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

Contents

[1 Introduction 2](#_Toc485031338)

[1.1 Revision History 2](#_Toc485031339)

[1.2 About this Document 2](#_Toc485031340)

[1.2.1 How to Read this Document 2](#_Toc485031341)

[1.2.2 Abbreviations 2](#_Toc485031342)

[1.2.3 Terminology 3](#_Toc485031343)

[1.3 System Requirements 3](#_Toc485031344)

[2 DsRestAPI 3](#_Toc485031345)

[2.1 Overview 3](#_Toc485031346)

[2.2 Description of Files in this Feature 3](#_Toc485031347)

[2.3 Description of Required Files from Other CLL Features 4](#_Toc485031348)

[2.4 Description of Required External Features outside CLL 4](#_Toc485031349)

[2.5 Installation 5](#_Toc485031350)

[2.6 Configuration 5](#_Toc485031351)

[2.7 General usage 5](#_Toc485031352)

[3 Requests and responses 6](#_Toc485031353)

[3.1 Trees and branches 7](#_Toc485031354)

[3.2 Expanding requests 7](#_Toc485031355)

[3.3 Filtering requests 8](#_Toc485031356)

[3.3.1 Range filter 8](#_Toc485031357)

[3.3.2 Filter 8](#_Toc485031358)

[3.3.3 Evaluating filters and selections 8](#_Toc485031359)

[3.4 Connection between request and response 9](#_Toc485031360)

[4 Using the APIs 9](#_Toc485031361)

[4.1 JavaScript 9](#_Toc485031362)

[4.2 Python 9](#_Toc485031363)

[4.3 TTCN 10](#_Toc485031364)

[5 Warning Messages 10](#_Toc485031365)

[6 Examples 11](#_Toc485031366)

[6.1 Request response pair 11](#_Toc485031367)

[6.2 A setData example 12](#_Toc485031368)

[6.3 A more complex example 13](#_Toc485031369)

[6.4 Filtering a single element with value 14](#_Toc485031370)

[6.5 Filtering a single element with request 15](#_Toc485031371)

[6.6 Filtering a list 16](#_Toc485031372)

[7 References 17](#_Toc485031373)

# Introduction

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Rev** | **Characteristics** | **Prepared** |
| 2016-09-16 | PA1 | Renaming UIModel to DsRestAPI | EIMRENA |
| 2016-09-16 | PA2 | Corrected typos. | EIMRENA |
| 2016-09-16 | PA3 | Corrected reference. | EIMRENA |
| 2016-10-13 | PA4 | Updated with Timeline sub-feature | EIMRENA |
| 2016-12-02 | A | Updated for release | EIMRENA |
| 2017-05-18 | PB1 | Updated Python API | EIMRENA |
| 2017-05-18 | PB2 | Editorial changes | EIMRENA |
| 2017-05-18 | PB3 | Editorial changes | EIMRENA |
| 2017-06-09 | B | Updated for release | EIMRENA |

## About this Document

### How to Read this Document

This is the User Guide for the DsRestAPI feature of the Ericsson Performance Test Framework (EPTF), Core Library (CLL). CLL is developed for the TTCN-3 [1] Toolset with TITAN [2]. This document should be read together with the Function Description of the DsRestAPI feature [5]. For more information consult the Product Revision Information, the User Guide [4] and the Function Specification [3] of CLL.

### Abbreviations

CLL Core Library

EPTF Ericsson Performance Test Framework

TitanSim New synonym for the EPTF Framework

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

GUI Graphical user interface

UI User interface

API Application programming interface

JSON JavaScript Object Notation

REST Representational State Transfer

AJAX Asynchronous JavaScript and XML

### Terminology

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

## System Requirements

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

# DsRestAPI

## Overview

The DsRestAPI feature is developed to provide APIs for the CLL Data Source. It also provides a sub-feature that collects timelines of data elements.

For more information about the Data Source and its purpose, please see [8] and [9].

