ETSI ES 201 873-1 V4.11.1 (2019-04)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 1: TTCN‑3 Core Language

**ETSI Standard**

Reference

RES/MTS-201873-1v4.11.1\_Core

Keywords

language, methodology, testing, TTCN-3

***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 prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

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>

***Copyright Notification***

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2019.

All rights reserved.

**DECT**TM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are trademarks of ETSI registered for the benefit of its Members.
**3GPP**TM and **LTE**TM 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.

## 15.3 Global and local templates

TTCN-3 allows defining global templates and local templates.

***Syntactical Structure***

**template** [ *restriction* ] [ **@fuzzy** ] [ @deterministic ] [ @abstract ] *Type* *TemplateIdentifier*

["(" *TemplateFormalParList* ")"] [ **modifies** *TemplateRef* ] ":=" *BaseTemplateBody*

NOTE: The optional restriction part is covered by clause 15.8.

***Semantic Description***

Global templates shall be defined in the module definitions part. Local templates shall be defined in testcases, functions, altsteps or statement blocks. Both global and local templates shall adhere to the scoping rules specified in clause 5.

Both global and local templates can be parameterized. The actual parameters of a template can include values and templates. The rules for formal and actual parameter lists shall be followed as defined in clause 5.2.

Both global and local templates are initialized at the place of their declaration. This means, all template fields which are not affected by parameterization shall receive a value or matching mechanism. Template fields affected by parameterization are initialized at the time of template use.

If functions are used for the initialization of module parameters, it is strongly advised to adhere to the rules defined in clause 16.1.4. Not following these rules may cause non-deterministic test executions.

At the time of their use (e.g. in communication operations **send**, **receive**, **call**, **getcall,** etc.), it is allowed to change template fields by in-line modified templates, to pass in values via value parameters as well as to pass in templates via template parameters. The effects of these changes on the values of the template fields do not persist in the template subsequent to the corresponding communication event.

***Restrictions***

In addition to restrictions in clause 15, the following restrictions apply:

a) The dot notation such as *myTemplateId.fieldId* shall not be used to set or retrieve values in templates in communication events. The "->" symbol shall be used for this purpose (see clause 23).

b) Restrictions on referencing elements of templates or template fields are described in clause 15.6.

c) There exist a number of restrictions on the functions used in expressions when specifying templates or template fields; these are specified in clause 16.1.4.

d) After completing initialization, global and local templates that do not contain the @abstract modifier shall be fully initialized.

 NOTE: Initialization of templates without parameters is completed at the place of their declaration. Initialization of parameterized templates is completed at the time of their use.

***Examples***

 // The template

 **template** MyMessageType mw\_myTemplate (**integer** p\_myFormalParam):=

 {

 field1 := p\_myFormalParam,

 field2 := **pattern** "abc\*xyz",

 field3 := **true**

 }

 // could be used as follows

 pco1.**receive**(mw\_myTemplate(123));

## 15.5 Modified templates

In cases where small changes are needed to specify a new template, it is possible to specify a modified template. A modified template specifies modifications to particular fields of the original template, either directly or indirectly. As well as creating explicitly named modified templates, TTCN‑3 allows the definition of in-line modified templates.

***Syntactical Structure***

Global or local modified template:

**template** [*restriction*] [ **@fuzzy** ] [ @deterministic ] [ @abstract ] *Type* *TemplateIdentifier*

["(" *TemplateFormalParList* ")"] **modifies** *TemplateRef* ":=" *BaseTemplateBody*

NOTE 1: The optional restriction part is covered by clause 15.8.

In-line modified template:

**modifies** *BaseTemplateBody* ":=" *BaseTemplateBody*

***Semantic Description***

The **modifies** keyword denotes the parent template from which the new modified template shall be derived. This parent template may be either an original template or a modified template or also a matching mechanism.

In case that the *BaseTemplateBody* on the right hand side of the modified template contains references to variables, these are evaluated before the modification and any uninitialized fields or elements in these variables are treated as unspecified, i.e. as if specified with the not used symbol "‑" for the following modification algorithm.

