## A.1.6 TTCN-3 syntax BNF productions

### A.1.6.0 TTCN-3 module

TTCN3Module ::= [TTCN3ModuleKeyword](#TTTCN3ModuleKeyword) [ModuleId](#TModuleId) "{" [[ModuleDefinitionsList](#TModuleDefinitionsList)]

"}" [[WithStatement](#TWithStatement)] [[SemiColon](#TSemiColon)]

TTCN3ModuleKeyword ::= "module"

ModuleId ::= [Identifier](#TIdentifier) [[LanguageSpec](#TLanguageSpec)]

LanguageSpec ::= [LanguageKeyword](#TLanguageKeyword) [FreeText](#TFreeText) {"," [FreeText](#TFreeText)}

LanguageKeyword ::= "language"

### A.1.6.1 Module definitions part

#### A.1.6.1.0 General

ModuleDefinitionsList ::= {[ModuleDefinition](#TModuleDefinition) [[SemiColon](#TSemiColon)]}+

ModuleDefinition ::= (([[Visibility](#TVisibility)] ([TypeDef](#TTypeDef) |

[ConstDef](#TConstDef) |

[TemplateDef](#TTemplateDef) |

[ModuleParDef](#TModuleParDef) |

[FunctionDef](#TFunctionDef) |

[SignatureDef](#TSignatureDef) |

[TestcaseDef](#TTestcaseDef) |

[AltstepDef](#TAltstepDef) |

[ImportDef](#TImportDef) |

[ExtFunctionDef](#TExtFunctionDef) |

[ModuleControlDef](#TModuleControlDef)

)) |

(["public"] [GroupDef](#TGroupDef)) |

(["private"] [FriendModuleDef](#TFriendModuleDef))

) [[WithStatement](#TWithStatement)]

Visibility ::= "public" |

"friend" |

"private"

#### A.1.6.1.1 Typedef definitions

TypeDef ::= [TypeDefKeyword](#TTypeDefKeyword) [TypeDefBody](#TTypeDefBody)

TypeDefBody ::= [StructuredTypeDef](#TStructuredTypeDef) | [SubTypeDef](#TSubTypeDef)

TypeDefKeyword ::= "type"

StructuredTypeDef ::= [RecordDef](#TRecordDef) |

[UnionDef](#TUnionDef) |

[SetDef](#TSetDef) |

[RecordOfDef](#TRecordOfDef) |

[SetOfDef](#TSetOfDef) |

[EnumDef](#TEnumDef) |

[PortDef](#TPortDef) |

[ComponentDef](#TComponentDef) |

[MapDef](#TMapDef)

MapDef ::= [MapKeyword](#TMapKeyword) [FromKeyword](#TFromKeyword) [Type](#TType) [ToKeyword](#TToKeyword) ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef) | [MapDef](#TMapDef))

RecordDef ::= [RecordKeyword](#TRecordKeyword) [StructDefBody](#TStructDefBody)

RecordKeyword ::= "record"

StructDefBody ::= [IdentifierOrAddr](#TIdentifierOrAddr) "{" [[StructFieldDef](#TStructFieldDef)

{"," [StructFieldDef](#TStructFieldDef)}]

"}"

StructFieldDef ::= ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef)) [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)] [[SubTypeSpec](#TSubTypeSpec)]

[[OptionalKeyword](#TOptionalKeyword)]

NestedTypeDef ::= [NestedRecordDef](#TNestedRecordDef) |

[NestedUnionDef](#TNestedUnionDef) |

[NestedSetDef](#TNestedSetDef) |

[NestedRecordOfDef](#TNestedRecordOfDef) |

[NestedSetOfDef](#TNestedSetOfDef) |

[NestedEnumDef](#TNestedEnumDef)

NestedRecordDef ::= [RecordKeyword](#TRecordKeyword) "{" [[StructFieldDef](#TStructFieldDef) {"," [StructFieldDef](#TStructFieldDef)}]

"}"

NestedUnionDef ::= [UnionKeyword](#TUnionKeyword) "{" [UnionFieldDef](#TUnionFieldDef) {"," [UnionFieldDef](#TUnionFieldDef)}

"}"

NestedSetDef ::= [SetKeyword](#TSetKeyword) "{" [[StructFieldDef](#TStructFieldDef) {"," [StructFieldDef](#TStructFieldDef)}]

"}"

NestedRecordOfDef ::= [RecordKeyword](#TRecordKeyword) [[StringLength](#TStringLength)] [OfKeyword](#TOfKeyword) ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef))

NestedSetOfDef ::= [SetKeyword](#TSetKeyword) [[StringLength](#TStringLength)] [OfKeyword](#TOfKeyword) ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef))

NestedEnumDef ::= [EnumKeyword](#TEnumKeyword) "{" [EnumerationList](#TEnumerationList) "}"

OptionalKeyword ::= "optional"

UnionDef ::= [UnionKeyword](#TUnionKeyword) [UnionDefBody](#TUnionDefBody)

UnionKeyword ::= "union"

UnionDefBody ::= [IdentifierOrAddr](#TIdentifierOrAddr) "{" [UnionFieldDef](#TUnionFieldDef) {"," [UnionFieldDef](#TUnionFieldDef)} "}"

UnionFieldDef ::= [[DefaultModifier](#TDefaultModifier)] ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef)) [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)] [[SubTypeSpec](#TSubTypeSpec)]

/\* STATIC SEMANTICS - at most one *UnionFieldDef* of *UnionDefBody* or *NestedUnionDef* shall contain a *DefaultModifier* \*/

SetDef ::= [SetKeyword](#TSetKeyword) [StructDefBody](#TStructDefBody)

SetKeyword ::= "set"

RecordOfDef ::= [RecordKeyword](#TRecordKeyword) [[StringLength](#TStringLength)] [OfKeyword](#TOfKeyword) [StructOfDefBody](#TStructOfDefBody)

OfKeyword ::= "of"

StructOfDefBody ::= ([Type](#TType) | [NestedTypeDef](#TNestedTypeDef)) [IdentifierOrAddr](#TIdentifierOrAddr) [[SubTypeSpec](#TSubTypeSpec)]

SetOfDef ::= [SetKeyword](#TSetKeyword) [[StringLength](#TStringLength)] [OfKeyword](#TOfKeyword) [StructOfDefBody](#TStructOfDefBody)

EnumDef ::= [EnumKeyword](#TEnumKeyword) [IdentifierOrAddr](#TIdentifierOrAddr) "{" [EnumerationList](#TEnumerationList) "}"

EnumKeyword ::= "enumerated"

