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

EPTF CLL SMacro, Function Description

Contents

[1. Introduction 2](#_Toc405383404)

[1.1 Revision History 2](#_Toc405383405)

[1.2 How to Read this Document 2](#_Toc405383406)

[1.3 Scope 2](#_Toc405383407)

[1.4 Recommended Way of Reading 2](#_Toc405383408)

[1.5 Typographical Conventions 2](#_Toc405383409)

[1.6 Abbreviations 2](#_Toc405383410)

[1.7 Terminology 3](#_Toc405383411)

[2 General Description 3](#_Toc405383412)

[2.1 Functionality 3](#_Toc405383413)

[2.2 String Templates 4](#_Toc405383414)

[2.3 String Macros 4](#_Toc405383415)

[3 Resolving String Templates 4](#_Toc405383416)

[4 Rules for String Macros 5](#_Toc405383417)

[4.1 String Macro Syntax 5](#_Toc405383418)

[4.2 String Macro Name 5](#_Toc405383419)

[4.3 String Macro Parameters 6](#_Toc405383420)

[4.3.1 String Template as a Parameter 6](#_Toc405383421)

[4.3.2 Escape Rules for String Macro Parameters 7](#_Toc405383422)

[4.4 String Macro Value 8](#_Toc405383423)

[4.4.1 String Template as a Value 8](#_Toc405383424)

[4.4.2 Handling Undefined String Macros 8](#_Toc405383425)

[4.5 Calculating String Macro Value with Custom Functions 9](#_Toc405383426)

[4.5.1 Callback Function Type 9](#_Toc405383427)

[4.5.2 Register/Deregister Callback Function 9](#_Toc405383428)

[4.5.3 Define String Macros 10](#_Toc405383429)

[4.6 EVAL Built in Macro 11](#_Toc405383430)

[4.6.1 Supported Operators 11](#_Toc405383431)

[4.6.2 Auto-EVAL 12](#_Toc405383432)

[5 Functional Interface 12](#_Toc405383433)

[5.1 Naming Conventions 12](#_Toc405383434)

[5.2 Public Functions 12](#_Toc405383435)

[5.2.1 Initialization of the SMacro Feature 12](#_Toc405383436)

[5.2.2 Define String Macro 13](#_Toc405383437)

[5.2.3 Undefine String Macro 13](#_Toc405383438)

[5.2.4 Registering Macro Calculator Callback Function 13](#_Toc405383439)

[5.2.5 De-registering Macro Calculator Callback Function 14](#_Toc405383440)

[5.2.6 Resolve String Macro 14](#_Toc405383441)

[5.2.7 Built-in Callback Function for EVAL Macro 15](#_Toc405383442)

[5.3 Summary Table of All Public Functions for EPTF SMacro 15](#_Toc405383443)

[6 References 15](#_Toc405383444)

# Introduction

## Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Rev** | **Characteristics** | **Prepared** |
| 2014-11-14 | PA1 | First draft version | EZOLZSI |
| 2014-12-02 | A | Updated for release | ESZILSZ |

## How to Read this Document

This is the Function Description for the SMacro feature of the Ericsson Performance Test Framework (EPTF), Core Load 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 [3].

## Scope

This document is to specify the content and functionality of the SMacro 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 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 Load Test Framework, formerly TITAN Load Test Framework

MTC Main Test Component

PTC Parallel Test Component

SMacro String Macro feature

TitanSim Ericsson Load Test Framework, formerly TITAN Load Test 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. (That is, which is not protocol-, or application-dependent). The EPTF CLL is to be supplied and supported by the TCC organization. Any EPTF CLL development is to be funded centrally by Ericsson.

# General Description

This document specifies the SMacro feature of the EPTF CLL.

The SMacro feature is developed to replace the former external template handling mechanism in CLL LGenBase.

String Macros are necessary when someone needs to replace certain patterns in a text. These patterns are called String Macros and the string itself is termed as String Template.