The modifications occur in a linked fashion, eventually tracing back to the original template:

a) In case of templates, template fields or list elements of simple types, **union** and **enumerated** types, the matching mechanism specified in the modified template is simply replacing its corresponding content in its parent.

b) For templates, template fields and elements of **record** and **set** types, if a **record** or **set** field and its corresponding matching mechanism is specified in the modified template, then the specified matching mechanism replaces the one specified in the corresponding field of the parent template. If a **record** or **set** field or its corresponding matching mechanism is – implicitly or explicitly by using the not used symbol "‑" - left unspecified in the modified template, then the matching mechanism in the corresponding field of the parent template shall be used. When the field to be modified is nested within a template field which is a structured field itself, no other field of the structured field is changed apart from the explicitly denoted one(s).

c) For templates, template fields and elements of **record of** and **set of** types, the above rules specified for **records** and **sets** apply with the following deviations:

* if the value list notation is used, only the number of elements listed in the modified template is inherited from the parent (i.e. the list is truncated at the last element of the list notation in the modified template);
* when individual values of a modified template or a modified template field of **record of** or **set of** type wished to be changed, and only in these cases, the index assignment notation may also be used, where the left hand side of the assignment is the index of the element to be altered.

In case of **record of** and **set of** types first apply rule (c) to the complete structure (e.g. truncation) than apply further rules for the remaining individual type structure elements (see example 3).

Formal value or template parameters of modified templates inherit the default value or respectively template of the corresponding parameter of their parent templates only, if this is denoted by the dash (don't change) symbol at the place of the parameters' default value or respectively template.

Modified templates may also be restricted. Template restrictions are specified in clause 15.8.

A modified template may also be declared fuzzy using the **@fuzzy** modifier.

NOTE 2: If a fuzzy modified template modifies a non-fuzzy unparameterized template, the inherited fields before modification will be the same for every evaluation of the fuzzy template.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5, the following restrictions apply:

a) A modified template shall not refer to itself, either directly or indirectly, i.e. recursive derivation is not allowed.

b) If a base template has a formal parameter list, the following rules apply to all modified templates derived from that base template, whether or not they are derived in one or several modification steps:

1) the derived template shall not omit parameters and change types or names of parameters defined at any of the modification steps between the base template and the actual modified template;

2) a template parameter restriction of a derived template specified at any of the modification steps between the base template and the actual modified template can be changed to a stricter one (see clause 15.8);

3) a derived template can have additional (appended) parameters if wished;

4) if the dash (don't change) symbol is used at the place of a default value or default template, the corresponding parameter of the parent template shall have a valid default value or default template, either assigned directly or inherited. If not, this shall cause an error.

c) Restrictions on referencing elements of templates or template fields are described in clause 15.6: for modified templates the rules for the left hand side of assignments apply.

d) Limitations on template restrictions described in clause 15.8 shall apply.

e) After completing initialization, modified global and local templates that do not contain the @abstract modifier shall be fully initialized.

NOTE: Initialization of modified templates without parameters is completed at the place of their declaration. Initialization of modified parameterized templates is completed at the time of their use.

***Examples***

EXAMPLE 1: Modifying record templates (non-embedded case)

 // Modifying records

 **type record** MyRecordType

 {

 **integer** field1 **optional**,

 **charstring** field2,

 **boolean** field3

 }

 **template** MyRecordType m\_myRecTemplate1 :=

 {

 field1 := 123,

 field2 := "A string",

 field3 := **true**

 }

 // then writing

 **template** MyRecordType m\_myRecTemplate2 **modifies** m\_myRecTemplate1 :=

 {

 field1 := **omit,** // field1 is optional but present in m\_myTemplate1

 field2 := "A modified string"

 // field3 is unchanged

 }

 // is the same as writing

// **template** MyRecordType m\_myRecTemplate2 :=

 // {

 // field1 := **omit**,

 // field2 := "A modified string",

 // field3 := **true**

 // }

 **template** MyRecordType m\_myRecTemplate3 **modifies** m\_myRecTemplate1 := {**omit,** "A modified string"}

 //field3 is implicitly left unchanged;