## Description of Files in this Feature

The CLL DsRestAPI feature includes the following files:

* EPTF\_CLL\_DsRestAPI\_Definitions.ttcn: this TTCN-3 module contains common type definitions used by the DsRestAPI Component.
* EPTF\_CLL\_DsRestAPI\_Functions.ttcn: this TTCN-3 module contains the implementation of DsRestAPI functions.
* EPTF\_CLL\_DsRestAPI\_ExternalFunctions.cc: this module contains external functions for DsRestAPI.
* EPTF\_CLL\_DsRestAPI\_DSServer\_Definitions.ttcn: this module contains the definitions for the TTCN API of the Data Source.
* EPTF\_CLL\_DsRestAPI\_DSServer\_Functions.ttcn: this module contains the implementation of the TTCN API function.
* EPTF\_CLL\_DsRestAPI\_Timeline\_Definitions.ttcn: this module contains the definitions for the Timeline sub-feature.
* EPTF\_CLL\_DsRestAPI\_Timeline\_Functions.ttcn: this module contains the implementation of the Timeline sub-feature.
* EPTF\_CLL\_DsRestAPI\_Timeline\_ExternalFunctions.cc: this module contains external functions for the Timeline sub-feature.
* EPTF\_CLL\_DsRestAPI\_Filter\_Definitions.ttcn: this module contains the definitions for the filtering feature.
* EPTF\_CLL\_DsRestAPI\_Filter\_Functions.ttcn: this module contains the implementation of the filtering functions.
* EPTF\_CLL\_DsRestAPI\_Storage\_Definitions.ttcn: this module contains the definitions for the caching feature.
* EPTF\_CLL\_DsRestAPI\_Storage\_Functions.ttcn: this module contains the implementation of the caching functions.
* EPTF\_CLL\_DsRestAPI\_HTTPServer\_Definitions.ttcn: this module contains the common type definitions used by the DsRestAPI HTTPServer Component.
* EPTF\_CLL\_DsRestAPI\_HTTPServer\_Functions.ttcn: this module contains the implementation of the DsRestAPI HTTP server.
* DsRestAPIComm.js: the JavaScript module that contains the AJAX calls that can be used to communicate with the HTTP server.
* DsRestAPI.js: the JavaScript module that contains the DsRestAPI API functions described in [5].
* DsRestAPI.py: the Python module that contains the DsRestAPI API functions described in [5].

## Description of Required Files from Other CLL Features

The DsRestAPI feature depends on the following CLL features:

* Base
* Common
* FreeBusyQueue
* DataSource
* HashMap
* Logging
* Variable

## Description of Required External Features outside CLL

* TCCUsefulFunctions\_CNL113472
* IPL4asp\_CNL113531

## Installation

Since EPTF\_CLL\_DsRestAPI is used as a part of the TTCN-3 test environment this requires TTCN-3 Test Executor to be installed before any operation of these functions. For more details on the installation of TTCN-3 Test Executor, see the relevant section of [2].

If not noted otherwise in the respective sections, the following are needed to use EPTF\_CLL\_DsRestAPI:

* Copy the files which belong to the CLL features listed in section 2.2 and 2.3 to the directory of the test suite or create symbolic links to them.
* Import the DsRestAPI test or write your own application using EPTF DsRestAPI.
* Create Makefile or modify the existing one. For more details, see the relevant section of [2].
* Edit the config file according to your needs, see section 2.6.

## Configuration

The executable test program behaviour is determined through the run-time configuration file. This is a simple text file which contains various sections. The usual suffix of configuration files is .cfg. For further information on the configuration file, see [2].

The DsRestAPI feature does not define any module parameters.

## General usage

DsRestAPI provides three main ways to communicate with the Data Source. These three ways are: the TTCN API, the JavaScript API and the Python API.

DsRestAPI provides all three methods of communication, while the standalone DSServer only provides the TTCN API.