EnumerationList ::= [Enumeration](#TEnumeration) {"," [Enumeration](#TEnumeration)}

Enumeration ::= [Identifier](#TIdentifier) ["(" [IntegerValueOrRange](#TEnumValueOrRange) {"," [IntegerValueOrRange](#TEnumValueOrRange) } ")"]

IntegerValueOrRange ::= [IntegerValue](#TIntegerValue) [".." [IntegerValue](#TIntegerValue)]

IntegerValue ::= [[Minus](#TMinus)] [Number](#TNumber)

SubTypeDef ::= [Type](#TType) [IdentifierOrAddr](#TIdentifierOrAddr) [[ArrayDef](#TArrayDef)] [[SubTypeSpec](#TSubTypeSpec)]

SubTypeSpec ::= [AllowedValuesSpec](#TAllowedValuesSpec) [[StringLength](#TStringLength)] | [StringLength](#TStringLength)

/\* STATIC SEMANTICS - AllowedValues shall be of the same type as the field being subtyped \*/

AllowedValuesSpec ::= "(" (([TemplateOrRange](#TTemplateOrRange) {"," [TemplateOrRange](#TTemplateOrRange)}) |

[CharStringMatch](#TCharStringMatch)) ")"

TemplateOrRange ::= [RangeDef](#TRangeDef) |

[TemplateBody](#TTemplateBody) |

[Type](#TType)

/\* STATIC SEMANTICS - *RangeDef* production shall only be used with integer, charstring, universal charstring or float based types \*/

/\* STATIC SEMANTICS - When subtyping charstring or universal charstring range and values shall not be mixed in the same SubTypeSpec \*/

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

StringLength ::= [LengthKeyword](#TLengthKeyword) "(" [SingleExpression](#TSingleExpression) [".."([SingleExpression](#TSingleExpression) | [InfinityKeyword](#TInfinityKeyword))]  
 ")"

/\* STATIC SEMANTICS - *StringLength* shall only be used with String types or to limit set of and record of. *SingleExpression* and Bound shall evaluate to non-negative integer values (in case of Bound including infinity) \*/

LengthKeyword ::= "length"

PortDef ::= [PortKeyword](#TPortKeyword) [PortDefBody](#TPortDefBody)

PortDefBody ::= [Identifier](#TIdentifier) [PortDefAttribs](#TPortDefAttribs)

PortKeyword ::= "port"

PortDefAttribs ::= [MessageAttribs](#TMessageAttribs) |

[ProcedureAttribs](#TProcedureAttribs) |

[MixedAttribs](#TMixedAttribs)

MessageAttribs ::= [MessageKeyword](#TMessageKeyword) "{" {([AddressDecl](#TAddressDecl) |

[MessageList](#TMessageList) |

[ConfigParamDef](#TConfigParamDef)

) [[SemiColon](#TSemiColon)]}+ "}"

ConfigParamDef ::= [MapParamDef](#TMapParamDef) | [UnmapParamDef](#TUnmapParamDef)

MapParamDef ::= [MapKeyword](#TMapKeyword) [ParamKeyword](#TParamKeyword) "(" [FormalValuePar](#TFormalValuePar) {"," [FormalValuePar](#TFormalValuePar)} ")"

UnmapParamDef ::= [UnmapKeyword](#TUnmapKeyword) [ParamKeyword](#TParamKeyword) "(" [FormalValuePar](#TFormalValuePar) {"," [FormalValuePar](#TFormalValuePar)} ")"

AddressDecl ::= [AddressKeyword](#TAddressKeyword) [Type](#TType)

MessageList ::= [Direction](#TDirection) [AllOrTypeList](#TAllOrTypeList)

Direction ::= [InParKeyword](#TInParKeyword) |

[OutParKeyword](#TOutParKeyword) |

[InOutParKeyword](#TInOutParKeyword)

MessageKeyword ::= "message"

AllOrTypeList ::= [AllKeyword](#TAllKeyword) | [TypeList](#TTypeList)

/\* NOTE: The use of *AllKeyword* in port definitions is deprecated \*/

AllKeyword ::= "all"

TypeList ::= [Type](#TType) {"," [Type](#TType)}

ProcedureAttribs ::= [ProcedureKeyword](#TProcedureKeyword) "{" {([AddressDecl](#TAddressDecl) |

[ProcedureList](#TProcedureList) |

[ConfigParamDef](#TConfigParamDef)

) [[SemiColon](#TSemiColon)]}+ "}"

ProcedureKeyword ::= "procedure"

ProcedureList ::= [Direction](#TDirection) [AllOrSignatureList](#TAllOrSignatureList)

AllOrSignatureList ::= [AllKeyword](#TAllKeyword) | [SignatureList](#TSignatureList)

SignatureList ::= [Signature](#TSignature) {"," [Signature](#TSignature)}

MixedAttribs ::= [MixedKeyword](#TMixedKeyword) "{" {([AddressDecl](#TAddressDecl) |

[MixedList](#TMixedList) |

[ConfigParamDef](#TConfigParamDef)

) [[SemiColon](#TSemiColon)]}+ "}"

MixedKeyword ::= "mixed"

MixedList ::= [Direction](#TDirection) [ProcOrTypeList](#TProcOrTypeList)

ProcOrTypeList ::= [AllKeyword](#TAllKeyword) | ([ProcOrType](#TProcOrType) {"," [ProcOrType](#TProcOrType)})

ProcOrType ::= [Signature](#TSignature) | [Type](#TType)

ComponentDef ::= [ComponentKeyword](#TComponentKeyword) [Identifier](#TIdentifier) [[ExtendsKeyword](#TExtendsKeyword) [ComponentType](#TComponentType)

{"," [ComponentType](#TComponentType)}] "{" [[ComponentDefList](#TComponentDefList)] "}"

ComponentKeyword ::= "component"

ExtendsKeyword ::= "extends"

ComponentType ::= [ExtendedIdentifier](#TExtendedIdentifier)

ComponentDefList ::= {[ComponentElementDef](#TComponentElementDef) [[WithStatement](#TWithStatement)] [[SemiColon](#TSemiColon)]}

ComponentElementDef ::= [PortInstance](#TPortInstance) |

[VarInstance](#TVarInstance) |

[TimerInstance](#TTimerInstance) |

[ConstDef](#TConstDef) |

[TemplateDef](#TTemplateDef)

PortInstance ::= [PortKeyword](#TPortKeyword) [ExtendedIdentifier](#TExtendedIdentifier) [PortElement](#TPortElement) {"," [PortElement](#TPortElement)}

PortElement ::= [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)]

