ETSI ES 203 119-8 V1.2.1 (2023-10)

Methods for Testing and Specification (MTS);
The Test Description Language (TDL);
Part 8: Textual Syntax

Reference

RES/MTS-TDL1

Keywords

language, MBT, methodology, testing, TSS&TP, TTCN-3, UML

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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

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

The present document is part 8 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 textual syntax of the Test Description Language (TDL). The intended use of the present document is to serve as the basis for the development of textual 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].

NOTE: OMG®, UML® and OCL™ are the trademarks of Object Management Group (OMG). Xtext™ is trademark of the Eclipse® Foundation. This information is given for the convenience of users of the present document and does not constitute an endorsement by ETSI of the products named.

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 https://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: "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] Eclipse® Foundation: Xtext - The Grammar Language Website.

[i.2] ETSI TS 136 523-1 (V10.2.0): "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.3] ETSI TS 186 011-2: "Core Network and Interoperability Testing (INT); IMS NNI Interoperability Test Specifications (3GPP Release 10); Part 2: Test descriptions for IMS NNI Interoperability".

[i.4] ETSI: TDL Open Source Project.

[i.5] ETSI ES 203 119-4: "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 4: Structured Test Objective Specification (Extension)".

[i.6] ETSI ES 203 119-7: "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 7: Extended Test Configurations".

[i.7] ETSI TS 129 165: "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; Inter-IMS Network to Network Interface (NNI) (3GPP TS 29.165)".

[i.8] ETSI TS 124 229: "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); LTE; 5G; IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3 (3GPP TS 24.229)".

[i.9] ETSI TS 136 213: "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (3GPP TS 36.213)".

[i.10] ETSI TS 136 508: "LTE; Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); Common test environments for User Equipment (UE) conformance testing (3GPP TS 36.508)".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI ES 203 119-1 [1] and the following apply:

derivation: construction of an abstract syntactical structure, such as a model instance conforming to a meta-model, from a textual representation by applying the structural rules of a grammar, and potential mappings to the underlying meta-model

(formal) grammar: set of structural rules that define how to form valid strings from a language's alphabet that obey the syntax of the language

non-terminal symbol: placeholder for (groups of) other symbols that describe elements in a specified language

(production) rule: definition of a structured rule for the derivation of a non-terminal symbol based on other non-terminal symbols and terminal symbols

terminal symbol: symbols that appear explicitly in a specified language, such as a keyword, an identifier or other tokens

3.2 Symbols

Void.

3.3 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™
OMG	Object Management Group®
SUT	System Under Test
TDL	Test Description Language
TTCN-3	Testing and Test Control Notation version 3
UML	Unified Modelling Language®
URI	Uniform Record Identifier

4 Basic principles

4.1 Introduction

The meta-model of the Test Description Language (TDL) 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 requirements 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 textual syntax that provides a textual representation for the commonly used functionality of the TDL meta-model. In the current version of the present document, certain parts, such as 'Comment's and 'Annotation's in 'DataUse' elements, are syntactically excluded. Syntactic specifications for these may be added in future versions of the present document as needed.

The present document specifies the TDL textual file format, where the textual representations of the instances of the TDL meta-classes may be placed. A textual representation may contain keywords, delimiters, and textual labels within a defined structure. The rules, how these structures shall be interpreted, are described by means of Extended Backus-Naur Form (EBNF)-like expressions. In particular, in addition to the syntactical structure, the EBNF-like expressions also indicate how the textual labels and structures are mapped to the TDL meta-model.

4.2 Document Structure

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

Clause 5 specifies general rules for the specification and use of the TDL textual file format.

Clause 6 specifies the concrete production rules 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 present document several examples illustrating the features of the TDL Textual Syntax can be found.

4.3 Grammar Language

4.3.1 Overview

The rules that define the textual syntax of the TDL are described in present document using the grammar language of the Xtext framework. In addition to defining the lexical structure of the TDL syntax the grammar language also provides means for mapping those textual constructs to the TDL meta-model. Additional rules such as identity resolution and linking are described where applicable to provide complete mapping of textual TDL to the TDL model.

The grammar of textual TDL is composed of a number of grammar rules organized in a tree. The grammar structure follows the logical structure of the TDL meta-model and the root of the grammar is the 'Package' production rule. Production rules are used to construct model objects and assign values to the properties of those objects. Production rules consist of keywords (character literals) and calls to production rules, data type rules and terminal rules (which correspond to tokens of text).

The following clauses describe the syntax of the grammar language. See Xtext documentation for further details [i.1].

4.3.2 Operators

Various operators are used in grammar rule definitions to specify the order and cardinality of keywords and rule calls. Terminal rule specific operators are used to express various textual constructs. Production rule specific operators are used to define assignments and cross-references.

