ETSI ES 203 119-2 V1.2.1 (2016-09)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 2: Graphical Syntax

Reference

RES/MTS-203119-2v1.2.1

Keywords

graphical notation, language, MBT, methodology, testing

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00   Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document is part 2 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies the concrete graphical syntax of the Test Description Language (TDL). The intended use of the present document is to serve as the basis for the development of graphical TDL tools and TDL specifications. The meta-model of TDL and the meanings of the meta-classes are described in ETSI ES 203 119-1 [1].

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI ES 203 119-1 (V1.3.1): "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 136 523-1 (V10.2.0) (10-2012): "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 1: Protocol conformance specification (3GPP TS 36.523-1 version 10.2.0 Release 10)".

[i.2] ETSI TS 186 011-2 (V3.1.1) (06-2011): "IMS Network Testing (INT); IMS NNI Interoperability Test Specifications; Part 2: Test Description for IMS NNI Interoperability".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

diagram: placeholder of TDL shapes

lifeline: vertical line originates from a gate instance or a component instance, to which behavioural elements may be attached

NOTE: A lifeline from top to down represents how time passes.

shape: layout of the graphical representation of a TDL meta-class

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

EBNF	Extended Backus-Naur Form
IMS	IP Multimedia Subsystem
OCL	Object Constraint Language
TDL	Test Description Language
URI	Unified Resource Identifier

4 Basic principles

4.1 Introduction

The meta-model of the Test Description Language is specified in ETSI ES 203 119-1 [1]. The presentation format of the meta-model can be different according to the needs of the users or the requests of the domain, where the TDL is applied. These presentation formats can either be text-oriented or graphic-oriented and may cover all the functionalities of the TDL meta-model or just a part of it, which is relevant to satisfy the needs of a specific application domain.

The present document specifies a concrete graphical syntax that provides a graphical representation for the whole functionality of the TDL meta-model.

The document specifies the TDL diagram, where the graphical representations of the instances of the TDL meta-classes may be placed. A graphical representation may contain a shape with textual labels placed into it. The rules, how these labels shall be interpreted are described in OCL-like expressions.

4.2 Document Structure

The present document specifies the concrete graphical syntax of the Test Description Language (TDL).

Clause 5 specifies the TDL Diagram.

Clause 6 specifies the concrete shapes defined for the TDL meta-classes. (The meta-model of TDL and the meanings of the meta-classes are described in ETSI ES 203 119-1 [1].)

- Foundation (clause 6.1)
- Data (clause 6.2)
- Time (clause 6.3)
- Test Configuration (clause 6.4)
- Test Behaviour (clause 6.5)

At the end of the document several examples illustrating the features of the TDL Graphical Syntax can be found.

4.3 Notational Conventions

4.3.0 General

Elements from the TDL meta-model [1] are typed in italic, e.g. *StructuredDataType*.

The definition of the TDL concrete graphical syntax consists of both shapes and textual labels placed into these shapes. Textual labels are differentiated into non-terminal textual labels and terminal textual labels. The production rule of a non-terminal textual label is specified by a combination of EBNF symbols and OCL-like expressions to navigate over the abstract syntax meta-model of TDL.

4.3.1 Symbols and meanings for shapes

Shapes consist of outermost borders, compartments, and textual labels (i.e. non-terminal textual labels and terminal-textual labels). The following conventions apply:

- Non-terminal textual labels are typed in small capitals (e.g. PRODUCTIONRULELABEL). The name of the label refers to a production rule with the same name that specifies how the result of the production rule is determined.
- If a non-terminal symbol name is typed in special, e.g. UNDERLINED or **BOLD** small capitals, underlined or bold font shall be used in the shape for the result of the production rule of that non-terminal symbol, e.g. SIMPLEDATAINSTANCENAMELABEL (non-terminal) and MyValue:MyType (a result of the production rule of that non-terminal) or **COMPONENTROLELABEL** (non-terminal) and **TESTER** (a result of the production rule of that non-terminal), etc.
- Terminal textual labels are typed in non-small-capital characters. They shall be typeset in the same font, as they appear on the figure, e.g. if a terminal textual label is typed in **bold**, bold font shall be used in the shape for that terminal textual symbol, e.g. **timer**, etc.
- The outermost border of a shape shall not be hidden, unless it is stated explicitly.
- Compartments and non-terminal textual labels may be hidden to simplify the internal structure of the shape.
- In the figures, optional compartments are shaded in a light grey colour, while optional non-terminal textual labels are typed in grey colour. However, the colour and the shading indicates only the optionality of a compartment or a non-terminal label. That is, if they are actually present in a test description, they shall not be shaded and shall be typed in black.
- If a non-terminal textual label is defined to be optional, that non-terminal textual label shall only be shown if the surrounding compartment is shown and the corresponding non-terminal textual production rule results in a non-empty string or a non-empty collection of strings.
- If an optional compartment contains a mandatory terminal or non-terminal textual label, the text shall only be shown if the surrounding compartment is shown.
- References to non-terminal textual production rules external to the given shape are represented by the name of the referenced production rule enclosed in angle brackets (e.g. <REFERENCEDPRODUCTIONRULE>).
- A non-terminal textual label in between hashmarks (e.g. #ELEMENT#) denotes a placeholder for a shape identified by that non-terminal textual label.

4.3.2 Symbols for non-terminal textual labels

Non-terminal textual labels are specified by production rules (so called non-terminal textual label production rule). The formal specification of a non-terminal textual label production rule is expressed by OCL. The context meta-model element for the OCL expression is specified prior to the non-terminal textual label specification. In some cases, the definition of OCL expression would be too complex for understanding. In that case, pseudo-code like helper notations are used.

The OCL expressions are combined with a variant of the Backus-Naur Form (Extended Backus-Naur Form - EBNF). The conventions within the present document for the production rules are:

- OCL keywords and helper functions are typed in **bold.**
- The keyword **context** followed by the name of TDL metaclass determines the context element for the following production rule (e.g. **context** Package).
- Non-terminal textual labels production rule identifiers are always represented in small capitals (e.g. LABELPRODUCTIONRULE).
- Non-terminal textual label production rule definitions are signified with the '**::=** ' operator.
- OCL expressions are written in lower case characters (e.g. self.name).
- Non-terminal textual labels may contain terminal symbols. A terminal symbol is enclosed in single quotes (e.g. 'keyword' or '[').
- Alternative choices between symbols in a production rule are separated by the '|' symbol (e.g. symbol1 | symbol2).
- Symbols that are optional are enclosed in square brackets ' [ ] ' (e.g. [symbol]).
- In case the context of an OCL expression needs to be changed for non-terminal textual label production rule, the predefined function *variable* **as context in <LABELPRODUCTIONRULE>** shall be used to invoke a production rule of a different metaclass, where *variable* refers to an instance of a metaclass that complies with the context of the invoked **<LABELPRODUCTIONRULE>.**
- If the OCL expression of a production rule results in a collection of strings, a collection helper function **separator(String)** is used to specify the delimiter between any two strings in the collection, e.g. self.collectionProperty->**separator(',')**. The collection helper function **newline( )** inserts a line break between any two strings in the collection.
- Iterations over collections of attributes of a metaclass use a verbatim (non-OCL) helper function *foreach* with the following syntax: **foreach** *VariableName* ':' *VariableType* [**separator(String)|newline( )**] **in** *OCLexpression* **end**. *VariableName* is an alphanumeric word signifying the variable used for subsequent statement. *VariableType* is a string that shall be the same as a TDL metaclass name. *OCLexpression* is an OCL statement that resolves in a collection of metaclass elements compliant to the metaclass given in *VariableType*. For example, the statement LABEL **::= foreach** *e:Element* ***in*** *self.attribute* **end**, iterates of the elements in the collection *self.attribute* and stores resulting element of each iteration in variable *e*. The variable *e* can be used in the body of the loop for further calculations. In every iteration, the non-terminal textual production rule LABEL is invoked, and the respective instance of metaclass *Element* that is stored in *e* will be used in the invoked production rule. The collection helper functions **separator(String)** and **newline( )** may also be applied directly to the **foreach** construct.