#### A.1.6.1.2 Constant definitions

ConstDef ::= [ConstKeyword](#TConstKeyword) [Type](#TType) [ConstList](#TConstList)

ConstList ::= [SingleConstDef](#TSingleConstDef) {"," [SingleConstDef](#TSingleConstDef)}

SingleConstDef ::= [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)] [AssignmentChar](#TAssignmentChar) [ConstantExpression](#TConstantExpression)

ConstKeyword ::= "const"

#### A.1.6.1.3 Template definitions

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

[[AbstractModifier](#TAbstractModifier)] [BaseTemplate](#TBaseTemplate) [[DerivedDef](#TDerivedDef)] [AssignmentChar](#TAssignmentChar) [BaseTemplateBody](#TBaseTemplateBody)

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

TemplateKeyword ::= "template"

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

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 *BaseTemplateBody* 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:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/05PWRTCH/CR7709_v3.docx#TRange)) {"," ([BaseTemplateBody](#TTemplateBody) | [Range](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/05PWRTCH/CR7709_v3.docx#TRange)) } ")"

/\* STATIC SEMANTICS - each *TemplateBody* shall be an integer 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) |

[IndexRef](#TIndexRef) |

[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 \*/

IndexRef ::= "[" [SingleExpression](#TFieldOrBitNumber) "]"

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

/\* 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 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) | [OmitOp](#TOmitOp) | [PresentOp](#TPresentOp)

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

MatchKeyword ::= "match"

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

ValueofKeyword ::= "valueof"

OmitOp ::= [OmitKeyword](#TValueofKeyword) "(" [TemplateInstance](#TTemplateInstance)")"

PresentOp ::= [PresentKeyword](#TValueofKeyword) "(" [TemplateInstance](#TTemplateInstance)")"

#### A.1.6.1.4 Function definitions

FunctionDef ::= [FunctionKeyword](#TFunctionKeyword) [ [DeterministicModifier](#TDeterministicModifier) | [ControlModifier](#TControlModifier) ]

[IdentifierOrControl](#TIdentifierOrControl)

"(" [[FunctionFormalParList](#TFunctionFormalParList)] ")" [[RunsOnSpec](#TRunsOnSpec)] [[MtcSpec](#TMtcSpec)]

[[SystemSpec](#TSystemSpec)] [[ReturnType](#TReturnType)] [StatementBlock](#TStatementBlock)

FunctionKeyword ::= "function"

FunctionFormalParList ::= [FunctionFormalPar](#TFunctionFormalPar) {"," [FunctionFormalPar](#TFunctionFormalPar)}

FunctionFormalPar ::= [FormalValuePar](#TFormalValuePar) |

[FormalTemplatePar](#TFormalTemplatePar)

ReturnType ::= [ReturnKeyword](#TReturnKeyword) [[TemplateModifier](#TTemplateModifier)] [Type](#TType) [[ArrayDef](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TArrayDef)]

ReturnKeyword ::= "return"

RunsOnSpec ::= [RunsKeyword](#TRunsKeyword) [OnKeyword](#TOnKeyword) [ComponentType](#TComponentType)

RunsKeyword ::= "runs"

OnKeyword ::= "on"

MtcSpec ::= [MTCKeyword](#TMTCKeyword) [ComponentType](#TComponentType)

MTCKeyword ::= "mtc"

StatementBlock ::= "{" [[FunctionDefOrStatementList](#TFunctionStatementList)] "}"

FunctionDefOrStatementList ::= {( [FunctionBodyDef](#TFunctionBodyDef) | [FunctionStatement](#TFunctionStatement)) [[SemiColon](#TSemiColon)]}+

FunctionBodyDef ::= ([FunctionLocalDef](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/C3T541B1/CR7624-2.docx#TFunctionLocalDef) | [FunctionLocalInst](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/C3T541B1/CR7624-2.docx#TFunctionLocalInst)) [[WithStatement](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/C3T541B1/CR7624-2.docx#TWithStatement)]

FunctionLocalInst ::= [VarInstance](#TVarInstance) | [TimerInstance](#TTimerInstance)

FunctionLocalDef ::= [ConstDef](#TConstDef) | [TemplateDef](#TTemplateDef)

FunctionStatement ::= [ConfigurationStatements](#TConfigurationStatements) |

[TimerStatements](#TTimerStatements) |

[CommunicationStatements](#TCommunicationStatements) |

[BasicStatements](#TBasicStatements) |

[BehaviourStatements](#TBehaviourStatements) |

[SetLocalVerdict](#TSetLocalVerdict) |

[SUTStatements](#TSUTStatements) |

[TestcaseOperation](#TTestcaseOperation)

FunctionInstance ::= [FunctionRef](#TFunctionRef) [ "(" [[ActualParList](#TActualParList)] ")" ]

/\* STATIC SEMANTICS – the part is only optional if the *FunctionRef* uses the *ControlKeyword* and the referenced control function has no formal parameters \*/

FunctionRef ::= [[Identifier](#TIdentifier) [Dot](#TDot)] ([Identifier](#TIdentifier) | [PreDefFunctionIdentifier](#TPreDefFunctionIdentifier) | [ControlKeyword](#TControlKeyword) )

PreDefFunctionIdentifier ::= [Identifier](#TIdentifier) [[CaseInsenModifier](#TCaseInsenModifier)]

/\* STATIC SEMANTICS - The *Identifier* shall be one of the pre-defined TTCN-3 function identifiers from Annex C of ES 201 873-1. *CaseInsenModifier* shall be present only if *Identifier* is "regexp". \*/

/\* STATIC SEMANTICS – if a value parameter is used, an in-line template shall evaluate to a value \*/

#### A.1.6.1.5 Signature definitions

SignatureDef ::= [SignatureKeyword](#TSignatureKeyword) [Identifier](#TIdentifier) "(" [[SignatureFormalParList](#TSignatureFormalParList)]

")" [[ReturnType](#TReturnType) | [NoBlockKeyword](#TNoBlockKeyword)] [[ExceptionSpec](#TExceptionSpec)]

SignatureKeyword ::= "signature"

SignatureFormalParList ::= [FormalValuePar](#TFormalValuePar) {"," [FormalValuePar](#TFormalValuePar)}

ExceptionSpec ::= [ExceptionKeyword](#TExceptionKeyword) "(" [TypeList](#TTypeList) ")"

ExceptionKeyword ::= "exception"

Signature ::= [ExtendedIdentifier](#TExtendedIdentifier)

NoBlockKeyword ::= "noblock"