Example:

|  |  |
| --- | --- |
| String Template | “Username is: $SIGNUM” |
| String Macro | SIGNUM |
| Value of String Macro | “ethjgi” |
| Resolved String Template | “Username is: ethjgi” |

## Functionality

The SMacro feature provides the following functionality:

* the values of the String Macros can be defined, undefined, or redefined through API functions
* replaces the String Macros in the content of the String Template with the corresponding values of the String Macros
* it is possible to set or unset a custom function to calculate the value of the String Macro
* built-in macro (EVAL) calculates mathematical expressions

## String Templates

String Templates are generic strings that may contain the so called String Macros. When a String Template is processed, all defined String Macros in it are replaced by their corresponding values.

String Templates must follow the following rules:

1. They should have the TTCN-3 type of charstring
2. They can contain any characters allowed for the charstring TTCN-3 type
3. String Template may contain any number of String Macros

Undefined String Macros remain unchanged in the processed value of the String Template.

String Templates may contain mathematical expressions that can be evaluated automatically or with the help of the built-in EVAL macro.

## String Macros

String Macros are generic patterns in the String Templates that will be substituted with their defined values.

The value of the String Macros can also be calculated with a user given custom function that is registered for the given String Macro.

The value of String Macros can be redefined during runtime.

# Resolving String Templates

String Templates are strings that may contain any number of String Macros. The String Macros can be replaced with their values in the String Template using an API function of the SMacro feature. This function resolves each defined String Macro in the String Template by replacing the String Macro with its value.

String Templates may contain mathematical expressions as well which can be evaluated either by using the built-in macro called EVAL or the autoEVAL functionality of the SMacro feature. This feature replaces all mathematical expressions in the String Template with their calculated values automatically.

Undefined String Macros remain in the String Template unchanged.

# Rules for String Macros

This chapter describes how the String Macros should be specified in the String Templates.

## String Macro Syntax

String Macros should follow the following syntax in the String Template:

1. Simple String Macro: starts with $ character followed by the macro name.  
   Example: $MACRO\_NAME
2. Parenthesis can also be used. This notation is equivalent with the previous one if the String Macro does not have any parameters.  
   Example: $(MACRO\_NAME)
3. String Macro with parameters: String Macros can have parameters. The parameters should follow the macro name and they should be separated by comma character. The value of the parameter should be enclosed in quotation marks.  
   Example: $(MACRO\_NAME, "par1", "par2", " par3 ")

**Note:** if it is given as a charstring value, the quotation marks (and also other special characters) have to be escaped according to the TTCN-3 standard:  
"$(MACRO\_NAME, \"par1\", \"par2\", \" par3 \")"   
or  
"$(MACRO\_NAME, ""par1"", ""par2"", "" par3 "")"

## String Macro Name

Macro names must be TTCN-3 charstringtype, but only the following characters are allowed in the macro name:

* lowercase: [a-z]
* uppercase: [A-Z]
* number [0-9]
* underscore: \_

Example: $USER01\_NAME

## String Macro Parameters

String Macros can have any number of TTCN-3 charstringtype parameters which must be separated by comma characters from the macro name and from each other. Each parameter is passed to the macro calculator function (if it is registered for the String Macro) when the macro value has to be determined during processing the String Template.

Example:

|  |  |
| --- | --- |
| String Template | ”How much is $(SUM, \“31”\, \“11”\)?” |
| String Macro | SUM(arg1,arg2) |
| Value of String Macro | arg1&”+”&arg2 |
| Resolved String Template | “How much is 31+11?” |

In the example above the value of the SUM macro is calculated through a custom function which concatenates its two arguments and places a “+” sign between them.

As you can see in the example above the parameters are enclosed in quotation marks which are escaped with “\” character. This has to be done according to the rules mentioned in section 4.3.2.

### String Template as a Parameter

The parameters of the String Macros can be String Templates. It means that the parameters can contain String Macros.  
  
Example:

|  |  |
| --- | --- |
| String Template | “The sender address is: $(EMAIL,\”$(USER)”\,\ “$(HOST)”\)” |
| String Macros | EMAIL(name, host) USER HOST |
| Value of String Macros | name&”@”&host “ethjgi” “ericsson.com” |
| Resolved String Template | “The sender address is: ethjgi@ericsson.com” |

### Escape Rules for String Macro Parameters

As you can see in the rules above the parameters have to be enclosed in quotation marks in order to the parser of the SMacro feature can recognize them. Since the String Templates are TTCN-3 charstringtype these quotation mark needs to be escaped in the String Template when it is used in TTCN-3 code.