 //m\_myRecTemplate3 has the same content as m\_myRecTemplate2

 **template** MyRecordType m\_myRecTemplate4 **modifies** m\_myRecTemplate1 := {**omit,**"A modified string",-}

 //field3 is explicitly left unchanged;

 //m\_myRecTemplate4 has the same content as m\_myRecTemplate2 and m\_myRecTemplate3

EXAMPLE 2: Modifying record of templates (non-embedded case)

 **type record of integer** MyRecordOfType;

 **template** MyRecordOfType m\_myBaseTemplate := { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

 **template** MyRecordOfType m\_myRecOfTemplate1 **modifies** m\_myBaseTemplate :=

 { -, -, 3, 2, -, -, -, -, -, - };
 // m\_myRecOfTemplate1 contains { 0, 1, 3, 2, 4, 5, 6, 7, 8, 9 }

 **template** MyRecordOfType m\_myRecOfTemplate2 **modifies** m\_myBaseTemplate := { -, -, 3, 2 };
 // m\_myRecOfTemplate2 replaces m\_myBaseTemplate with: { 0, 1, 3, 2 };

 // elements 5 to 10 of m\_myBaseTemplate are truncated

 **template** MyRecordOfType m\_myRecOfTemplate3 **modifies** m\_myBaseTemplate := { [2] := 3, [3] := 2 };
 // m\_myRecOfTemplate3 has the same content as m\_myMod1Template: { 0, 1, 3, 2, 4, 5, 6, 7, 8, 9 }

EXAMPLE 3: Modifying embedded record and record of templates

 //Modifying a record embedded in a record of

 **type record of record** {
 integer a,
 integer b
 } MyListType

 **template** MyListType mw\_myBaseListTemplate := { ?, { a := 1, b := 2 }, ?, { a := 3, b := 4 } }

 **template** MyListType mw\_myListTemplate1 **modifies** mw\_myBaseListTemplate := { [1] := { a := 42 } }
 //Content of field ″a″ of the second element is modified,

 //the content of mw\_myListTemplate1 is: { ?, { a := 42, b := 2 }, ?, { a := 3, b := 4 } }

 **template** MyListType mw\_myListTemplate2 **modifies** mw\_myBaseListTemplate := { -, { a := 42 } ,- }
 //Content of field ″a″ of the second element is modified, and the

 //record of is truncated after the third element: { ?, { a := 42, b := 2 }, ? }

EXAMPLE 4: Modified in-line template

 // Given

 **template** MyRecordType m\_setup :=

 {

 field1 := 75,

 field2 := "abc",

 field3 := **true**

 }

 // Could be used to define an in-line modified template of Setup

 // pco1.**send** (**modifies** m\_setup := {field1:= 76});

EXAMPLE 5: Modified parameterized template

 // Given

 **template** MyRecordType m\_myTemplate1(**integer** p\_myPar):=

 {

 field1 := p\_myPar,

 field2 := "A string",

 field3 := **true**

 }

 // then a modification could be

 **template** MyRecordType m\_myTemplate2(**integer** p\_myPar) **modifies** m\_myRecTemplate1 :=

 // field1 is parameterized in m\_myTemplate1 and remains also parameterized in m\_myTemplate2

 {

 field2 := "A modified string"

 }

EXAMPLE 6: Default values of modified parameterized templates

 // Given

 **template** MyRecordType m\_myTemplate11 (**integer** p\_int := 5 ):=

 // p\_int has the default value 5

 {

 field1 := p\_int,

 field2 := "A string",

 field3 := **true**

 }

 // then possible template modifications are

 **template** MyRecordType m\_myTemplate12(**integer** p\_int) **modifies** m\_myTemplate11 :=

 // p\_int had a default value in m\_myTemplate11 but has none in this template

 {

 field2 := "B string"

 }

 **template** MyRecordType m\_myTemplate13(**integer** p\_int := 0) **modifies** m\_myTemplate12 := { }

 // p\_int has the default value 0

 // no change is made to the template's content, but only to the default value of p\_int

 **template** MyRecordType m\_myTemplate14(**integer** p\_int := - ) **modifies** m\_myTemplate13 :=

 // p\_int inherits the default value 0 from its parent m\_myTemplate13

 {

 field2 := "C string"