To use the DsRestAPI feature, perform the following steps:

1. Extend your component with the EPTF\_DsRestAPI\_CT component
2. Call the init function (f\_EPTF\_DsRestAPI\_init\_CT) of the DsRestAPI to initialize the feature
3. Call the f\_EPTF\_DsRestAPI\_start function to start the DSServer and HTTP server
4. (Optional) Call f\_EPTF\_DsRestAPI\_Timeline\_start to start collecting timelines. For more information, see [5]
5. We can now use the three APIs to issue requests:
   1. In TTCN environment use the f\_EPTF\_DsRestAPI\_DSHandler\_processJSONRequest function to process a request
   2. Use the functions of the JavaScript API to send HTTP requests to the HTTP server
   3. Use the Python API to send HTTP requests to the HTTP server
6. Call f\_EPTF\_Base\_cleanup\_CT function before your component terminates

To use the standalone DSServer without starting the HTTP server, fewer steps are required:

1. Extend your component with the EPTF\_DsRestAPI\_DSServer\_CT component
2. Call the init function (f\_EPTF\_DsRestAPI\_DSServer\_init\_CT) of the DSServer to initialize the feature
3. Use the f\_EPTF\_DsRestAPI\_DSHandler\_processJSONRequest function to process a request
4. Call f\_EPTF\_Base\_cleanup\_CT function before your component terminates

To use the Timeline as a standalone feature, do the following:

1. Extend your component with the EPTF\_DsRestAPI\_Timeline\_CT component
2. Call the init function (f\_EPTF\_DsRestAPI\_Timeline\_init\_CT) of the Timeline to initialize the feature
3. Use the data elements of the DsRestAPI Timeline DataSource Client to collect and access timelines.
4. Call f\_EPTF\_Base\_cleanup\_CT function before your component terminates

See [5] for more details about the API functions.

# Requests and responses

To communicate with the Data Source, JSON requests need to be sent to the HTTP server. For more detailed information on the format of the JSON and its encoding, please see the appropriate parts of [5].

## Trees and branches

Using the *children* member of the getData request, we can create a tree of nested requests.

**Definition**: The *request tree* is a graph, the nodes of the tree are the requests themselves, and an arc connects request A with request B if B is contained in the children list of A.

This is useful when we want to use the result of one request as a parameter of another one.

**Definition**: We call the requests located on a path from the root to some other request a *branch* of the request tree.

To refer to the result of the first request (the root) in a branch of the tree in a parameter, use %Parent0% in the paramValue of the parameter.

Similarly, if we want to refer to the result of the 28th request on the branch, use %Parent27% in the paramValue.

It is also possible to refer to more than one parent in the paramValue. In some cases we might want to have “%Parent0%.%Parent1%” as the paramValue.

When the parent is a list, we can also refer to the current index using %Parent0::idx%. **Important**: this will only work if the request does not have a filter or a rangeFilter.

A setData request cannot have children. However, they can be nested in a tree of other getData requests.

See section 6 for examples, and consult the Data Source help and [8] for the parameters and the possible nesting of the requests.

## Expanding requests

Suppose we have a request A whose response is a list. What happens to the requests of the children of A?

The answer is, they are processed as many times as the number of elements in the list.

Usually, this is not the intended behaviour. We can fine tune which requests get processed by using the *selection* member of the request A.

The *selection* member of a getData request is a list of integers.

When the response to the request is a list and the request has child requests, then the child requests will only be processed on those elements in the list that are specified in the *selection*.

If the selection is an empty list, the child requests will not be processed on any of the elements in the list.

If the selection element is omitted, then the child requests are processed on all elements.

In a setData request, if the element is a list and we want to change the value of the third element in the list, we would have “indxsInList”: [2] as a member of the setData request.

For an example, please see section 6.

## Filtering requests

Filters can be used to filter requests and responses based on specific criteria.

### Range filter

The *rangeFilter* member of a getData request can be used to query only a continuous part of a list.

The offset member of the rangeFilter is the index of the first element in the original list, and the count is the maximum number of elements we wish to query.

### Filter

The *filter* member of a getData request can be used to filter a list using conditions.

The filter is a rooted tree. If the root evaluates to false on an element, it is filtered.

If a single element response is filtered, the response will change to an empty string whose tp is 0 (see [5] for more details about response types).

If the response is a list, and the filter does not contain any reference to the list elements, then it is treated as a single element response in terms of filtering (the result will be either an empty string with tp = 0, or the whole list).

If the response is a list and the filter contains references to its elements, then the filter will be evaluated for every element of the original list. The elements, for which the filter evaluates to false, will be removed from the result.

See section 6 for some examples.

### Evaluating filters and selections

Suppose we are processing a request whose response is a list, and it contains a filter with reference to the elements, a rangeFilter and selection.

