2016-12-02 | A | Updated for release | EIMRENA |

## How to Read this Document

This is the Function Description for the DsRestAPI feature of the Ericsson Performance Test Framework (EPTF), Core Library (CLL). EPTF Core Library is developed for the TTCN-3 [1] Toolset with TITAN [2].

## Scope

This document is to specify the content and functionality of the DsRestAPI 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 [3]. They should get familiar with the list of acronyms and the glossary in Section 1.6 and 1.7, respectively.

## Typographical Conventions

Important concepts are denoted by *italic* font wherever they are first used in the given context.

## Abbreviations

CLL Core Library

EPTF Ericsson Performance Test Framework

MTC Main Test Component

PTC Parallel Test Component

TitanSim New synonym for the EPTF Framework

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

GUI Graphical user interface

UI User interface

API Application programming interface

JSON JavaScript Object Notation

REST Representational State Transfer

## Terminology

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

# General Description

This document specifies the DsRestAPI feature of the CLL.

The DsRestAPI feature provides the following functionality:

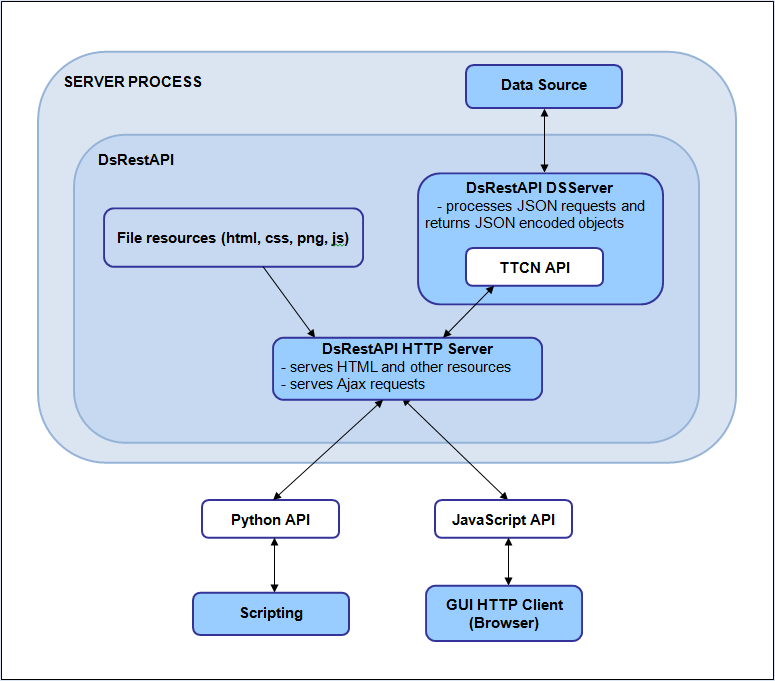
* Data can be obtained from the Data Source through HTTP calls and a TTCN API function.
* General HTTP requests are supported (GET, PUT)
* The HTTP server can be started and stopped

# Communicating with the Data Source

DsRestAPI contains API functions that can be used to obtain data in JSON format from the Data Source of CLL.

The feature contains a HTTP server. The server processes the incoming requests and uses a standalone TTCN API to retrieve the response from the Data Source. The communication with the server is done using either the JavaScript API or the Python API.

The following figure shows the flow of communication.



For more details about the CLL Data Source, please see [5].

## General HTTP server functions

The following actions are possible using HTTP calls:

* Issuing Data Source requests
* Loading files
* Saving files
* Listing the contents of a directory
* Creating a directory

## Data Source specific calls

JSON requests are sent to the HTTP server to communicate with the Data Source through the APIs.

The request can either be a *getData* or a *setData* request.

### The getData request

The format of the getData request is the following:

{  
 “getData”: {  
 “source”  
 “element”  
 “ptcname”  
 “filter”: {  
 “dataValue” or  
 “request”  
 }

“rangeFilter”: {

“offset”

“count”

}   
 “params”: [  
 {  
 “paramName”   
 “paramValue”  
 },  
 {  
 “paramName”  
 “paramValue”  
 }  
 ]  
 “selection”  
 “writableInfo”  
 “children”  
 “timeline”: {

“period”

“maxpoints”

“since”

}  
 }  
}

### The setData request

A setData request can have following members:

{  
 “setData”: {  
 “source”  
 “element”  
 “ptcname”  
 “params”  
 “content”  
 “tp”   
 “indxsInList”   
 }  
}