 }

 **template** MyRecordType m\_myTemplate15(**integer** p\_int := - ) **modifies** m\_myTemplate14 :=

 // p\_int inherits the default value 0 from m\_myTemplate13 via m\_myTemplate14

 {

 field2 := "D string"

 }

 **template** MyRecordType m\_myTemplate16(**integer** p\_int) **modifies** m\_myTemplate15 := { }

 // p\_int has no default value; no change in the template's content

 **template** MyRecordType m\_myTemplate17(**integer** p\_int := - ) **modifies** m\_myTemplate16 :=

 // causes an error as p\_int has no default value in the parent template m\_myTemplate16

 {

 field2 := "E string"

 }

EXAMPLE 7: Modifies with variables

 **var template** MyRecordType v\_modification :=

 {

 field2 := "G string" // field1/field3 are uninitialized

 }

 **var template** MyRecordType v\_myTemplate := **modifies** m\_myTemplate1(5) := v\_modification;

 // no modification of field1/field3

 // results in { field1 := 5, field2 := "G string", field3 := **true** }

## A.1.5 TTCN‑3 terminals

### A.1.5.0 General

TTCN‑3 terminal symbols and reserved words are listed in tables A.2 and A.3.

Table A.: List of TTCN‑3 special terminal symbols

|  |  |
| --- | --- |
| Begin/end block symbols | **{ }** |
| Begin/end list symbols | **( )**  |
| Element specifier symbols | **[ ]** |
| Range symbol | **..** |
| Line and block comments | **/\* \*/**  **//** |
| Statement separator symbol | **;** |
| Arithmetic operator symbols | **+ / - \*** |
| Concatenation operator symbol | **&** |
| Relational operator symbols | **!= == >= <= < >** |
| Shift operator symbols | **<< >>** |
| Rotate operator symbols | **<@ @>** |
| String enclosure symbols | **" '**  |
| Wildcard/matching symbols | **? \***  |
| Assignment symbol | **:=**  |
| Communication operation assignment  | **->** |
| Bitstring, hexstring and Octetstring values | **B H O**  |
| Float exponent | **E** |
| List element separator symbol | **,** |
| Field reference | **.** |
| Decoded field reference | **=>** |

The predefined function identifiers defined in table 15 and described in annex C shall also be treated as reserved words.

Table A.: List of TTCN‑3 terminals which are reserved words

|  |  |  |  |
| --- | --- | --- | --- |
| **action****activate****address****alive****all****alt****altstep****and****and4b****any****anytype****bitstring****boolean****break****case****call****catch****char****charstring****check****clear****complement****component****connect****const** **continue****control****create****deactivate****decmatch****default****disconnect****display****do****done****else****encode****enumerated****error****except****exception****execute****extends****extension****external** | **fail****false****float****for****friend****from****function****getverdict****getcall****getreply****goto****group****halt****hexstring****if****ifpresent****import****in****inconc****infinity****inout****integer****interleave****kill****killed****label****language****length****log****map****match****message****mixed****mod****modifies****module****modulepar****mtc** | **noblock****none****not****not\_a\_number****not4b****nowait****null****octetstring****of****omit****on****optional****or****or4b****out****override****param****pass****pattern****permutation****port****present****private****procedure****public****raise****read****receive****record****recursive****rem****repeat****reply****return****running****runs** | **select****self****send****sender****set****setencode****setverdict****signature****start****stop****subset****superset****system****template****testcase****timeout****timer****to****trigger****true****type****union****universal****unmap****value****valueof****var****variant****verdicttype****while****with****xor****xor4b** |

The TTCN‑3 terminals listed in table A.3 shall not be used as identifiers in a TTCN‑3 module. These terminals shall be written in all lowercase letters.

Additionally, there are special TTCN-3 terminals consisting of an @-symbol, directly followed by an identifier. These terminals shall also be written in all lowercase letters.

NOTE: These terminals can be used in combination with the @-symbol, which results in a specific semantics for the annotated language element. They can also be used like any other identifier without any special meaning.