Following operators are used in all rule definitions:

- '?' indicates that preceding construct shall occur 0 or 1 time;
- '*' indicates that preceding construct shall occur 0 or more times;
- '+' indicates that preceding construct shall occur 1 or more times;
- '|' is used between alternative constructs; and
- '(' and ')' are used to group constructs defined in between.

Following operators are used in terminal rule definitions:

- '!' is used to negate a construct;
- '->' is used to indicate that everything is ignored until the following construct is detected;
- '..' is used between characters to define a range; and
- '.' denotes any character.

Following operators are used in production rule definitions:

- '=' is used to define a simple assignment of a right hand construct to a property on the left;
- '+=' is used for assigning (adding to) multi-valued property;
- '?=' is used for assigning the value 'true' to a Boolean property on the condition that the right hand side construct is present; and
- '[', '|' and ']' are used to define a cross-reference.

Various special symbols are included in the grammar definitions of production rules that are included solely as implementation detail (to help the generation of a parser for textual TDL) and do not alter the definition of the syntax. Such symbols include '->' and '=>'.

4.3.3 Terminal rules and keywords

Lexical tokens in the TDL grammar are either keywords of character sequences that are matched and consumed by terminal rules during parsing. In the grammar definition, keywords are placed between apostrophes (').

Terminal rule declarations start with the keyword 'terminal' followed by the rule name (in upper-case letters by convention). The rule name is followed by 'returns' keyword and the reference to a data type that is used for creating a value using the consumed token.

The definition of the rule starts with a colon (':') and ends with a semi-colon (';'). Terminal rule definitions consist of terminal rule calls (indicated by rule name), characters and operators.

EXAMPLE: Terminal INT returns EInt: ('0'..'9')+;

Some terminal rules (such as comments and whitespace) are defined as hidden in TDL grammar and corresponding text shall be allowed anywhere in textual TDL (outside of tokens).

4.3.4 Production rules

Production rules are used to create model objects or data values. The rules that return a data type instead of a meta-class are known as data type rules.

Production rule declarations start with the rule name followed by 'returns' keyword and the reference to the meta-class that defines the object that is produced by the rule. The definition of the rule starts with a colon (':') and ends with a semi-colon (';'). Production rule definitions consist of rule calls, keywords and operators.

EXAMPLE 1:

```
Comment returns tdl::Comment:
    'Note:' body=EString
;
```

An assignment is defined as a property name followed by an assignment operator (see clause 4.3.2) followed by a rule call (name of production or data type rule) or a cross-reference. A cross-reference is defined as a meta-class reference followed by '|' and a terminal rule call that defines the format for the identifier. The cross-reference definition is placed between square brackets ('[' and ']').

EXAMPLE 2:

```
Annotation returns tdl::Annotation:
    '@' key=[tdl::AnnotationType|Identifier]
    (':' value=EString)?
;
```

Production rule calls may also be used without assignment. In that case the model object that is returned from the calling rule is the one that is created in the called rule.

Production rules may be created as fragments by prefixing the declaration with the 'fragment' keyword. In that case the rule does not produce an object by itself but rather assigns to properties of the object that is created in the calling rule. Fragment rules are always unassigned.

4.4 Conformance

For an implementation claiming to conform to this version of the TDL Concrete Textual 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 General rules

5.1 Identities and references

In TDL models, references between objects are based on unique identifiers that are generated by the modelling framework and stored in model files. Such identifiers are generally hidden from the user. In textual TDL, all attributes shall be part of the text document and the use of such identifiers is not feasible.

In textual TDL, objects are identified by 'name' or 'qualifiedName' property. The allowed values for the 'name' property are restricted by the terminal rule 'ID' (see clause 5.5). The exception to this rule is made for objects that are predefined in TDL and are mapped to special symbols in textual TDL (such as AnyValue).

If the 'name' property shall have a value that is equal to a keyword in textual TDL then that value shall be prefixed with '^' in the text.

5.2 Models and importing

TDL objects stored in a single file are collectively referred to as model. Both the TDL model and textual TDL allow single 'Package' object as the root of the model. Thus, logically the root package of a TDL file is a TDL model.

Naming of textual TDL files and the location of those files is out of the scope of the present document. Implementations of the textual TDL shall provide means to make TDL models available for importing.

Imported 'Package's shall be referred to by the value of the 'qualifiedName' property.

5.3 Linking

Linking refers to the phase in the compilation process of textual TDL where name-based cross-references are resolved to actual objects that they represent. By default, linking utilizes object identities as described in clause 5.1.

In some cases where explicit cross-references are not required by the grammar rules, the linking may apply context specific logic to assign references to object properties. Such cases are described in the relevant clauses.

5.4 Alternative syntaxes

Although the keywords are specified with certain case (lower-case or title-case) in the present document, the case itself is not prescribed. Therefore, an implementation can be case-insensitive as well. It is recommended that users apply a consistent case nonetheless.