### The members of the JSON requests

* *source*: the name of the Data Source client;  
  a mandatory string
* *ptcname*: identifies the Data Source together with source;  
  an optional string
* *element*: the name of the element we want to query;  
  a mandatory string
* *rangeFilter*: an optional object that can be used to query parts of a long list. The *offset* is the first index in the list and the *count* is the number of maximum elements we want to be returned.
* *params*: the parameters needed to query the given element;  
  an optional object see [5] for more details about parameters.
* *paramName*: the name of the parameter;  
  a mandatory string member of every param object
* *paramValue*: the value of the parameter;  
  a mandatory string member of every param object
* *filter*: an optional object that can either have a dataValue member, which is a simple string, or a request member.  
  The request is an object that has a source, element, ptcname and params. The paramValues however are also filter objects, so it is possible to create more complex filters.
* *children*: requests can be nested into one another;  
  an optional list of requests
* *selection*: if the response to a request is a list only the subrequests of the given elements will be processed;  
  an optional list of integers
* *writableInfo*: check whether the corresponding data element is writable. If it is not writable, the response will contain a negative tp.
* *timeline*: get the collected timeline data
  + *period:* the number of seconds between measurements
  + *maxpoints:* the number of measurements to store
  + *since*: get only measurements after the given time
* *content*: in a setData request, change the value of the given element to this one;  
  a mandatory string
* *tp*: the type of the value as a mandatory integer member, see in EPTF\_DataSource\_ValueType

1. filtered element
2. intType,
3. floatType,
4. boolType,
5. charstringType,
6. octetstringType,
7. hexstringType,
8. bitstringType,
9. integerlistType,
10. floatlistType,
11. charstringlistType,
12. statusLEDType

A negative tp means that the element cannot be set.

* *indxInList*: if the response is a list, change only the elements located in the given indexes in the list  
  this member must be an integer list and it must be present if the response is a list

**Note**: The first four members (*source, ptcname, element, params*) define the element we want to query.

**Note**: The *children* and *selection* members can be used to create nested requests.

### The response

A response is also a JSON object with a single member: *contentList*. The contentList can contain two kinds of objects: *node* and *list*.

The *node* object:

{  
 “node”: {  
 “val”: string  
 “tp”: integer  
 “childVals”: [ node or list objects ]  
 }  
}

The *list* object:

{  
 “list”: [ node objects ]  
}

In summary, the response always has the following form:

{  
 “contentList”: [node or list objects]  
}

The actual response for a request is contained in the *val* member of the node objects as a string, and *tp* defines its type. The list of available types can be found in 3.2.3.

The *childVals* list contains the responses for the child requests.

If the response represents a timeline, it will be charstring type, whose value is a JSON string (where the quotation marks are escaped). The JSON contains the following members:

* tp: the type of the data
* x: the list of times measurements were taken
* y: the list of datapoints

# APIs

DsRestAPI provides three RESTful APIs that can communicate with the Data Source.

## JavaScript

The JavaScript API can be used in a web application.

Since the API functions are asynchronous, a callback function is called with the returned response in every case.

The following public functions can be used:

* getData(aHandler, aSource, aElement, aParams, aPtcName):  
  Issues a single getData request with the given members. The aHandler callback will be called with the response.
* setData(aHandler , aSource, aElement, aContent, aType, aParams, aPtcName, aIndxs):  
  Issues a single setData request with the given members and calls aHandler with the response.
* getList(aParam, aHandler):  
  Issues a list of requests contained in the aParam parameter.
* getHelp(callback, source, element):  
  Gets the help of the given Data Source and element. If the element is missing, the help of the Data Source client will be returned. If the source is also missing, the help of all Data Source Clients is returned.  
  See [5] for more details about the help.
* putFile(url, data, callback):  
  Saves the data in a file specified by the url.
* loadFile(url, callback):  
  Calls the callback function with the contents of the specified file.
* getDirectory(url, callback):  
  Calls the callback function with the contents of the given directory. The returned value is an object of the form {“contentList”: [dir\_contents]}.
* createDirectory(url, callback):  
  Creates the given directory and invokes the callback when server responds.
* importJsFile(file, callback):  
  Imports the given JavaScript file and calls the callback when finished.

## Python

The Python interface is similar to the JavaScript API but it does not contain the file specific functions and the functions are synchronous so the response is simply returned.

The following public functions are available:

* getData(aSource, aElement, aParams, aPtcName):  
  Issues a single getData request and returns the response.
* setData(aSource, aElement, aContent, aType, aParams, aPtcName, aIndxs):  
  Issues a single setData request and returns the response.
* getList(aParam):  
  Issues a list of requests contained in the aParam parameter.
* getHelp(aSource, aElement):  
  Gets the help of the given Data Source and element.  
  **Note**: see the corresponding JavaScript function.

## TTCN

See the description of the API function in 5.2.5.

# Functional Interface

Apart from this description a cross-linked reference guide for the EPTF Core Library Functions can be reached for on-line reading [4]. This chapter describes the public TTCN interface of the DsRestAPI feature.

## Naming Conventions

All functions have the prefix f\_EPTF\_DsRestAPI\_.

## Public Functions

### Initialization of the DsRestAPI Feature

Before using the EPTF DsRestAPI functions, the

f\_EPTF\_DsRestAPI\_init\_CT(pl\_selfName)

function should be called. This initializes the EPTF DsRestAPI feature. The name of the component should be passed as parameter pl\_selfName.