Table A.: List of TTCN‑3 terminals which are modifiers

|  |  |  |  |
| --- | --- | --- | --- |
| **@abstract****@control** | **@decoded****@default****@deterministic**  | **@fuzzy****@index**  | **@lazy****@local** **@nocase** |

Table A.: List of TTCN‑3 terminals which are reserved words in extension packages

|  |  |  |  |
| --- | --- | --- | --- |
| **apply****assert****at****configuration****conjunct****cont****delta****disjunct****duration****finished** | **history****implies****inv****mode****notinv****now****onentry****onexit** | **par****prev****realtime****seq****setstate****static****stepsize****stream** | **timestamp****until****values****wait** |

The TTCN‑3 terminals listed in table A.5 are used as keywords inside the TTCN-3 extension packages. Using these terminals in the code is not recommended as it might lead to issues in the future.

These terminals shall be written in all lowercase letters.

#### A.1.6.1.3 Template definitions

TemplateDef ::= [TemplateKeyword](#TTemplateKeyword) [[TemplateRestriction](#TTemplateRestriction)] [[FuzzyModifier](#TFuzzyModifier) [DeterministicModifier]]

 [AbstractModifier] [BaseTemplate](#TBaseTemplate) [[DerivedDef](#TDerivedDef)] [AssignmentChar](#TAssignmentChar) Base[TemplateBody](#TTemplateBody)

BaseTemplate ::= ([Type](#TType) | [Signature](#TSignature)) [Identifier](#TIdentifier) ["(" [TemplateOrValueFormalParList](#TTemplateOrValueFormalParList)

 ")"]

TemplateKeyword ::= "template"

DerivedDef ::= [ModifiesKeyword](#TModifiesKeyword) [ExtendedIdentifier](#TExtendedIdentifier)

ModifiesKeyword ::= "modifies"

TemplateOrValueFormalParList ::= [TemplateOrValueFormalPar](#TTemplateOrValueFormalPar) {"," [TemplateOrValueFormalPar](#TTemplateOrValueFormalPar)}

TemplateOrValueFormalPar ::= [FormalValuePar](#TFormalValuePar) | [FormalTemplatePar](#TFormalTemplatePar)

/\* STATIC SEMANTICS - FormalValuePar shall resolve to an in parameter \*/

TemplateBody ::= [DerivedTemplateBody](#TDerivedTemplateBody) | [BaseTemplateBody](#TBaseTemplateBody)

BaseTemplateBody ::= ([SimpleSpec](#TSimpleSpec) |

 [FieldSpecList](#TFieldSpecList) |

 [ArrayValueOrAttrib](#TArrayValueOrAttrib)

 ) [[ExtraMatchingAttributes](#TExtraMatchingAttributes)]

/\* STATIC SEMANTICS - Within BaseTeplateBody the ArrayValueOrAttrib can be used for array, record, record of and set of types. \*/

SimpleSpec ::= ([SingleExpression](#TSingleExpression) ["&" [SimpleTemplateSpec](#TSimpleTemplateSpec)]) | [SimpleTemplateSpec](#TSimpleTemplateSpec)

SimpleTemplateSpec ::= [SingleTemplateExpression](#TSingleTemplateExpression) ["&" [SimpleSpec](#TSimpleSpec)]

SingleTemplateExpression ::= [MatchingSymbol](#TMatchingSymbol) |

 ({[TemplateRefWithParList](#TTemplateRefWithParList) [[ExtendedFieldReference](#TExtendedFieldReference)]) |

 [ExtendedIdentifier](#TExtendedIdentifier) [EnumTemplateExtension](#TEnumTemplateExtension)

/\*\* STATIC Semantics: ExtendedIdentifier shall refer to an enumerated value with associated value \*/

EnumTemplateExtension ::= "(" ([BaseTemplateBody](#TTemplateBody) | [Range](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7709_v3.docx#TRange)) {"," ([BaseTemplateBody](#TTemplateBody) | [Range](file:///C%3A%5CUsers%5Cethgry%5CAppData%5CLocal%5CMicrosoft%5CWindows%5CTemporary%20Internet%20Files%5CContent.IE5%5C05PWRTCH%5CCR7709_v3.docx#TRange)) } ")"