#### A.1.6.1.6 Testcase definitions

TestcaseDef ::= [TestcaseKeyword](#TTestcaseKeyword) [Identifier](#TIdentifier) "(" [[TemplateOrValueFormalParList](#TTemplateOrValueFormalParList)]

")" [ConfigSpec](#TConfigSpec) [StatementBlock](#TStatementBlock)

TestcaseKeyword ::= "testcase"

ConfigSpec ::= [[RunsOnSpec](#TRunsOnSpec)] [[SystemSpec](#TSystemSpec)]

SystemSpec ::= [SystemKeyword](#TSystemKeyword) [ComponentType](#TComponentType)

SystemKeyword ::= "system"

TestcaseInstance ::= [ExecuteKeyword](#TExecuteKeyword) "(" [ExtendedIdentifier](#TExtendedIdentifier) "(" [[ActualParList](#TActualParList)]

")" ["," ([Expression](#TExpression) | [Minus](#TMinus)) ["," [SingleExpression](#TSingleExpression)]]

")"

ExecuteKeyword ::= "execute"

#### A.1.6.1.7 Altstep definitions

AltstepDef ::= [AltstepKeyword](#TAltstepKeyword) [ [ControlModifier](#TControlModifier) ] [ [InterleavedKeyword](#TInterleavedKeyword) ] [Identifier](#TIdentifier)

"(" [[FunctionFormalParList](#TFunctionFormalParList)] ")" [[RunsOnSpec](#TRunsOnSpec)] [[MtcSpec](#TMtcSpec)] [[SystemSpec](#TSystemSpec)]

"{" [AltstepLocalDefList](#TAltstepLocalDefList) [AltGuardList](#TAltGuardList) "}"

AltstepKeyword ::= "altstep"

AltstepLocalDefList ::= {[AltstepLocalDef](#TAltstepLocalDef) [[WithStatement](#TWithStatement)] [[SemiColon](#TSemiColon)]}

AltstepLocalDef ::= [VarInstance](#TVarInstance) |

[TimerInstance](#TTimerInstance) |

[ConstDef](#TConstDef) |

[TemplateDef](#TTemplateDef)

AltstepInstance ::= [ExtendedIdentifier](#TExtendedIdentifier) "(" [[ActualParList](#TActualParList)] ")"

#### A.1.6.1.8 Import definitions

ImportDef ::= [ImportKeyword](#TImportKeyword) [ImportFromSpec](#TImportFromSpec) [[PortRedirectSymbol](#TPortRedirectSymbol) [Identifier](#TIdentifier)]

([AllWithExcepts](#TAllWithExcepts) | ("{" [ImportSpec](#TImportSpec) "}"))

ImportKeyword ::= "import"

AllWithExcepts ::= [AllKeyword](#TAllKeyword) [[ExceptsDef](#TExceptsDef)]

ExceptsDef ::= [ExceptKeyword](#TExceptKeyword) "{" [ExceptSpec](#TExceptSpec) "}"

ExceptKeyword ::= "except"

ExceptSpec ::= {[ExceptElement](#TExceptElement) [[SemiColon](#TSemiColon)]}

ExceptElement ::= [ExceptGroupSpec](#TExceptGroupSpec) |

[ExceptTypeDefSpec](#TExceptTypeDefSpec) |

[ExceptTemplateSpec](#TExceptTemplateSpec) |

[ExceptConstSpec](#TExceptConstSpec) |

[ExceptTestcaseSpec](#TExceptTestcaseSpec) |

[ExceptAltstepSpec](#TExceptAltstepSpec) |

[ExceptFunctionSpec](#TExceptFunctionSpec) |

[ExceptSignatureSpec](#TExceptSignatureSpec) |

[ExceptModuleParSpec](#TExceptModuleParSpec)

ExceptGroupSpec ::= [GroupKeyword](#TGroupKeyword) ([QualifiedIdentifierList](#TQualifiedIdentifierList) | [AllKeyword](#TAllKeyword))

IdentifierListOrAll ::= [IdentifierList](#TIdentifierList) | [AllKeyword](#TAllKeyword)

TypeIdListOrAll ::= [TypeIdentifierList](#TTypeIdentifierList) | [AllKeyword](#TAllKeyword)

FuncIdListOrAll ::= [FuncIdentifierList](#TFuncIdentifierList) | [AllKeyword](#TAllKeyword)

ExceptTypeDefSpec ::= [TypeDefKeyword](#TTypeDefKeyword) [TypeIdListOrAll](#TTypeIdListOrAll)

ExceptTemplateSpec ::= [TemplateKeyword](#TTemplateKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ExceptConstSpec ::= [ConstKeyword](#TConstKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ExceptTestcaseSpec ::= [TestcaseKeyword](#TTestcaseKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ExceptAltstepSpec ::= [AltstepKeyword](#TAltstepKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ExceptFunctionSpec ::= [FunctionKeyword](#TFunctionKeyword) [FuncIdListOrAll](#TFuncIdListOrAll)

ExceptSignatureSpec ::= [SignatureKeyword](#TSignatureKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ExceptModuleParSpec ::= [ModuleParKeyword](#TModuleParKeyword) [IdentifierListOrAll](#TIdentifierListOrAll)

ImportSpec ::= {[ImportElement](#TImportElement) [[SemiColon](#TSemiColon)]}

ImportElement ::= [ImportGroupSpec](#TImportGroupSpec) |

[ImportTypeDefSpec](#TImportTypeDefSpec) |

[ImportTemplateSpec](#TImportTemplateSpec) |

[ImportConstSpec](#TImportConstSpec) |

[ImportTestcaseSpec](#TImportTestcaseSpec) |

[ImportAltstepSpec](#TImportAltstepSpec) |

[ImportFunctionSpec](#TImportFunctionSpec) |

[ImportSignatureSpec](#TImportSignatureSpec) |

[ImportModuleParSpec](#TImportModuleParSpec) |

[ImportImportSpec](#TImportImportSpec)

ImportFromSpec ::= [FromKeyword](#TFromKeyword) [ModuleId](#TModuleId)

ImportGroupSpec ::= [GroupKeyword](#TGroupKeyword) ([GroupRefListWithExcept](#TGroupRefListWithExcept) | [AllGroupsWithExcept](#TAllGroupsWithExcept))

GroupRefListWithExcept ::= [QualifiedIdentifierWithExcept](#TQualifiedIdentifierWithExcept) {"," [QualifiedIdentifierWithExcept](#TQualifiedIdentifierWithExcept)}

AllGroupsWithExcept ::= [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword) [QualifiedIdentifierList](#TQualifiedIdentifierList)]