#### Escaping in 1st Level

The theoretical syntax of the String Macro with parameters looks like this (this is what you see when you open a file that contains this String Template):

*MACRO1 value: $(MACRO1, "parameter1", "parameter2")*

*MACRO2 value: $(MACRO2, "parameter3")*

However, when it is read into a TTCN-3 charstring, it looks like this (the whole string is enclosed in quotation marks and special characters are escaped):

"MACRO1 value: $(MACRO1, \"parameter1\", \"parameter2\")

MACRO2 value: $(MACRO2, \"parameter3\")"

In practice it is recommended to use the log2str built-in TITAN function which does this escaping task automatically. Example TTCN-3 code to create the String Template above:

var charstring my\_stringTemplate :=

"MACRO1 value: $(MACRO1, "&log2str("parameter1")&", "&log2str("parameter2")&")

MACRO2 value: $(MACRO2, "&log2str("parameter3")&")";

#### Higher escape levels

The escape level depends on how deep the macro is embedded into the parameters. For example, when MACRO2 is given as the 2nd parameter of MACRO1 (theoretical syntax), the String Template reads:

*My string template:  
$(MACRO1,"parameter1","$(MACRO2, \"parameter3\")")*

According to the escaping rules it follows, in TTCN3 charstring the already escaped quotation mark should be escaped again, so the String Template in TTCN3 looks like this:

"My string template:  
$(MACRO1, \"parameter1\", \"$(MACRO2, \\\"parameter3\\\")\")"

Much more readable version with log2str built-in TITAN function:

"My string template:  
$(MACRO1, "&log2str("parameter1")&", "  
&log2str("$(MACRO2, "&log2str("parameter3")&")")&")"

It can be simplified even more if the parameters are put into charstring variables:

var charstring vl\_par1 := "parameter1";

var charstring vl\_par3 := "parameter3";

var charstring vl\_par2 :=   
"$(MACRO2, "&log2str(vl\_par3)&")";

var charstring vl\_stringTemplate :=  
"My string template:  
$(MACRO1, "&log2str(vl\_par1)&", "  
&log2str(vl\_par2)&")"

## String Macro Value

The String Macro value is a constant value given at the definition of the String Macro. However, it can also be calculated by a custom function during runtime. The value of the String Macro can be any value accepted by TTCN-3 charstring type. It can also contain String Macros, which means that the value of the String Macro can be a String Template.

### String Template as a Value

The value of the String Macro can be a String Template as well. In this case the resolved String Template is re-evaluated until the final string does not contain any String Macro.

Example:

|  |  |
| --- | --- |
| String Template | “The sender address is: $(SENDER)” |
| String Macros | SENDER USER |
| Value of String Macros | “$USER@ericsson.com” “ethjgi” |
| Resolved String Template | “The sender address is: ethjgi@ericsson.com” |

### Handling Undefined String Macros

String Macro which cannot be resolved stays in the String Template without changes.

Example:

|  |  |
| --- | --- |
| String Template | “Username is: $USER” |
| String Macro | UNKNOWN |
| Value of String Macro | Not defined |
| Resolved String Template | “Username is: $USER” |

## Calculating String Macro Value with Custom Functions

The value of the String Macros can be calculated by user defined callback functions.

### Callback Function Type

Macro calculator functions must be type of:

f\_EPTF\_CLL\_SMacro\_calc\_FT(  
 in EPTF\_CharstringList pl\_args,  
 in EPTF\_IntegerList pl\_userArgs := {}  
) runs on self return charstring

The pl\_args formal parameter gets its value from the parameters of the String Macro in the String Template.

The return value must be TTCN-3 charstring type. Its return value is used as the value of the String Macro for which this function is registered.

Example for the SUM macro:

function f\_calc\_sum(  
 in EPTF\_CharstringList pl\_args,  
 in EPTF\_IntegerList pl\_userArgs := {}) {

return pl\_args[0]&”+”&pl\_args[1];

}

### Register/Deregister Callback Function

Macro calculator functions should be registered by calling f\_EPTF\_SMacro\_registerCalcFn function.

Example:

f\_EPTF\_SMacro\_registerCalcFn(  
 pl\_functionName := “f\_calc\_sum”,  
 pl\_macro\_function := refers(f\_calc\_sum)  
)

It is possible to register the same function with different names. If the register function is called more than once with the same function name, the last function reference is used for that name, therefore it is possible to reregister the callback function.