The delimiters for 'Block's and other constructs are specified in an abstract manner with the 'BEGIN' and 'END' terminal symbols. While the default assumption is that these terminal symbols are mapped to left and right braces ('{' and '}'), referred to as 'brace-based' syntax, an alternative implementation using white space indentation is also possible, where synthetic delimiters for the beginning and end of indented parts shall be used instead, referred to as 'indentation-based' syntax. Besides the replacement of the 'BEGIN' and 'END' symbols, no other differences shall be present between implementations of the 'brace-based' and 'indentation-based' syntax. Left and right braces ('{' and '}') shall be used in certain contexts even within the 'indentation-based' syntax, e.g. for 'TimeConstraint's and data-related 'Constraint's.

The examples in the present document conform to the default assumption. Additional examples illustrating the indentation-based syntax are included in Annex B.

5.5 Terminals

The base terminal symbol definitions include the following:

```
terminal ID: '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
terminal INT returns EInt: ('0'..'9')+;
terminal STRING:
            '"' ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\'|'"') )* '"' |
            "'" ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\'|"'") )* "'"
        ;
terminal ML_COMMENT : '/*' -> '*/';
terminal SL_COMMENT : '//' !('\n'|'\r')* ('\r'? '\n')?;

terminal WS      : (' '|'\t'|'\r'|'\n')+;

terminal ANY_OTHER: .;

terminal TRUE : 'true';
terminal FALSE : 'false';

terminal BEGIN: '{';
terminal END: '}';
```


The 'WS', 'ML_COMMENT', and 'SL_COMMENT' tokens shall be hidden.

For the indentation-based syntax variant, the 'BEGIN' and 'END' terminal symbols are redefined to the following (with 'synthetic:BEGIN' and 'synthetic:END' representing an increase and a decrease in the indentation, respectively):

```
@Override
terminal BEGIN: 'synthetic:BEGIN';  // increase indentation

@Override
terminal END: 'synthetic:END';     // decrease indentation
```

In addition to the terminal symbols, data type parser rules for context-sensitive 'pseudo-terminals' include the following:

```
EString:
  STRING
;
```

```
Identifier:
  ID
;

GRIdentifier:
  ID ('::' ID)?
;

QIdentifier:
  ID ('.' ID)*
;

NIdentifier:
    ('-'? INT ('.' INT)?)
;

LBrace:
  BEGIN
;

RBrace:
  END
;

LParen:
  '('
;

RParen:
  ')'
;

BIGINTEGER returns ecore::EBigInteger:
    INT
;

BOOLEAN returns EBoolean:
    TRUE | FALSE
;
```


The 'LBrace' and 'RBrace' rules differentiate the use of left '{' and right '}' braces in certain contexts (e.g. 'Constraint's and 'TimeConstraint's) from their use as delimiters in the brace-based variant of the syntax. For the indentation-based variant of the syntax, these rules shall be overridden as follows:


```
//Retain Braces even in indentation-based
@Override
LBrace:
  '{'
;

@Override
RBrace:
  '}'
;

//for both indented and un-indented blocks within parentheses
@Override
LParen:
  '(' BEGIN?
;

@Override
RParen:
  END? ')'
;
```

The redefinition of the 'LParen' and 'RParen' with optional 'BEGIN' and 'END' tokens enables the use of indentation in blocks within parentheses in the indentation-based variant as all indentation is semantically relevant. In case indentation needs to be optionally allowed in other cases, a similar pattern can be applied for further tailoring of the indentation-based syntax variant.

5.6 File format

No assumptions are made about the file format at present. For practical purposes, certain conventions regarding the naming of files using the indentation-based and brace-based variants of the syntax are recommended, e.g. using different file endings or "extensions".

6 Production Rules

6.1 Foundation

6.1.1 Element

Concrete Textual Notation

```
fragment AnnotationFragment returns tdl::Element:
  (annotation+=Annotation)*


fragment AnnotationCommentFragment returns tdl::Element:
  (comment+=Comment)*
  (annotation+=Annotation)*
;

fragment NameFragment returns tdl::Element:
  'Name:' name=Identifier
;

fragment WithCommentFragment returns tdl::Element:
  'with'
  BEGIN
    (comment+=Comment)+
  END
;

fragment WithNameFragment returns tdl::Element:
  'with'
  BEGIN
    NameFragment
  END
;
```


Comments

This is an abstract metaclass, therefore no textual representation is defined for the element. The concrete textual notation represents reusable fragments that can be embedded in the concrete textual notation of metaclasses inheriting from this metaclass.

The different fragments are used in different contexts.

Examples

```
Note: "Example test objective"
@Example


with {
    Note: "Comment on nested package"
}

with {
    Name: anOptionalNameForElementWithoutMandatoryName
}
```