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

Guideline to create custom GUI

Contents

[1 Introduction 3](#_Toc327538870)

[1.1 Revision history 3](#_Toc327538871)

[1.2 References 3](#_Toc327538872)

[1.3 Presumed Knowledge 3](#_Toc327538873)

[1.4 Scope 3](#_Toc327538874)

[1.5 Abbreviations 3](#_Toc327538875)

[2 Premises 4](#_Toc327538876)

[3 Design your GUI 4](#_Toc327538877)

[3.1 New WoW 4](#_Toc327538878)

[3.2 External data elements 4](#_Toc327538879)

[3.2.1 Attributes of the data elements 4](#_Toc327538880)

[3.2.2 Parameters 5](#_Toc327538881)

[3.2.3 Example 6](#_Toc327538882)

[3.3 Iterators 6](#_Toc327538883)

[3.3.1 Iterator attributes 7](#_Toc327538884)

[3.3.2 Iterator parameters 7](#_Toc327538885)

[3.3.3 Example 7](#_Toc327538886)

[3.4 External values 8](#_Toc327538887)

[3.4.1 Example 8](#_Toc327538888)

[3.5 Conditional elements 8](#_Toc327538889)

[3.5.1 Condition 9](#_Toc327538890)

[3.5.2 Insertif 9](#_Toc327538891)

[3.5.3 Example 10](#_Toc327538892)

[4 Integrate into the code 12](#_Toc327538893)

[5 Under the hood 12](#_Toc327538894)

[5.1 Collaboration 12](#_Toc327538895)

[5.2 The returned data 13](#_Toc327538896)

[5.2.1 Iterators and data elements 13](#_Toc327538897)

[5.2.2 Conditions 14](#_Toc327538898)

[6 Publish your data 14](#_Toc327538899)

[6.1.1 Initialization 14](#_Toc327538900)

[6.1.2 Responding the data requests 15](#_Toc327538901)

[6.1.3 Responding the condition requests 15](#_Toc327538902)

[7 Available data source elements 16](#_Toc327538903)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 2011-09-01 | PA1 | First draft version | ELSZSKU |
| 2012-04-13 | PB1 | Externalvalue update | EKOVIST |
|  |  |  |  |

## 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. 18/198 17-CNL 113 512 Uen  
   EPTF CLL Variable, User Guide
3. 155 17-CNL 113 663 Uen  
   XTDP Protocol Module Function Specification
4. 198 17-CNL 113 663 Uen  
   XTDP Protocol Module User Guide

## Presumed Knowledge

In order to properly understand the documentation you shall be familiar with the TTCN-3 language ‎[1], the use of the EPTF Variable feature ‎[2], and the XTDP protocol module ‎[3],‎[4].

## Scope

This document is to give some useful suggestions and information to the users of the Core Load Library to make the customized GUI creation easier.

## Abbreviations

WoW Way of Working

CT TTCN-3 Component Type

CLL Core Load Library

EPTF Ericsson Performance Test Framework, formerly TITAN Load Test Framework

CPS Call per Second

PTC Parallel Test Component

# Premises

In the former WoW the component types published their data directly on the GUI. The layout of the GUI was built into the code; therefore each customization request needed the change of the code.

Of course no one can design a GUI which is suitable for all use cases. The complaints and change requests have been mushroomed, and the customization became more and more complicated. Therefore a new, customizable GUI description API had to be implemented.

# Design your GUI

## New WoW

In the new WoW the component types publish their data, which can be used to build up a GUI. You can pick the data you are interested in, and put them on the GUI as you wish. You do it using an XML file, which describes both the layout and the data content together.

The XML is described by the XTDL.xsd file, which is the part of the XTDP protocol module. It contains the same widgets as its ancestor, the XUL, which was the part of the XTDPAsp testport.

But as long as in the previous WoW the GUI and the data behind it was absolutely separated (the XUL described the layout, the data was added from code), in the new WoW the GUI description contains both the layout and the data behind it. Descriptors of widgets – which can be joined to some kind of data – have been extended with an "externaldata" element, which describe the data to be joined.

## External data elements

The component types publish their data in form of named data elements. Hereafter the component type publishing data elements are called to "data sources".