/\*\* STATIC Semantics: each TemplateBody shall be an integer template template and the limits of each Range an integer value\*/

FieldSpecList ::= "{" [FieldSpec](#TFieldSpec) {"," [FieldSpec](#TFieldSpec)} "}"

FieldSpec ::= [FieldReference](#TFieldReference) [AssignmentChar](#TAssignmentChar) ([TemplateBody](#TTemplateBody) | [Minus](#TMinus))

FieldReference ::= [StructFieldRef](#TStructFieldRef) |

 [ArrayOrBitRef](#TArrayOrBitRef) |

 [ParRef](#TParRef)

StructFieldRef ::= [Identifier](#TIdentifier) |

 [PredefinedType](#TPredefinedType) |

 [TypeReference](#TTypeReference)

/\* STATIC SEMANTICS - PredefinedType and TypeReference shall be used for anytype value notation only. PredefinedType shall not be AnyTypeKeyword.\*/

ParRef ::= [Identifier](#TIdentifier)

/\* STATIC SEMANTICS - Identifier in ParRef shall be a formal parameter identifier from the associated signature definition \*/

ArrayOrBitRef ::= "[" [FieldOrBitNumber](#TFieldOrBitNumber) "]"

/\* STATIC SEMANTICS - ArrayRef shall be optionally used for array types and TTCN-3 record of and set of. The same notation can be used for a Bit reference inside an TTCN-3 charstring, universal charstring, bitstring, octetstring and hexstring type \*/

FieldOrBitNumber ::= [SingleExpression](#TSingleExpression)

/\* STATIC SEMANTICS - SingleExpression will resolve to a value of integer type \*/

ArrayValueOrAttrib ::= "{" [[ArrayElementSpecList](#TArrayElementSpecList)] "}"

ArrayElementSpecList ::= [ArrayElementSpec](#TArrayElementSpec) {"," [ArrayElementSpec](#TArrayElementSpec)}

ArrayElementSpec ::= [Minus](#TMinus) |

 [PermutationMatch](#TPermutationMatch) |

 [TemplateBody](#TTemplateBody)

MatchingSymbol ::= [Complement](#TComplement) |

 ([AnyValue](#TAnyValue) [[WildcardLengthMatch](#TWildcardLengthMatch)]) |

 ([AnyOrOmit](#TAnyOrOmit) [[WildcardLengthMatch](#TWildcardLengthMatch)]) |

 [ListOfTemplates](#TListOfTemplates) |

 [Range](#TRange) |

 [BitStringMatch](#TBitStringMatch) |

 [HexStringMatch](#THexStringMatch) |

 [OctetStringMatch](#TOctetStringMatch) |

 [CharStringMatch](#TCharStringMatch) |

 [SubsetMatch](#TSubsetMatch) |

 [SupersetMatch](#TSupersetMatch) |

 [DecodedContentMatch](#TDecodedContentMatch)

DecodedContentMatch ::= [DecodedMatchKeyword](#TDecodedMatchKeyword) ["(" [[Expression](#TExpression)] ")"] [TemplateInstance](#TTemplateInstance)

DecodedMatchKeyword ::= "decmatch"

/\* STATIC SEMANTIC – WildcardLengthMatch shall be used when MatchingSymbol is used in fractions of a concatenated string or list (see clause 15.11) and shall not be used in other cases. In this case, the Complement, ListOfTemplates, Range, BitStringMatch, HexStringMatch, OctetStringMatch, CharStringMatch, SubsetMatch and SupersetMatch productions shall not be used. \*/

ExtraMatchingAttributes ::= [StringLength](#TStringLength) |

 [IfPresentKeyword](#TIfPresentKeyword) |

 ([StringLength](#TStringLength) [IfPresentKeyword](#TIfPresentKeyword))

BitStringMatch ::= "'" {[BinOrMatch](#TBinOrMatch)} "'" "B"

BinOrMatch ::= [Bin](#TBin) |

 [AnyValue](#TAnyValue) |

 [AnyOrOmit](#TAnyOrOmit)