Callback functions can be deregistered by calling f\_EPTF\_SMacro\_deregisterCalcFn function.

### Define String Macros

String Macros can be defined by calling f\_EPTF\_SMacro\_define function. This function assigns a value to the given macro name.

When the value of the String Macro should be calculated through custom function, the value given for the define function should be the same as the registered name of the callback function.

Example: defining SUM macro where the macro value is an already registered function name

f\_EPTF\_SMacro\_define(“SUM”, “f\_calc\_sum”);

|  |  |
| --- | --- |
| String Template | ”How much is $(SUM, \“31”\, \“11”\)?” |
| Registered function for SUM | f\_calc\_sum |
| Resolved String Template | “How much is 31+11?” |

If the “f\_calc\_sum” is not registered or registered as a NULL macro calculation function, the macro has the constant defined value “f\_calc\_sum”.

Example:

|  |  |
| --- | --- |
| String Template | ”How much is $(SUM, \“31”\, \“11”\)?” |
| Registered function for SUM | NULL |
| Resolved String Template | “How much is f\_calc\_sum?” |

## EVAL Built in Macro

There is a built in macro called EVAL to evaluate mathematical expressions. It does not have to be defined by the user. However, it can be redefined or its function (f\_EPTF\_SMacro\_calcFn\_EVAL) can be re-registered or even undefined if necessary with the public API functions.

Example:

|  |  |
| --- | --- |
| String Template | “The value of $(SUM, \“31\”, \“11\”) is: $(EVAL,\”$(SUM, \\\“31\\\”,\\\“11\\\”)\”).” |
| String Macro | SUM(arg1,arg2) |
| Value of String Macro | arg1&”+”&arg2 |
| Resolved String Template | “The value of 31+11 is: 42.” |

### Supported Operators

The followings operators are supported:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Precedence level** | **Type of 1st argument** | **Type of 2nd argument** | **Type of result** |
| parenthesis | 0 | float | n.a. | float |
| Multiplication \* | 1 | float | float | float |
| Division / | 1 | float | float (ERROR in case of zero) | float |
| Division / (modulus) | integer | integer (ERROR in case of zero | integer |
| Remainder % | 1 | integer | integer (ERROR in case of zero) | integer |
| Addition + | 2 | float | float | float |
| Subtraction - | 2 | float | float | float |

If the divisor is zero in case of division or remainder, the macro value is resolved according to the following string: (EVAL: Error: Division by zero.).

The evaluation of operators with the same precedence level is performed from left to right.

### Auto-EVAL

It is possible to automatically call the built-in EVAL macro when the String Template is fully resolved. (That is when there is no more String Macro that needs to be resolved in the String Template). This functionality can be turned on by setting the pl\_autoEVAL parameter of f\_EPTF\_SMacro\_resolve function to true.

Example: without autoEVAL

|  |  |
| --- | --- |
| String Template | "Result of 2+4/2\*3-1 is: $(EVAL, \"2+4/2\*3-1\" )." |
| Resolved String Template | “Result of 2+4/2\*3-1 is: 7.” |

Example: with autoEVAL set to true (even if there is no EVAL macro in the String Template, the mathematical expression is calculated)

|  |  |
| --- | --- |
| String Template | "Result of 2+4/2\*3-1 is: $(EVAL, \"2+4/2\*3-1\" )." |
| Resolved String Template | “Result of 7 is: 7.” |

# Functional Interface

Apart from this description a cross-linked reference guide for the EPTF CLL Functions can be reached for on-line reading [5]. This chapter describes the public interface of the SMacro feature.

## Naming Conventions

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

## Public Functions

### Initialization of the SMacro Feature

Before using the EPTF SMacro functions, the

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

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

It does the following:

* initializes the needed EPTF CLL features (Base, HashMap, FBQ)
* creates and initializes the needed component variables
* registers the built-in EVAL macro
* registers the clean-up function of SMacro (f\_EPTF\_SMacro\_cleanup\_CT)

### Define String Macro

The following function can be used to define String Macro.

function f\_EPTF\_SMacro\_define(  
 in charstring pl\_macro\_name,  
 in charstring pl\_macro\_value  
) runs on EPTF\_SMacro\_CT return integer

It assigns the macro value given by pl\_macro\_value to the macro name given by pl\_macro\_name.