The current CPS rate of a scenario is published e.g. by the data source named ExecCtrl in a data element which is named to "CurrentCPS", the state of a traffic case in a data element named to "TcStatus", the data element behind the start button of a traffic case is called to "TcStartButton".

### Attributes of the data elements

The data elements have an "element" attribute, which identifies the data element by its name.

The "source" attribute identifies the data source which publishes the given data element. Usually one component type publishes its data on one name, usually on its feature name, but it's possible to publish the data of the same component type under different names, and the "source" names can be different than the feature name. E.g. the LoggingServer CT publishes its data under the name of "Logging".

Usually more than one CT distributed on different PTCs can extend the same CT, e.g. all load generator PTCs extend the LoggingClient CT. Therefore there can be more than one PTC in the configuration which publishes its data under the same name. The "ptcname" attribute identifies the PTC from which the required data has been published. This name is the pl\_selfName parameter of the first call of the f\_EPTF\_Base\_init\_CT function.

If there is only one instance from the given CT, the "ptcname" attribute is optional.

So a sample data element description in the XML looks something like this:

<externaldata element=*'CurrentCPS'* source=*'ExecCtrl'* ptcname=*'SimpleExecCtrl'*>

This means that the widget which contains this element will be joined to the data element called "CurrentCPS" of the ExecCtrl source which is placed on the PTC having the name "SimpleExecCtrl".

### Parameters

There are some data elements, which have a real meaning "as they are", e.g. the "Exit" data element means the element behind the "Exit" button. But most of the data elements need further specification. E.g. the "TcStatus" data element mentioned above needs the identification of the traffic case. These data elements have parameters, one or more, depending on the data element. E.g. the "TcStatus" has three parameters: entity group name, scenario name, and traffic case name.

The parameters have a name to identify them, and a value. The data description of the CPS of a scenario in the XML looks something like this:

<externaldata element=*'CurrentCPS'* source=*'ExecCtrl'* ptcname=*'SimpleExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'DefaultEGrp'*/>

<dataparam name=*'Scenario'* value=*'DefaultWSc'*/>

</params>

</externaldata>

The order of the parameters is irrelevant.

There can be data elements, which have different parameters in different combination, and the meaning of the data element depends on the combination of the parameters. E.g. the only data element of the LoggingServer CT accepts the following parameters: "ComponentType", "Class". If there is no parameter specified, the data element means the global log enabled state. If only the "ComponentType" parameter is set, the data element means component type log state. If both the "ComponentType" and "Class" parameters are set, the data element means the log enabled state of the given log class of the given component type. Any other combination is invalid.

### Example

This sample shows how to display a data element (current CPS of the DefaultEGrp.DefaultWSc weighted scenario) in a treecell widget:

<Widgets xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl'>

<window>

<tree id='SCList'>

<treecols>

<treecol label='Name' widgetType='string' />

<treecol label='CPS' widgetType='floatField' />

</treecols>

<treechildren>

<treeitem>

<treerow>

<treecell label='DefaultEGrp.DefaultEGrp' />

<treecell>

<externaldata element='CurrentCPS' source='ExecCtrl'>

<params>

<dataparam name='EntityGroup' value='DefaultEGrp' />

<dataparam name='Scenario' value='DefaultWSc' />

</params>

</externaldata>

</treecell>

</treerow>

</treeitem>

</treechildren>

</tree>

</window>

</Widgets>

The resulted GUI:

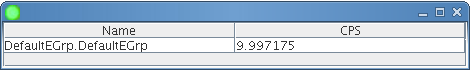


Figure 1 GUI with a joined data element

## Iterators

Of course it's difficult to describe all scenarios, traffic cases, etc. of the configuration, and modify the GUI descriptor XML each time when the configuration is modified. To make it possible to write a more flexible GUI description, the data sources provide list elements to go through the main configurable elements. These elements are called to "iterator".

If you place an iterator into the GUI descriptor XML, during the process of the XML its embedded content will be multiplied as many times as many elements the iterator contains. E.g. if there are two entity groups in the configuration, and the "EntityGroups" iterator contains a label widget, during the build up of the GUI there will be two label widgets instead of the iterator.

### Iterator attributes

Iterators have the same attributes as the data elements, and these attributes have the same meaning. However, iterators have an additional attribute: "id". You can write the content of this mandatory string attribute into the value of any string attribute in the content of the iterator between two % signs (see ‎3.3.3 later), and it will be replaced with the actual value of the iterator.