QualifiedIdentifierWithExcept ::= [QualifiedIdentifier](#TQualifiedIdentifier) [[ExceptsDef](#TExceptsDef)]

IdentifierListOrAllWithExcept ::= [IdentifierList](#TIdentifierList) | [AllWithExcept](#TAllWithExcept)

TypeIdListOrAllWithExcept ::= [TypeIdentifierList](#TTypeIdentifierList) | [AllTypesExcept](#TAllTypesExcept)

FuncIdListOrAllWithExcept ::= [FuncIdentifierList](#TFuncIdentifierList) | [AllFunctionsExcept](#TAllFunctionsExcept)

ImportTypeDefSpec ::= [TypeDefKeyword](#TTypeDefKeyword) [TypeIdListOrAllWithExcept](#TTypeIdListOrAllWithExcept)

AllWithExcept ::= [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword) [IdentifierList](#TIdentifierList)]

AllTypesExcept ::= [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword) [TypeIdentifierList](#TTypeIdentifierList)]

AllFunctionsExcept ::= [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword) [FuncIdentifierList](#TFuncIdentifierList)]

ImportTemplateSpec ::= [TemplateKeyword](#TTemplateKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportConstSpec ::= [ConstKeyword](#TConstKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportAltstepSpec ::= [AltstepKeyword](#TAltstepKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportTestcaseSpec ::= [TestcaseKeyword](#TTestcaseKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportFunctionSpec ::= [FunctionKeyword](#TFunctionKeyword) [FuncIdListOrAllWithExcept](#TFuncIdListOrAllWithExcept)

ImportSignatureSpec ::= [SignatureKeyword](#TSignatureKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportModuleParSpec ::= [ModuleParKeyword](#TModuleParKeyword) [IdentifierListOrAllWithExcept](#TIdentifierListOrAllWithExcept)

ImportImportSpec ::= [ImportKeyword](#TImportKeyword) [AllKeyword](#TAllKeyword)

TypeIdentifierList ::= [IdentifierOrAddr](#TIdentifierOrAddr) {"," [IdentifierOrAddr](#TIdentifierOrAddr) }

IdentifierOrAddr ::= [Identifier](#TIdentifier) | [AddressKeyword](#TAddressKeyword)

FuncIdentifierList ::= [IdentifierOrControl](#TIdentifierOrControl) {"," [IdentifierOrControl](#TIdentifierOrControl) }

IdentifierOrControl ::= [Identifier](#TIdentifier) | [ControlKeyword](#TControlKeyword)

#### A.1.6.1.9 Group definitions

GroupDef ::= [GroupKeyword](#TGroupKeyword) [Identifier](#TIdentifier) "{" [[ModuleDefinitionsList](#TModuleDefinitionsList)] "}"

GroupKeyword ::= "group"

#### A.1.6.1.10 External function definitions

ExtFunctionDef ::= [ExtKeyword](#TExtKeyword) [FunctionKeyword](#TFunctionKeyword) [[DeterministicModifier](#TDeterministicModifier) | [ControlModifier](#TControlModifier)]

[Identifier](#TIdentifier) "(" [[FunctionFormalParList](#TFunctionFormalParList)] ")" [[ReturnType](#TReturnType)]

ExtKeyword ::= "external"

#### A.1.6.1.11 Void

#### A.1.6.1.12 Module parameter definitions

ModuleParDef ::= [ModuleParKeyword](#TModuleParKeyword) ([ModulePar](#TModulePar) | ("{" [MultitypedModuleParList](#TMultitypedModuleParList) "}"))

ModuleParKeyword ::= "modulepar"

MultitypedModuleParList ::= {[ModulePar](#TModulePar) [[SemiColon](#TSemiColon)]}

ModulePar ::= [ [TemplateModifier](#TTemplateModifier) ] [Type](#TType) [ModuleParList](#TModuleParList)

ModuleParList ::= [Identifier](#TIdentifier) [[AssignmentChar](#TAssignmentChar) [TemplateBody](#TConstantExpression)] {","

[Identifier](#TIdentifier) [[AssignmentChar](#TAssignmentChar) [TemplateBody](#TConstantExpression)]}

#### A.1.6.1.13 Friend module definitions

FriendModuleDef ::= "friend" "module" [IdentifierList](#TIdentifierList) [[SemiColon](#TSemiColon)]

### A.1.6.2 Module control function

ModuleControlDef ::= [ControlKeyword](#TControlKeyword) [StatementBlock](#TStatementBlock)

ControlKeyword ::= "control"

### A.1.6.3 Local definitions

#### A.1.6.3.1 Variable instantiation

VarInstance ::= [VarKeyword](#TVarKeyword) (([([LazyModifier](#TLazyModifier) | [FuzzyModifier](#TFuzzyModifier)) [[DeterministicModifier](#TDeterministicModifier)] ]

[Type](#TType) [VarList](#TVarList)) | ( [TemplateModifier](#TTemplateModifier)

[([LazyModifier](#TLazyModifier) | [FuzzyModifier](#TFuzzyModifier)) [[DeterministicModifier](#TDeterministicModifier)] ]

[Type](#TType) [TempVarList](#TTempVarList)) )

VarList ::= [SingleVarInstance](#TSingleVarInstance) {"," [SingleVarInstance](#TSingleVarInstance)}

SingleVarInstance ::= [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)] [[AssignmentChar](#TAssignmentChar) [Expression](#TExpression)]

VarKeyword ::= "var"

TempVarList ::= [SingleTempVarInstance](#TSingleTempVarInstance) {"," [SingleTempVarInstance](#TSingleTempVarInstance)}

SingleTempVarInstance ::= [Identifier](#TIdentifier) [[ArrayDef](#TArrayDef)] [[AssignmentChar](#TAssignmentChar) [TemplateBody](#TTemplateBody)]

ValueRef ::= [Identifier](#TIdentifier) [[ExtendedFieldReference](#TExtendedFieldReference)]

#### A.1.6.3.2 Timer instantiation

TimerInstance ::= [TimerKeyword](#TTimerKeyword) [VarList](#TVarList)

TimerKeyword ::= "timer"

ArrayIdentifierRef ::= [Identifier](#TIdentifier) {[IndexRef](#TIndexRef)}

### A.1.6.4 Operations

#### A.1.6.4.1 Component operations

ConfigurationStatements ::= [ConnectStatement](#TConnectStatement) |

[MapStatement](#TMapStatement) |

[DisconnectStatement](#TDisconnectStatement) |

[UnmapStatement](#TUnmapStatement) |

[ [NoDefaultModifier](#TNoDefaultModifier) ] [DoneStatement](#TDoneStatement) |

[ [NoDefaultModifier](#TNoDefaultModifier) ] [KilledStatement](#TKilledStatement) |