It is possible to redefine macro value by calling the function again. In that case the latest assigned value is used.

If the macro is successfully defined, the return value is zero, otherwise non-zero.

### Undefine String Macro

The following function can be used to undefine String Macro.

function f\_EPTF\_SMacro\_undefine(  
 in charstring pl\_macro\_name)

It removes the assignment between the macro name given by pl\_macro\_name and its macro value. So the macro is undefined after calling this function.

Undefined String Macro is not resolved in the String Template, therefore the content of the String Template is unchanged.

### Registering Macro Calculator Callback Function

The following function can be used to register macro calculator callback functions:

function f\_EPTF\_SMacro\_registerCalcFn(  
 in charstring pl\_functionName := "",  
 in f\_EPTF\_CLL\_SMacro\_calc\_FT pl\_macro\_function,  
 in EPTF\_IntegerList pl\_userArgs := {}  
) runs on EPTF\_SMacro\_CT

It registers the function given by pl\_macro\_function with function name given by pl\_function\_name.

The parameter pl\_function\_name is optional, by default it is an empty string. If it is not specified, the function name is automatically set to log2str(refers(pl\_macro\_function)) without refers()statement.

The parameter pl\_userArgs is an integer list which can be used to pass parameters to the macro calculator function when it is invoked. It is optional, by default it is an empty list.

Example:

f\_EPTF\_SMacro\_registerCalcFn(  
 pl\_macro\_function := refers(f\_calc\_sum)   
)

The macro calculator function is registered according to the followings:

* functionName := “<module name>.f\_calc\_sum”
* macro\_function := refers(<module name>.f\_calc\_sum)

### De-registering Macro Calculator Callback Function

The following function can be used to de-register macro calculator call back functions:

function f\_EPTF\_SMacro\_deregisterCalcFn(  
in charstring pl\_functionName := ""  
)

It de-registers the function given by pl\_function\_name.

### Resolve String Macro

The following function can be used to resolve the String Template given by pl\_stringTemplate:

function f\_EPTF\_SMacro\_resolve(  
 in charstring pl\_stringTemplate,  
 in boolean pl\_autoEVAL := false  
) runs on EPTF\_SMacro\_CT return charstring

It returns with the value of the String Template where all defined String Macro is replaced by its value.

If pl\_autoEVAL formal parameter is set to true, all mathematical expressions are evaluated after each String Macro is resolved in the String Template. It is the same behavior as if the EVAL built-in macro was applied on the whole String Template.

### Built-in Callback Function for EVAL Macro

The following function is used to calculate the value of the built-in EVAL String Macro.

function f\_EPTF\_SMacro\_calcFn\_EVAL(

in EPTF\_CharstringList pl\_macroArgs,

in EPTF\_IntegerList pl\_userArgs := {}

) runs on EPTF\_SMacro\_CT return charstring

This function can be registered to be used in user defined String Macros.

## Summary Table of All Public Functions for EPTF SMacro

Table 1. Summary of SMacro functions

|  |  |
| --- | --- |
| **Function name** | **Description** |
| f\_EPTF\_SMacro\_init\_CT | Initializes the SMacro Component |
| f\_EPTF\_SMacro\_define | Defines the String Macro |
| f\_EPTF\_SMacro\_undefine | Undefines the String Macro |
| f\_EPTF\_SMacro\_registerCalcFn | Registers macro calculator callback function |
| f\_EPTF\_SMacro\_deregisterCalcFn | Deregisters macro calculator callback function |
| f\_EPTF\_SMacro\_resolve | Resolves the String Template |
| f\_EPTF\_SMacro\_calcFn\_EVAL | Built-in callback function for EVAL String Macro |

# 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 113 200/4 Uen  
   User Guide for the TITAN TTCN-3 Test Executor
3. 109 21-CNL 113 512-20 Uen   
   EPTF CLL for TTCN-3 toolset with TITAN, Product Revision Information
4. 155 17-CNL 113 512 Uen  
   EPTF CLL for TTCN-3 toolset with TITAN, Function Specification
5. 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)