### Iterator parameters

Of course there can be iterators which require parameters, just like the data elements. The use and the behavior of the parameters of the iterators are the same as in case of data elements.

### Example

This sample shows how to use iterators to list the scenarios existing in the configuration:

<Widgets xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl'>

<window>

<tree id='SCList'>

<treecols>

<treecol label='Name' widgetType='string' />

</treecols>

<treechildren>

<treeitem>

<iterator element='EntityGroups' id='EGrp' source='ExecCtrl'>

<iterator element='Scenarios' id='SC' source='ExecCtrl'>

<params>

<dataparam name='EntityGroup' value='%EGrp%' />

</params>

<treerow>

<treecell label='%EGrp%.%SC%' />

</treerow>

</iterator>

</iterator>

</treeitem>

</treechildren>

</tree>

</window>

</Widgets>

The highlighted parts show how to refer to the iterators in the other elements.

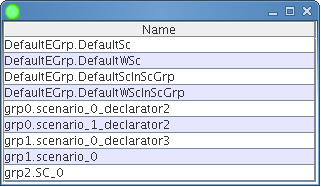


Figure 2 GUI which lists the elements of an iterator

## External values

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 returnes the given variable value in charstring format. The %ID::ref% returnes the variable name. Used in all places, where the iterators as well.

### Example

This sample shows how to use externalvalues to read existing variable contents in the configuration:

<Widgets xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl'>

<window>

<tree id='SCList'>

<treecols>

<treecol label='Name' widgetType='string' />

</treecols>

<treechildren>

<treeitem>

<externalvalue id='titleValue' source='source' element='title'>

<treecell label='%titleValue%' />

</ externalvalue >

</treeitem>

</treechildren>

</tree>

</window>

</Widgets>

The highlighted part show how to refer to the externalvalue in the other elements.

## Conditional elements

There are elements which are valid only in specified cases. E.g. the CPS of a scenario – which is mentioned in ‎3.2.3 – is valid only in case of weighted scenarios.

### Condition

The GUI descriptions can contain "condition" elements. They define a condition which can be used in their content.

The "condition" element has the same attributes than the iterators, and they can have parameters too, just like the data elements and the iterators.

### Insertif

To use the conditions, in the GUI description there is an "insertif" element. As its name tells it, the content of the element will be added to the GUI if the referred condition is true. The "insertif" element has an "id" attribute, which refers to the "id" attribute of the referred condition element.

The conditions of the insertif element can be negated. It has a "negate" optional boolean attribute. If it's set to true, the negated value of the referred condition will be used during the process.

### Example

The following sample describes how to use conditional elements to display only the weighted scenarios with their CPS:

<Widgets xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl'>

<window>

<tree id=*'SCList'*>

<treecols>

<treecol label=*'Name'* widgetType=*'string'* />

<treecol label='CPS' widgetType='floatField' />

</treecols>

<treechildren>

<treeitem>

<iterator element=*'EntityGroups'* id=*'EGrp'* source=*'ExecCtrl'*>

<iterator element=*'Scenarios'* id=*'SC'* source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'*></dataparam>

</params>

<condition

element=*'IsWeightedSc'*

id=*'%EGrp%.%Scenario%.weighted'*

source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'*></dataparam>

<dataparam name=*'Scenario'* value=*'%SC%'*></dataparam>

</params>

<insertif id=*'%EGrp%.%Scenario%.weighted'*>

<treerow>

<treecell label=*'%EGrp%.%SC%'* />

<treecell>

<externaldata element=*'CurrentCPS'* source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'* />

<dataparam name=*'Scenario'* value=*'%SC%'* />

</params>

</externaldata>

</treecell>

</treerow>

</insertif>

</condition>

</iterator>

</iterator>

</treeitem>

</treechildren>

</tree>

</window>

</Widgets>

As we saw on Figure 2, there are 9 scenarios in the configuration. But using the insertif the resulting GUI contains only the weighted scenarios, as it can be seen on Figure 3:

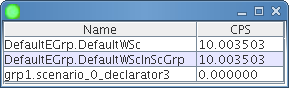


Figure 3 Filtered list of scenarios

You can use the insertif and its negated version to display different content if the scenario is weighted or not:

<Widgets xmlns='http://ttcn.ericsson.se/protocolModules/xtdp/xtdl'>

<window>

<tree id=*'SCList'*>

<treecols>

<treecol label=*'Name'* widgetType=*'string'* />

<treecol label='CPS' widgetType='floatField' />

</treecols>

<treechildren>

<treeitem>

<iterator element=*'EntityGroups'* id=*'EGrp'* source=*'ExecCtrl'*>

<iterator element=*'Scenarios'* id=*'SC'* source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'*></dataparam>

</params>

<condition

element=*'IsWeightedSc'*

id=*'%EGrp%.%Scenario%.weighted'*

source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'*></dataparam>

<dataparam name=*'Scenario'* value=*'%SC%'*></dataparam>

</params>

<treerow>

<treecell label=*'%EGrp%.%SC%'* />

<insertif id=*'%EGrp%.%Scenario%.weighted'*>

<treecell>

<externaldata element=*'CurrentCPS'* source=*'ExecCtrl'*>

<params>

<dataparam name=*'EntityGroup'* value=*'%EGrp%'* />

<dataparam name=*'Scenario'* value=*'%SC%'* />

</params>

</externaldata>

</treecell>

</insertif>

<insertif id=*'%EGrp%.%Scenario%.weighted'* negate=*'true'*>

<treecell />

</insertif>

</treerow>

</condition>

</iterator>

</iterator>

</treeitem>

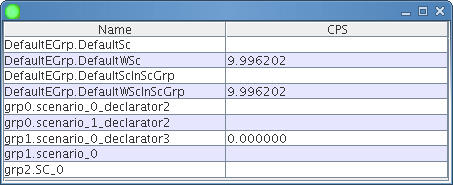
</treechildren>

</tree>

</window>

</Widgets>

This sample XML lists all scenarios not taking care if they were weighted or not, but the cells in the second column are linked to data elements only if the scenario was weighted. The resulted GUI is displayed on



# Integrate into the code

In the UIHandler feature there are two functions to create the custom GUI:

**public** **function** f\_EPTF\_UIHandler\_createGUI (

**in** **charstring** pl\_xul,

**in** **charstring** pl\_parentWidgetId := "" )

**public** **function** f\_EPTF\_UIHandler\_createGUIFromFile(

**in** **charstring** pl\_fileName,

**in** **charstring** pl\_parentWidgetId := "")

Both functions do the same, but the f\_EPTF\_UIHandler\_createGUIFromFile function reads the layout from a file, and f\_EPTF\_UIHandler\_createGUI gets it from a charstring parameter.

The GUI description tree must start with the "widgets" element. It can contain "window" element, or any other element from the "widgets" group.

The UIHandler processes the GUI description, builds up the GUI layout, and joins the data elements to the appropriate widgets. This time the configuration must be ready. If a component starts later, and its referred data source elements not published yet, the process results error. Or, if some configuration elements are not ready at the time of building the GUI, the iterators won't be updated later, and the GUI remains partially configured.

You can create a whole window, or its parts. If the first child element of the root "widgets" element is a window, a clearGUI call will be performed first.

You can call these functions several times and build up the GUI from parts if the first child was not a window.

# Under the hood

## Collaboration

Each component type which publishes its data must extend the EPTF\_DataSourceClient\_CT component type. In the init function it joins to the specified EPTF\_DataSource\_CT, which acts as a server. The data publisher components register their request handler functions.

Data Source Server

DS Client: LGenBase 1

Register

DS Client: ExecCtrl

Register

Register

DS Client: LGenBase n

Figure 4 DataSource client-server architecture

The users of the published data source elements (data elements, iterators, etc.) can ask them from the EPTF\_DataSource\_CT. The EPTF\_DataSource\_CT processes the request, identifies the appropriate client, and forwards the request to it. Using this method the data requesting is independent from the configuration, e.g. from the number of load generators.

## The returned data

### Iterators and data elements

In the Core Load Library the general way of handling data is the use of the Variable feature. Therefore it's evident that the values behind the published data are represented by EPTF variables.