It does the following:

* initializes the needed CLL features (Base, DataSource, DsRestAPI DSServer, DsRestAPI Timeline)
* creates and initializes the required component variables
* registers the clean-up function of DsRestAPI (f\_EPTF\_DsRestAPI\_cleanup\_CT)

### Starting the DSServer, the HTTP server and creating symlinks

The

f\_EPTF\_DsRestAPI\_start(  
 in charstring pl\_hostIPAddress,  
 in integer pl\_hostPort,  
 in charstring pl\_HTTPServer\_dir := “”,  
 in charstring pl\_customizableApp\_dir := “”,  
 in charstring pl\_API\_dir := “”  
)

function can be used to start the DSServer and the HTTP server. The latter will use the given IP address and hostPort for communication.

The pl\_HTTPServer\_dir server root directory parameter is optional. If specified, the server will be able to serve resources from it, and if not, the server will still serve the RESTful JSON API.

When calling the function in case the argument pl\_HTTPServer\_dir is present, two symlinks are created if their location is given and is not an empty string.

If specified, the pl\_API\_dir should be the directory where the Javascript API for DsRestAPI can be found. The directory will be symlinked to pl\_HTTPServer\_dir/utils/DsRestAPI.

If specified, the pl\_customizableApp\_dir is a user created directory which may contain GUI elements to be used in a GUI. The directory will be symlinked to pl\_HTTPServer\_dir/CustomizableContent/CustomizableApp.  
If the symlink is created and the pl\_HTTPServer\_dir/CustomizableContent/CustomizableApp/TimelineRequest.json file exists, it will be appended to the Timeline collector. See 5.2.6 for more details.

### Stopping the server

The f\_EPTF\_DsRestAPI\_stop()function will stop the HTTP server and delete the created symlinks.

### Initialization of DSServer

DSServer is a standalone sub-feature that provides the TTCN API for the Data Source and can be used on its own. The f\_EPTF\_DsRestAPI\_init\_CT function automatically initializes the DSServer as well.

The f\_EPTF\_DsRestAPI\_DSServer\_init\_CT(pl\_selfName) function is used to initialize the DSServer when using it without the DsRestAPI feature. The name of the component should be passed as parameter pl\_selfName.

It does the following:

* initializes the needed CLL features (Base, DataSource)
* initializes the needed DsRestAPI sub-features:
  + DsRestAPI\_Filter that handles the filters
  + DsRestAPI\_Storage that caches requests and responses
* creates and initializes the needed component variables
* registers the clean-up function of DsRestAPI (f\_EPTF\_DsRestAPI\_DSServer\_cleanup\_CT)

### Processing JSON requests

The

f\_EPTF\_DsRestAPI\_DSServer\_processRequest(  
 in EPTF\_DsRestAPI\_Requests pl\_request  
) runs on EPTF\_DsRestAPI\_DSServer\_CT return EPTF\_DsRestAPI\_ContentList

function can be used to process a request. The function is synchronous.

The function needs a request set organized in a tree as an input. This request tree is built out of atomic getData or setData request elements, and their children, which are also getData or setData request elements.

The child request's parameters could depend on the result of its parent requests.

The API calls will also result in an object, representing the response values, organized also in a tree.

The structure of the result tree matches the structure of the request tree. The elements in the response tree are the resulted values of the getData and setData elements located on the corresponding places of the request tree.

f\_EPTF\_DsRestAPI\_DSServer\_processJSONRequest(  
 in octetstring pl\_request  
) runs on EPTF\_DsRestAPI\_DSServer\_CT return octetstring

function can be used to process a JSON encoded request. The request will be decoded and the result of f\_EPTF\_DsRestAPI\_DSServer\_processRequestf\_EPTF\_DsRestAPI\_DSServer\_processJSONRequest returned as a JSON encoded octetstring.

The JSON encoded objects are represented by octetstrings. Such octetstrings can be obtained for example from charstrings by using the char2oct function.

### Collecting timelines

The DsRestApi Timeline subfeature can be used to collect a timeline of dataelements. DsRestAPI automatically initializes this component.

The feature can also be used standalone, by calling the f\_EPTF\_DsRestAPI\_Timeline\_init\_CT method.

The method f\_EPTF\_DsRestAPI\_Timeline\_start has to be called to start the timeline collection.

The component is a DataSource Client. Timeline data and other dataelements can be accessed using the DataSource. See [5] and [7] for more details.

Timeline collection of an element is started when it is queried for the first time. Requests can also be appended to the timeline manually using the appropriate dataelements. These requests will be issued when the f\_EPTF\_DsRestAPI\_Timeline\_start function is called.

# References

1. ETSI ES 201 873-1 v3.2.1 (2007-02)  
   The Testing and Test Control Notation version 3. [Part 1: Core Language](http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.02.01_60/es_20187301v030201p.pdf)
2. 1/198 17-CRL 113 200/5 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 155 17-CNL 113 512 Uen  
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
4. 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. 32/155 16-CNL 113 512 Uen  
   EPTF CLL DataSource, Function Description
6. 30/159 41-FCPCA 101 181 Uen  
   Improving TitanSim’s internal communication and user interfaces
7. [CLL API Doc](http://ttcn.ericsson.se/TitanSim/CLL_APIdoc/)