First, the rangeFilter is applied. Then we run the filter on all elements. And finally, the selection is applied.

This is why using %Parent0::idx% is not advised, since filtering cannot be done in this case.

## Connection between request and response

The response has similar structure to the request.

The response is also a tree and is built according to the rules described previously.

Imagine the response as a graph which is a tree. We can obtain this tree structure from the request tree if we make a copy of certain subtrees. Namely take the node closest to the root whose response is a list. Copy the subtree that belongs to this node as many times as the length of the response list and connect the copies to the parent of the node. Repeat this for all nodes whose corresponding request returns a list.

# Using the APIs

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

## JavaScript

Please see the description of the f\_EPTF\_DsRestAPI\_start function in [5] to correctly configure the HTTP server before using the JavaScript API.

The DsRestAPIComm.js and DsRestAPI.js files have to be included into a web application and a new instance of DsRestAPI has to be created.

Afterwards, the functions described in the JavaScript specific part of [5] can be used.

## Python

The DsRestAPI.py can be used in two ways.

We can use it as a script with the following command line arguments:

* *-?* print local help (for usage, examples, and types)
* *-h source element* print Data Source help for the given source and element (element or both are optional)
* *-u http://address:port*: the address of the HTTP server
* *-f file.json*: send the request found in the JSON file
* *-j ‘json string’*: send the JSON string as the request

To issue specific getData requests, use the following parameters:

* *-source SomeSource*
* *-element SomeElement*
* *-params ParamName1 ParamValue1 ParamName2 ParamValue2*
* *-ptcname SomePTCName*

In addition to these, a setData request must contain the following parameters:

* *-content TheContent*
* *-tp SomeInt* (see [5] for more details)

For further explanations about the arguments, please see the Python specific part of [5].

We can also import the Python script as a module and use the API class similarly to the JavaScript API.

## TTCN

The full description of the API function can be found in [5].

# Warning Messages

**Note:** besides the below described warning messages, warning and error messages shown in [2] or those of other used features or product may also appear.

“DsRestAPI cannot listen on port: port\_number”

This warning indicates that the port assigned to DsRestAPI is already in use.

“HTTPServer\_directory does not exist: dir\_name”

This warning occurs if the given directory for the HTTP server does not exist and it is not the empty string.

“CustomizableApp directory does not exist: dir\_name”

This warning occurs if the given directory for the customizable app symlink does not exist and it is not the empty string.

“DsRestAPI API directory does not exist: dir\_name”

This warning occurs if the given directory for the JavaScript API symlink does not exists and it is not the empty string.

"Symlink creation was unsuccessful for …”

This warning means that a symlink could not be created.

“… directory does not exist and could not be created it in …”

This warning means that a directory creation failed.

“Symlink cannot be removed: symlink\_name”

This warning appears if a symlink cannot be removed during cleanup.

"Failed to get absolute path of …"

This warning means that a relative path could not be resolved.

# Examples

## Request response pair

[

{

"list": [

{

"node": {

"val": "DefaultEGrp",

"type": 9,

"childVals": [

{

"list": [

{

"node": {

"val": "DefaultSc",

"type": 9,

"childVals": [

{

"node": {

"val": "[led:blue]Idle",

"type": 10

}

}

]

}

},

{

"node": {

"val": "DefaultWSc",

"type": 9

}

},

{

"node": {

"val": "SC\_PH\_0",

"type": 9

}

}

]

}

]

}

},

{

"node": {

"val": "DefaultEGrp2",

"type": 9

}

}

]

}

]

[

{

"getData": {

"source": "ExecCtrl",

"element": "EntityGroups",

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "Scenarios",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

}

],

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "ScStatus",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

},

{

"paramName": "Scenario",

"paramValue": "%Parent1%"

}

]

}

}

],

"selection": [

0

]

}

}

],

"selection": [

0

]

}

}

]

As we can see, the EntityGroups request has a child request that uses its value as a parameter.

In addition, the subrequest is only processed for the first entity group because of the *selection* member. This can be observed in the response, where there is a scenario list below DefaultEGrp, but there is no list below DefaultEGrp2.