4.3.3 Examples

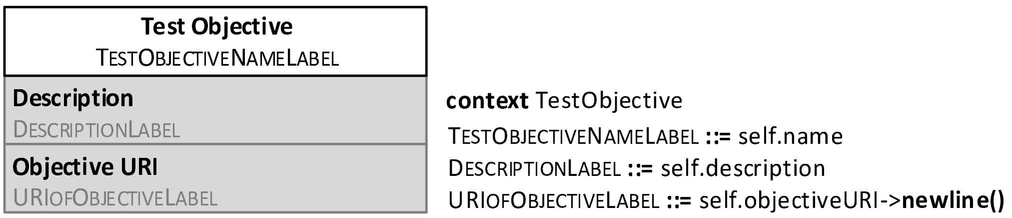
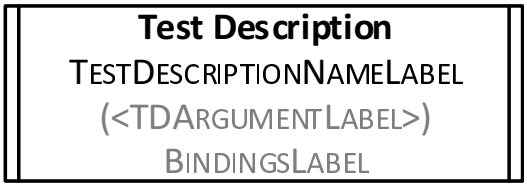


Figure 4.1: Notational convention example 1

In figure 4.1, the following notational concepts of the TDL Concrete Graphical Syntax are shown:

- The uppermost compartment contains a terminal textual label (a keyword) 'Test Objective' typed in bold.
- The context meta-model element of this shape is *TestObjective*.

- The non-terminal textual label production rule TESTOBJECTIVENAMELABEL results in the name of the context element (i.e. self.name).
- There are two optional compartments (i.e. shaded grey) shown ordered from top to down.
- Both compartments contain a mandatory terminal textual label (i.e. the label shall be shown if the surrounding compartment is shown). The terminal textual labels shall be typed in bold (**Description** and **Objective URI**, respectively).
- Both compartments contain an optional non-terminal textual label (i.e. the label shall be shown if the surrounding compartment is shown and the production rules results in a non-empty string or a non-empty collection of strings).
- The separator between the elements of the self.objectiveURI in production rule URIOFOBJECTIVELABEL is a new line.




```
context TestDescriptionReference
TESTDESCRIPTIONNAMELABEL ::= self.testDescription.name

TDARGUMENTLABEL ::= foreach d:DataUse in self.actualParameter separator(',')
                                d as context in <DATAUSELABEL>
                    end

BINDINGSLABEL ::= foreach c : ComponentInstanceBinding in self.componentInstanceBinding separator(',')
                              c.componentInstanceBinding.actualComponent.name '->'
                              c.componentInstanceBinding.formalComponent.name
                  end
```


Figure 4.2: Notational convention example showing the foreach helper function

In figure 4.2, the use of a non-OCL *foreach* helper function is illustrated. The context element when entering the foreach loop is *TestDescriptionReference*. The first foreach loop assigns iteratively each element in the collection *self.actualParameter* to the variable *d* of type *DataUse*. The variable *d* then used as it is described in the referenced production rule DATAUSELABEL. The separator between the results of the iterations is ',' (a comma character). The second foreach loop assigns iteratively each element in the collection *self.componentInstanceBinding* to the variable *c* of type *ComponentInstanceBinding*. The variable *c* is then used in a subsequent non-terminal textual label production rule to build the label for the production rule. The separator between the results of the iterations is ',' (a comma character).

4.4 Conformance

For an implementation claiming to conform to this version of the TDL Concrete Graphical Syntax, all features specified in the present document and in ETSI ES 203 119-1 [1] shall be implemented consistently with the requirements given in the present document and ETSI ES 203 119-1 [1].

5 Diagram

There is only one diagram kind provided by TDL Concrete Graphical Syntax. This diagram is called TDL Diagram. There may be multiple instances of a TDL Diagram at the same time.

The shapes that may be placed onto a TDL Diagram are specified in clause 6.

6 Shapes

6.1 Foundation

6.1.1 Element

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

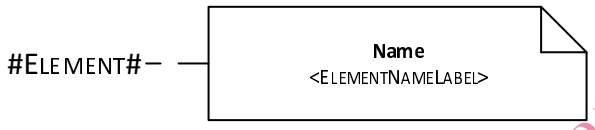
Formal Description

context Element

ELEMENTNAMELABEL ::= self.name

Comments

To a shape of any subclass of *Element*, the name of that *Element* may be attached by a thin dashed line unless it is stated otherwise in the shape definition of a given subclass of *Element*.