[StartTCStatement](#TStartTCStatement) |

[StopTCStatement](#TStopTCStatement) |

[KillTCStatement](#TKillTCStatement) |

[SetEncodeStatement](#TSetEncodeStatement)

ConfigurationOps ::= [CreateOp](#TCreateOp) |

[SelfOp](#TSelfOp) |

[SystemKeyword](#TSystemKeyword) |

[MTCKeyword](#TMTCKeyword) |

[RunningOp](#TRunningOp) |

[AliveOp](#TAliveOp)

CreateOp ::= [ComponentType](#TComponentType) [Dot](#TDot) [CreateKeyword](#TCreateKeyword) ["(" ([SingleExpression](#TSingleExpression) |

[Minus](#TMinus)) ["," [SingleExpression](#TSingleExpression)] ")"] [[AliveKeyword](#TAliveKeyword)]

SelfOp ::= "self"

DoneStatement ::= [ComponentOrAny](#TComponentOrAny) [Dot](#TDot) [DoneKeyword](#TDoneKeyword) [ [PortRedirectSymbol](#TPortRedirectSymbol)

[ [ValueStoreSpec](#TValueStoreSpec) ] [ [IndexSpec](#TIndexSpec) ] ]

/\* STATIC SEMANTICS – If PortRedirectSymbol is present, at least one of *ValueStoreSpec* and IndexSpec shall be present \*/

ComponentOrAny ::= |

([AnyKeyword](#TAnyKeyword) ([ComponentKeyword](#TComponentKeyword) | [FromKeyword](#TFromKeyword) [ValueRef](#TValueRef))) |

([AllKeyword](#TAllKeyword) [ComponentKeyword](#TComponentKeyword))

ValueStoreSpec ::= [ValueKeyword](#TValueKeyword) [ValueRef](#TValueRef)

IndexAssignment ::= [PortRedirectSymbol](#TPortRedirectSymbol) [IndexSpec](#TIndexSpec)

IndexSpec ::= [IndexModifier](#TIndexModifier) [ValueStoreSpec](#TValueStoreSpec)

KilledStatement ::= [ComponentOrAny](#TComponentOrAny) [Dot](#TDot) [KilledKeyword](#TKilledKeyword) [ [PortRedirectSymbol](#TPortRedirectSymbol)

[ [ValueStoreSpec](#TValueStoreSpec) ] [ [IndexSpec](#TIndexSpec)] ]

/\* STATIC SEMANTICS – If PortRedirectSymbol is present, at least one of *ValueStoreSpec* and IndexSpec shall be present \*/

DoneKeyword ::= "done"

KilledKeyword ::= "killed"

RunningOp ::= [ComponentOrAny](#TComponentOrAny) [Dot](#TDot) [RunningKeyword](#TRunningKeyword) [[IndexAssignment](#TIndexAssignment)]

RunningKeyword ::= "running"

AliveOp ::= [ComponentOrAny](#TComponentOrAny) [Dot](#TDot) [AliveKeyword](#TAliveKeyword) [[IndexAssignment](#TIndexAssignment)]

CreateKeyword ::= "create"

AliveKeyword ::= "alive"

ConnectStatement ::= [ConnectKeyword](#TConnectKeyword) [SingleConnectionSpec](#TSingleConnectionSpec)

ConnectKeyword ::= "connect"

SingleConnectionSpec ::= "(" [PortRef](#TPortRef) "," [PortRef](#TPortRef) ")"

PortRef ::= [ComponentRef](#TComponentRef) [Colon](#TColon) [ArrayIdentifierRef](#TArrayIdentifierRef)

ComponentRef ::= |

[SystemKeyword](#TSystemKeyword) |

[SelfOp](#TSelfOp) |

[MTCKeyword](#TMTCKeyword)

DisconnectStatement ::= [DisconnectKeyword](#TDisconnectKeyword) [[SingleConnectionSpec](#TSingleConnectionSpec) |

[AllConnectionsSpec](#TAllConnectionsSpec) |

[AllPortsSpec](#TAllPortsSpec) |

[AllCompsAllPortsSpec](#TAllCompsAllPortsSpec)

]

AllConnectionsSpec ::= "(" [PortRef](#TPortRef) ")"

AllPortsSpec ::= "(" [ComponentRef](#TComponentRef) ":" [AllKeyword](#TAllKeyword) [PortKeyword](#TPortKeyword) ")"

AllCompsAllPortsSpec ::= "(" [AllKeyword](#TAllKeyword) [ComponentKeyword](#TComponentKeyword) ":" [AllKeyword](#TAllKeyword)

[PortKeyword](#TPortKeyword) ")"

DisconnectKeyword ::= "disconnect"

MapStatement ::= [MapKeyword](#TMapKeyword) [SingleConnectionSpec](#TSingleConnectionSpec) [[ParamClause](#TParamClause)]

ParamClause ::= [ParamKeyword](#TParamKeyword) [ActualParList](#TActualParList)

MapKeyword ::= "map"

UnmapStatement ::= [UnmapKeyword](#TUnmapKeyword) [[SingleConnectionSpec](#TSingleConnectionSpec) [[ParamClause](#TParamClause)] |

[AllConnectionsSpec](#TAllConnectionsSpec) [[ParamClause](#TParamClause)] |

[AllPortsSpec](#TAllPortsSpec) |

[AllCompsAllPortsSpec](#TAllCompsAllPortsSpec) |

"(" [ValueRef](#TComponentRef) "," [SingleExpression](#TSingleExpression) ")"

]

UnmapKeyword ::= "unmap"

StartTCStatement ::= [Dot](#TDot) [StartKeyword](#TStartKeyword)

"(" ([FunctionInstance](#TFunctionInstance) | [AltstepInstance](#TAltstepInstance)) ")"

StartKeyword ::= "start"

StopTCStatement ::= [StopKeyword](#TStopKeyword) | ([ComponentReferenceOrLiteral](#TComponentReferenceOrLiteral) | [AllKeyword](#TAllKeyword)

[ComponentKeyword](#TComponentKeyword)) [Dot](#TDot) [StopKeyword](#TStopKeyword)

ComponentReferenceOrLiteral ::= |

[MTCKeyword](#TMTCKeyword) |

[SelfOp](#TSelfOp)

KillTCStatement ::= [KillKeyword](#TKillKeyword) | (([ComponentReferenceOrLiteral](#TComponentReferenceOrLiteral) |