Each publisher component type registers the data request handler function:

**type** **function** fcb\_EPTF\_DataSourceClient\_dataHandler(

**out** **charstring** pl\_dataVarName,

**in** **charstring** pl\_source,

**in** **charstring** pl\_ptcName,

**in** **charstring** pl\_element,

**in** EPTF\_DataSource\_Params pl\_params

) **runs** **on** **self** **return** **integer**;

When a request arrives asking a data element or an iterator, the function processes the relevant incoming parameters, and puts the name of the EPTF variable representing the requested data into the pl\_dataVarName out parameter. The return value of the function must be 0 if the request was successful, otherwise it must be a nonzero value.

In case of iterators, the returned EPTF variable always must have charstringlistVal direct content. In case of data elements it can be any valid EPTF variable direct content type.

### Conditions

There is another callback function for the conditions.

**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**;

Since a condition can be true or false, its output is a boolean value. The return value is the same as in case of the data request handler function.

You have to register as many condition request handler functions, as many conditions your CT provides.

# Publish your data

The following chapters demonstrate how easy you can publish your data using the data source. They contain code parts from the source code of the ExecCtrl feature.

### Initialization

The init function of the CT has an EPTF\_DataSource\_CT parameter. This pl\_dataSource\_compRef parameter has a default null value to make possible the use of the CT without DataSource server.

**public** **function** f\_EPTF\_ExecCtrl\_init\_CT(

**in** **charstring** pl\_selfName,

...

**in** EPTF\_DataSource\_CT pl\_dataSource\_compRef := null)

**runs** **on** EPTF\_ExecCtrl\_CT {

The init function initializes the DataSourceClient feature and registers the data request and condition request handler functions.

f\_EPTF\_DataSourceClient\_init\_CT(pl\_selfName, pl\_dataSource\_compRef);

**if**(pl\_dataSource\_compRef != null){

...

f\_EPTF\_DataSourceClient\_registerData(

c\_ExecCtrl\_DataSource\_sourceId, f\_EPTF\_Base\_selfName(),

refers(f\_EPTF\_ExecCtrl\_DSProcessData));

f\_EPTF\_DataSourceClient\_registerCondition(c\_ExecCtrl\_DataSource\_sourceId,

f\_EPTF\_Base\_selfName(),

refers(f\_EPTF\_ExecCtrl\_conditionHandler\_isWeightedSc),

c\_ExecCtrl\_conditionIsWeightedSc);

f\_EPTF\_DataSourceClient\_registerCondition(

c\_ExecCtrl\_DataSource\_sourceId, f\_EPTF\_Base\_selfName(),

refers(f\_EPTF\_ExecCtrl\_conditionHandler\_isInScGroup),

c\_ExecCtrl\_conditionIsInScGroup);

}

That's all you have to do during the initialization.

### Responding the data requests

The data request handler function receives all attributes and parameters which are present in the request (as it's described in ‎3.2 and ‎3.3).

**friend** **function** f\_EPTF\_ExecCtrl\_DSProcessData(

**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\_ExecCtrl\_CT **return** **integer**{

Usually the functions need only the pl\_element attribute and the associated pl\_params, because the function provides data of one PTC and one data source. But you can create a more complicated distribution if you wish.

The function fills out the pl\_dataVarName function parameter according to the pl\_element and the pl\_params with the name of the appropriate variable:

**select**(pl\_element){

**case**(c\_ExecCtrl\_iteratorEntityTypesForLGen){

...

}

**case**(c\_ExecCtrl\_iteratorRegulatedItems){

...

pl\_dataVarName := c\_ExecCtrl\_DS\_iteratorVar\_prefix &

c\_ExecCtrl\_iteratorRegulatedItems\_varName;

...

}

If there was an error, returns a nonzero value.

case else { //error, no rule for that

pl\_dataVarName := "";

f\_EPTF\_ExecCtrl\_warning(

%definitionId& ": unhandled element: "& pl\_element);

**return** -1;

}

}

...

If there request was successful, returns zero.

return 0;

### Responding the condition requests

The condition request handler functions are really similar to the data request handler functions. They receive all attributes and parameters which are present in the request too.

friend function f\_EPTF\_ExecCtrl\_conditionHandler\_isWeightedSc(

**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\_ExecCtrl\_CT **return** **integer** {

And they fill out the pl\_dataVarName function parameter according to the pl\_method and the pl\_params.

# Available data source elements

The available data source elements, such as source identifiers, data element, iterator and condition names you can find in the APIDoc, in the definition modules of the features as charstring constants.