6.1.2 NamedElement

Concrete Graphical Notation

This is an abstract metaclass, therefore no graphical representation is defined.

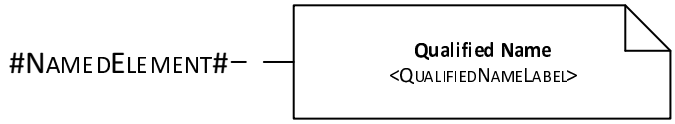
Formal Description

context NamedElement

QUALIFIEDELEMENTLABEL ::= self.qualifiedName

Comments

To a shape of any subclass of *NamedElement*, the qualified name of that *NamedElement* may be attached by a thin dashed line.



6.1.3 ElementImport

Concrete Graphical Notation

This metaclass has no dedicated shape, it is used solely in the shapes of other metaclasses.

Formal Description

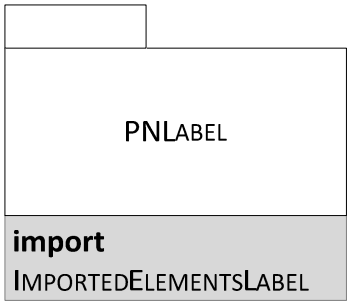
```
context ElementImport
IMPORTLABEL ::= 'from' self.importedPackage.qualifiedName
        if self.importedElement->isEmpty() then
            'all'
        else
            self.importedElement.name->separator(',')
        endif
```

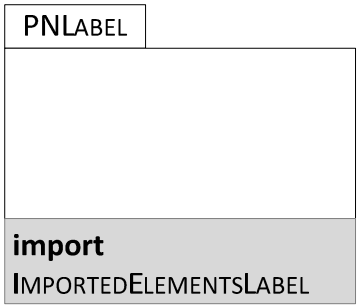
Comments

No comments.

6.1.4 Package

Concrete Graphical Notation





Formal Description

```
context Package
PNLABEL ::= self.name
IMPORTEDELEMENTSLABEL ::= foreach i:ElementImport in self.import
                              i as context in <IMPORTLABEL> separator(',')
                          end
```

Comments

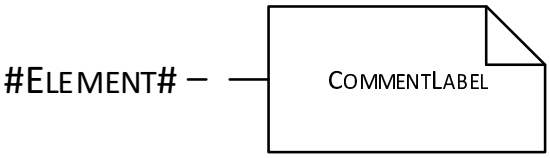
The figures above indicate the two possible representations of the *Package* shape: the PNLABEL may be written either in the top, small compartment or in the middle one.

The elements the package contains (packagedElements) may be shown within the large rectangle in the middle. In this case the PNLABEL shall be in the upper small compartment.

The lower **import** compartment is optional, it shall only be represented if the package imports other package(s) or elements from other package(s). If this compartment is present, its content shall also be present.

6.1.5 Comment

Concrete Graphical Notation



Formal Description

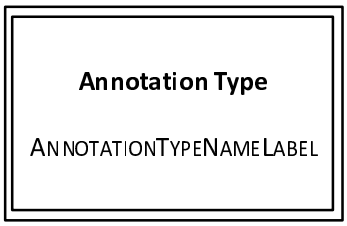
context Comment
COMMENTLABEL ::= self.body

Comments

A *Comment* shape shall be attached to the commented element by a thin dashed line.

6.1.6 AnnotationType

Concrete Graphical Notation



Formal Description

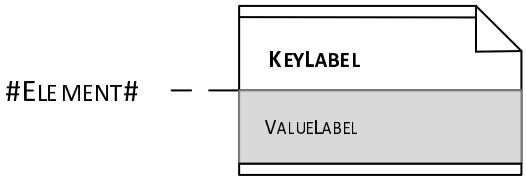
context AnnotationType
ANNOTATIONTYPENAMELABEL ::= self.name

Comments

No comments.

6.1.7 Annotation

Concrete Graphical Notation



Formal Description

context Annotation
KEYLABEL ::= self.key.name
VALUELABEL ::= self.value