## A setData example

{

"setData": {

"source": "ExecCtrl",

"element": "TcTargetCPSOrWeight",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "DefaultEGrp"

},

{

"paramName": "Scenario",

"paramValue": "DefaultSc"

},

{

"paramName": "TrafficCase",

"paramValue": "DefaultTC1"

}

],

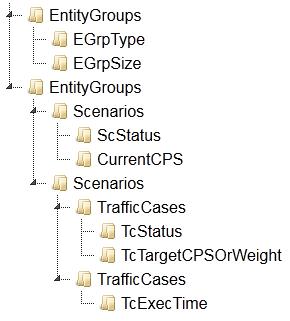
"content": "20",

"type": 1

}

}

## A more complex example



This is a large request. Only the element names are displayed. In a real world scenario, we might want to control the selection of some requests.

For example, imagine that the first entity groups request does not have a selection, so the response will contain the type and size of every entity group.

The second entity groups request can have a selection which we may control from a user interface, so only the scenarios of the selected entity group will be contained in the response.

## Filtering a single element with value

{

"getData": {

"source": "ExecCtrl",

"element": "EntityGroups",

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "Scenarios",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

}

],

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "IsWeightedSc",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

},

{

"paramName": "Scenario",

"paramValue": "%Parent1%"

}

],

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "ScTargetCPS",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

},

{

"paramName": "Scenario",

"paramValue": "%Parent1%"

}

],

"filter": {

"dataValue": "%Parent2%"

}

}

}

]

}

}

]

}

}

]

}

}

The filter will ensure that the ScTargetCPS request is only evaluated on weighted scenarios. The response will contain all scenarios, and the target cps of weighted ones, and a filtered element response (empty string and tp=0) for the other target cps requests.

## Filtering a single element with request

{

"getData": {

"source": "ExecCtrl",

"element": "EntityGroups",

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "Scenarios",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

}

],

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "ScTargetCPS",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

},

{

"paramName": "Scenario",

"paramValue": "%Parent1%"

}

],

"filter": {

"request": {

"source": "ExecCtrl",

"element": "IsWeightedSc",

"params": [

{

"paramName": "EntityGroup",

"paramValue": {

"dataValue": "%Parent0%"

}

},

{

"paramName": "Scenario",

"paramValue": {

"dataValue": "%Parent1%"

}

}

]

}

}

}

}

]

}

}

]

}

}

The filter is now a request with parameters. The result will be the same as in the previous example.

## Filtering a list

{

"getData": {

"source": "ExecCtrl",

"element": "EntityGroups",

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "Scenarios",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

}

],

"children": [

{

"getData": {

"source": "ExecCtrl",

"element": "ScTargetCPS",

"params": [

{

"paramName": "EntityGroup",

"paramValue": "%Parent0%"

},

{

"paramName": "Scenario",

"paramValue": "%Parent1%"

}

]

}

}

],

"filter": {

"request": {

"source": "ExecCtrl",

"element": "IsWeightedSc",

"params": [

{

"paramName": "EntityGroup",

"paramValue": {

"dataValue": "%Parent0%"

}

},

{

"paramName": "Scenario",

"paramValue": {

"dataValue": "%Parent1%"

}

}

]

}

}

}

}

]

}

}

The filter is now part of the scenarios request and it contains a reference to the list elements: %Parent1%. This means that the response scenario list will only contain weighted scenarios.

# 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 113 200/6 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 155 17-CNL 113 512 Uen   
   EPTF Core Library for TTCN-3 toolset with TITAN, Function Specification
4. 198 17-CNL 113 512 Uen  
   EPTF Core Library for TTCN-3 toolset with TITAN, User Guide
5. 39/155 16-CNL 113 512 Uen  
   EPTF Core Library DsRestAPI, Function Description
6. EPTF CLL for TTCN-3 toolset with TITAN, [Reference Guide](http://ttcn.ericsson.se/TCC_Releases/Libraries/EPTF_Core_Library_CNL113512/doc/apidoc/html/index.html)
7. 30/159 41-FCPCA 101 181 Uen  
   Improving TitanSim’s internal communication and user interfaces
8. 32/155 16-CNL 113 512 Uen  
   EPTF CLL DataSource, Function Description
9. 32/198 17-CNL 113 512 Uen  
   EPTF CLL DataSource, User Guide