[AllKeyword](#TAllKeyword) [ComponentKeyword](#TComponentKeyword)) [Dot](#TDot) [KillKeyword](#TKillKeyword))

ObjectReference ::= [ValueRef](#TComponentRef) | [FunctionInstance](#TFunctionInstance)

KillKeyword ::= "kill"

SetEncodeStatement ::= ( [SingleExpression](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TSingleExpression) | ( [AllKeyword](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TAllKeyword) [PortKeyword](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TPortKeyword) ) | [SelfOp](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TSelfOp) ) | "."  
 [SetEncodeKeyword](#TSetEncodeKeyword) "(" [Type](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TType) "," [SingleExpression](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/08RUYH79/CR7714-v1.docx#TSingleExpression) ")"

SetEncodeKeyword ::= "setencode"

#### A.1.6.4.2 Port operations

CommunicationStatements ::= [SendStatement](#TSendStatement) |

[CallStatement](#TCallStatement) |

[ReplyStatement](#TReplyStatement) |

[RaiseStatement](#TRaiseStatement) |

[ [NoDefaultModifier](#TNoDefaultModifier) ] [ReceivingStatement](#TReceiveStatement) |

[ClearStatement](#TClearStatement) |

[StartStatement](#TStartStatement) |

[StopStatement](#TStopStatement) |

[HaltStatement](#THaltStatement) |

[CheckStateStatement](#TCheckStateStatement)

ReceivingStatement ::= [ReceiveStatement](#TReceiveStatement) |

[TriggerStatement](#TTriggerStatement) |

[GetCallStatement](#TGetCallStatement) |

[GetReplyStatement](#TGetReplyStatement) |

[CatchStatement](#TCatchStatement) |

[CheckStatement](#TCheckStatement)

SendStatement ::= [Dot](#TDot) [PortSendOp](#TPortSendOp)

PortSendOp ::= [SendOpKeyword](#TSendOpKeyword) "(" [TemplateInstance](#TTemplateInstance)")" [[ToClause](#TToClause)]

SendOpKeyword ::= "send"

ToClause ::= [ToKeyword](#TToKeyword) ([TemplateInstance](#TTemplateInstance)|

[AddressRefList](#TAddressRefList) |

[AllKeyword](#TAllKeyword) [ComponentKeyword](#TComponentKeyword)

)

AddressRefList ::= "(" [TemplateInstance](#TTemplateInstance){"," [TemplateInstance](#TTemplateInstance)} ")"

ToKeyword ::= "to"

CallStatement ::= [Dot](#TDot) [PortCallOp](#TPortCallOp) [[PortCallBody](#TPortCallBody)]

PortCallOp ::= [CallOpKeyword](#TCallOpKeyword) "(" [CallParameters](#TCallParameters) ")" [[ToClause](#TToClause)]

CallOpKeyword ::= "call"

CallParameters ::= [TemplateInstance](#TTemplateInstance) ["," [CallTimerValue](#TCallTimerValue)]

CallTimerValue ::= [Expression](#TExpression) | [NowaitKeyword](#TNowaitKeyword)

NowaitKeyword ::= "nowait"

PortCallBody ::= "{" [CallBodyStatementList](#TCallBodyStatementList) "}"

CallBodyStatementList ::= {[CallBodyStatement](#TCallBodyStatement) [[SemiColon](#TSemiColon)]}+

CallBodyStatement ::= [CallBodyGuard](#TCallBodyGuard) [StatementBlock](#TStatementBlock)

CallBodyGuard ::= [AltGuardChar](#TAltGuardChar) [CallBodyOps](#TCallBodyOps)

CallBodyOps ::= [GetReplyStatement](#TGetReplyStatement) | [CatchStatement](#TCatchStatement)

ReplyStatement ::= [Dot](#TDot) [PortReplyOp](#TPortReplyOp)

PortReplyOp ::= [ReplyKeyword](#TReplyKeyword) "(" [TemplateInstance](#TTemplateInstance) [[ReplyValue](#TReplyValue)] ")" [[ToClause](#TToClause)]

ReplyKeyword ::= "reply"

ReplyValue ::= [ValueKeyword](#TValueKeyword) [TemplateBody](#TTemplateBody)

/\* STATIC SEMANTICS - *TemplateBody* shall be type compatible with the return type. It shall evaluate to a value or template (literal or template instance) conforming to the template(value) restriction. \*/

RaiseStatement ::= [Dot](#TDot) [PortRaiseOp](#TPortRaiseOp)

PortRaiseOp ::= [RaiseKeyword](#TRaiseKeyword) "(" [Signature](#TSignature) "," [TemplateInstance](#TTemplateInstance)")" [[ToClause](#TToClause)]

NoDefaultModifier ::= "@nodefault"

RaiseKeyword ::= "raise"

ReceiveStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortReceiveOp](#TPortReceiveOp)

PortOrAny ::= | ([AnyKeyword](#TAnyKeyword) ([PortKeyword](#TPortKeyword) | [FromKeyword](#TFromKeyword) [ValueRef](#TComponentRef)))

PortReceiveOp ::= [ReceiveOpKeyword](#TReceiveOpKeyword) ["("[TemplateInstance](#TTemplateInstance)")"] [[FromClause](#TFromClause)] [[PortRedirect](#TPortRedirect)]

ReceiveOpKeyword ::= "receive"

FromClause ::= [FromKeyword](#TFromKeyword) ([TemplateInstance](#TTemplateInstance) |

[AddressRefList](#TAddressRefList) |

[AnyKeyword](#TAnyKeyword) [ComponentKeyword](#TComponentKeyword)

)

FromKeyword ::= "from"

PortRedirect ::= [PortRedirectSymbol](#TPortRedirectSymbol) (([ValueSpec](#TValueSpec) [[SenderSpec](#TSenderSpec)] [[IndexSpec](#TIndexSpec)]) |

([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

[IndexSpec](#TIndexSpec)

)

PortRedirectSymbol ::= "->"

ValueSpec ::= [ValueKeyword](#TValueKeyword) ([ValueRef](#TComponentRef) | ("(" [SingleValueSpec](#TSingleValueSpec) {"," [SingleValueSpec](#TSingleValueSpec)} ")"))

SingleValueSpec ::= [ValueRef](#TComponentRef) [[AssignmentChar](#TAssignmentChar) [ [DecodedModifier](#TDecodedModifier) ["(" [[Expression](#TExpression)] ")"] ]

[FieldReference](#TFieldReference) [ExtendedFieldReference](#TExtendedFieldReference)]