HexStringMatch ::= "'" {[HexOrMatch](#THexOrMatch)} "'" "H"

HexOrMatch ::= [Hex](#THex) |

 [AnyValue](#TAnyValue) |

 [AnyOrOmit](#TAnyOrOmit)

OctetStringMatch ::= "'" {[OctOrMatch](#TOctOrMatch)} "'" "O"

OctOrMatch ::= [Oct](#TOct) |

 [AnyValue](#TAnyValue) |

 [AnyOrOmit](#TAnyOrOmit)

CharStringMatch ::= [PatternKeyword](#TPatternKeyword) [[CaseInsenModifier](#TCaseInsenModifier)] [PatternParticle](#TPatternParticle) {"&" [PatternParticle](#TPatternParticle)}

PatternParticle ::= [Pattern](#TPattern) | [ReferencedValue](#TReferencedValue)

PatternKeyword ::= "pattern"

Pattern ::= """ {[PatternElement](#TPatternElement)} """

PatternElement ::= (("\" ("?" | "\*" | "\" | "[" | "]" | "{" | "}" |

 """ | "|" | "(" | ")" | "#" | "+" | "d" |

 "w" | "t" | "n" | "r" | "s" | "b"

 )) | ("?" | "\*" | "\" | "|" | "+"

 ) | ("[" ["^"] [{[PatternClassChar](#TPatternClassChar) ["-"

 [PatternClassChar](#TPatternClassChar)]}]

 "]") |

 ("{" ["\"] [ReferencedValue](#TReferencedValue) "}") | ("\" "N" "{"

 ([ReferencedValue](#TReferencedValue) |

 [Type](#TType)) "}") |

 (""" """) |

 ("(" [PatternElement](#TPatternElement) ")") |

 ("#" ([Num](#TNum) |

 ("(" [Number](#TNumber) "," [[Number](#TNumber)] ")") |

 ("(" "," [Number](#TNumber) ")") |

 ("(" [","] ")") [Num](#TNum) ")"

 ))

 ) | [PatternChar](#TPatternChar)

PatternChar ::= [NonSpecialPatternChar](#TNonSpecialPatternChar) | [PatternQuadruple](#TPatternQuadruple)

/\* STATIC SEMANTICS: Characters "?", "\*", "\", "[", "]", "{", "}", """, "|", "(", ")", "#", "+", "d", "^", "N" have special semantics – they are metacharacters for the definition of pattern elements – only if they follow the BNF as defined above, if not they are interpreted like normal characters \*/

NonSpecialPatternChar ::= [Char](#TChar)

PatternClassChar ::= [NonSpecialPatternClassChar](#TNonSpecialPatternClassChar) |

 [PatternQuadruple](#TPatternQuadruple) |

 "\" [EscapedPatternClassChar](#TEscapedPatternClassChar)

NonSpecialPatternClassChar ::= [Char](#TChar)

/\* STATIC SEMANTICS: Characters "[", "-", "^", "]", "\", "q", ","have special semantics – they are metacharacters for the definition of pattern class characters – only if they follow the BNF as defined above, if not they are interpreted like normal characters \*/

EscapedPatternClassChar ::= "[" | "-" | "^" | "]"

PatternQuadruple ::= "\" "q" "(" [Number](#TNumber) "," [Number](#TNumber) "," [Number](#TNumber) ","

 [Number](#TNumber) ")"

Complement ::= [ComplementKeyword](#TComplementKeyword) [ListOfTemplates](#TListOfTemplates)

ComplementKeyword ::= "complement"

ListOfTemplates ::= "(" [TemplateListItem](#TTemplateListItem) {"," [TemplateListItem](#TTemplateListItem)} ")"

TemplateListItem ::= [TemplateBody](#TTemplateBody) | [AllElementsFrom](#TAllElementsFrom)

AllElementsFrom ::= [AllKeyword](#TAllKeyword) [FromKeyword](#TFromKeyword) [TemplateBody](#TTemplateBody)

SubsetMatch ::= [SubsetKeyword](#TSubsetKeyword) [ListOfTemplates](#TListOfTemplates)

SubsetKeyword ::= "subset"

SupersetMatch ::= [SupersetKeyword](#TSupersetKeyword) [ListOfTemplates](#TListOfTemplates)

SupersetKeyword ::= "superset"

PermutationMatch ::= [PermutationKeyword](#TPermutationKeyword) [ListOfTemplates](#TListOfTemplates)

/\* STATIC SEMANTICS: Restrictions on the content of TemplateBody within the ListOfTemplates are given in clause B.1.3.3. \*/

PermutationKeyword ::= "permutation"

AnyValue ::= "?"

AnyOrOmit ::= "\*"

WildcardLengthMatch ::= [LengthKeyword](#TLengthKeyword) "(" [SingleExpression](#TSingleExpression) ")"

/\* STATIC SEMANTICS: SingleExpression shall evaluate to type integer \*/

IfPresentKeyword ::= "ifpresent"

PresentKeyword ::= "present"

Range ::= "(" [Bound](#TBound) ".." [Bound](#TBound) ")"

Bound ::= (["!"] [SingleExpression](#TSingleExpression)) | ([[Minus](#TMinus)] [InfinityKeyword](#TInfinityKeyword))

/\* STATIC SEMANTICS - Bounds shall evaluate to types integer, charstring, universal charstring or float. In case they evaluate to types charstring or universal charstring, the string length shall be 1. infinity as lower bound and –infinity as upper bound are allowed for float types only. \*/

InfinityKeyword ::= "infinity"

ActualParAssignment ::= [Identifier](#TIdentifier) ":=" [TemplateInstance](#TInLineTemplate)

/\* STATIC SEMANTICS – if a value parameter is used, an in-line template shall evaluate to a value \*/ TemplateRefWithParList ::= [ExtendedIdentifier](#TExtendedIdentifier) [[ActualParList](#TActualParList)]

TemplateInstance ::= [([Type](#TType) | [Signature](#TSignature)) [Colon](#TColon)]

[TemplateBody](#TTemplateBody)

DerivedTemplateBody ::= [ModifiesKeyword](#TModifiesKeyword) [BaseTemplateBody](#TBaseTemplateBody) [AssignmentChar](#TAssignmentChar) [BaseTemplateBody](#TBaseTemplateBody)

ActualParList ::= "(" [([ActualPar](#TActualPar) {"," [ActualPar](#TActualPar) })

 {"," [ActualParAssignment](#TActualParAssignment)} |

 ([ActualParAssignment](#TActualParAssignment) {"," [ActualParAssignment](#TActualParAssignment)})]

 ")"

ActualPar ::= [TemplateInstance](#TInLineTemplate) | [Minus](#TMinus)

/\* STATIC SEMANTICS - When the corresponding formal parameter is not of template type the TemplateInstance production shall resolve to one or more SingleExpressions \*/

TemplateOps ::= [MatchOp](#TMatchOp) | [ValueofOp](#TValueofOp)

MatchOp ::= [MatchKeyword](#TMatchKeyword) "(" [Expression](#TExpression) "," [TemplateInstance](#TInLineTemplate) ")"

MatchKeyword ::= "match"

ValueofOp ::= [ValueofKeyword](#TValueofKeyword) "(" [TemplateInstance](#TTemplateInstance)")"

ValueofKeyword ::= "valueof"

### A.1.6.9 Miscellaneous productions

Dot ::= "."

Minus ::= "-"

SemiColon ::= ";"

Colon ::= ":"

Underscore ::= "\_"

AssignmentChar ::= ":="

IndexModifier ::= "@index"

DeterministicModifier ::= "@deterministic"

LazyModifier ::= "@lazy"

FuzzyModifier ::= "@fuzzy"

CaseInsenModifier ::= "@nocase"

DecodedModifier ::= "@decoded"

DefaultModifier ::= "@default"

ControlModifier ::= "@control"

AbstractModifier ::= "@abstract"

# G.15 Partially initialized global and local templates

Previous versions of the present document (up to and including V4.11.1) allowed global and local templates without the @abstract modifier to be partially initialized after their initialization has been completed. This feature is deprecated and may be fully removed in a future edition of the present document.