/\* STATIC SEMANTICS – *FieldReference* shall not be *ParRef* and *ExtendedFieldReference* shall not be *TypeDefIdentifier*\*/

ValueKeyword ::= "value"

SenderSpec ::= [SenderKeyword](#TSenderKeyword) [ValueRef](#TComponentRef)

SenderKeyword ::= "sender"

TriggerStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortTriggerOp](#TPortTriggerOp)

PortTriggerOp ::= [TriggerOpKeyword](#TTriggerOpKeyword) ["(" [TemplateInstance](#TInLineTemplate)  ")"] [[FromClause](#TFromClause)] [[PortRedirect](#TPortRedirect)]

TriggerOpKeyword ::= "trigger"

GetCallStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortGetCallOp](#TPortGetCallOp)

PortGetCallOp ::= [GetCallOpKeyword](#TGetCallOpKeyword) ["(" [TemplateInstance](#TTemplateInstance) ")"] [[FromClause](#TFromClause)]

[[PortRedirectWithParam](#TPortRedirectWithParam)]

GetCallOpKeyword ::= "getcall"

PortRedirectWithParam ::= [PortRedirectSymbol](#TPortRedirectSymbol) [RedirectWithParamSpec](#TRedirectWithParamSpec)

RedirectWithParamSpec ::= ([ParamSpec](#TParamSpec) [[SenderSpec](#TSenderSpec)] [[IndexSpec](#TIndexSpec)]) |

([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

[IndexSpec](#TIndexSpec)

ParamSpec ::= [ParamKeyword](#TParamKeyword) [ParamAssignmentList](#TParamAssignmentList)

ParamKeyword ::= "param"

ParamAssignmentList ::= "(" ([AssignmentList](#TAssignmentList) | [VariableList](#TVariableList)) ")"

AssignmentList ::= [VariableAssignment](#TVariableAssignment) {"," [VariableAssignment](#TVariableAssignment)}

VariableAssignment ::= [ValueRef](#TComponentRef) [AssignmentChar](#TAssignmentChar) [ [DecodedModifier](#TDecodedModifier) ["(" [Expression](#TExpression)] ")"]

[Identifier](#TIdentifier)

VariableList ::= [VariableEntry](#TVariableEntry) {"," [VariableEntry](#TVariableEntry)}

VariableEntry ::= [ValueRef](#TComponentRef) | [Minus](#TMinus)

GetReplyStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortGetReplyOp](#TPortGetReplyOp)

PortGetReplyOp ::= [GetReplyOpKeyword](#TGetReplyOpKeyword) ["(" [TemplateInstance](#TTemplateInstance) [[ValueMatchSpec](#TValueMatchSpec)]

")"] [[FromClause](#TFromClause)] [[PortRedirectWithValueAndParam](#TPortRedirectWithValueAndParam)]

PortRedirectWithValueAndParam ::= [PortRedirectSymbol](#TPortRedirectSymbol) [RedirectWithValueAndParamSpec](#TRedirectWithValueAndParamSpec)

RedirectWithValueAndParamSpec ::= ([ValueSpec](#TValueSpec) [[ParamSpec](#TParamSpec)] [[SenderSpec](#TSenderSpec)]

[[IndexSpec](#TIndexSpec)]) | [RedirectWithParamSpec](#TRedirectWithParamSpec)

GetReplyOpKeyword ::= "getreply"

ValueMatchSpec ::= [ValueKeyword](#TValueKeyword) [TemplateInstance](#TTemplateInstance)

CheckStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortCheckOp](#TPortCheckOp)

PortCheckOp ::= [CheckOpKeyword](#TCheckOpKeyword) ["(" [CheckParameter](#TCheckParameter) ")"]

CheckOpKeyword ::= "check"

CheckParameter ::= [CheckPortOpsPresent](#TCheckPortOpsPresent) |

[FromClausePresent](#TFromClausePresent) |

[RedirectPresent](#TRedirectPresent)

FromClausePresent ::= [FromClause](#TFromClause) [[PortRedirectSymbol](#TPortRedirectSymbol) (([SenderSpec](#TSenderSpec)

[[IndexSpec](#TIndexSpec)]) |

[IndexSpec](#TIndexSpec))]

RedirectPresent ::= [PortRedirectSymbol](#TPortRedirectSymbol) (([SenderSpec](#TSenderSpec) [[IndexSpec](#TIndexSpec)]) |

[IndexSpec](#TIndexSpec))

CheckPortOpsPresent ::= [PortReceiveOp](#TPortReceiveOp) |

[PortGetCallOp](#TPortGetCallOp) |

[PortGetReplyOp](#TPortGetReplyOp) |

[PortCatchOp](#TPortCatchOp)

CatchStatement ::= [PortOrAny](#TPortOrAny) [Dot](#TDot) [PortCatchOp](#TPortCatchOp)

PortCatchOp ::= [CatchOpKeyword](#TCatchOpKeyword) ["(" [CatchOpParameter](#TCatchOpParameter) ")"] [[FromClause](#TFromClause)] [[PortRedirect](#TPortRedirect)]

CatchOpKeyword ::= "catch"

CatchOpParameter ::= [Signature](#TSignature) [ "," [TemplateInstance](#TTemplateInstance) ] | [TimeoutKeyword](#TTimeoutKeyword)

ClearStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [ClearOpKeyword](#TClearOpKeyword)

PortOrAll ::= | [AllKeyword](#TAllKeyword) [PortKeyword](#TPortKeyword)

ClearOpKeyword ::= "clear"

StartStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [StartKeyword](#TStartKeyword)

StopStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [StopKeyword](#TStopKeyword)

StopKeyword ::= "stop"

HaltStatement ::= [PortOrAll](#TPortOrAll) [Dot](#TDot) [HaltKeyword](#THaltKeyword)

HaltKeyword ::= "halt"

AnyKeyword ::= "any"

CheckStateStatement ::= [PortOrAllAny](#TPortOrAllAny) [Dot](#TDot) [CheckStateKeyword](#TCheckStateKeyword) "(" [SingleExpression](#TSingleExpression) ")"

PortOrAllAny ::= [PortOrAll](#TPortOrAll) | [AnyKeyword](#TAnyKeyword) [PortKeyword](#TPortKeyword)

CheckStateKeyword ::= "checkstate"

#### A.1.6.4.3 Timer operations

TimerStatements ::= [StartTimerStatement](#TStartTimerStatement) |

[StopTimerStatement](#TStopTimerStatement) |

[ [NoDefaultModifier](#TNoDefaultModifier) ] [TimeoutStatement](#TTimeoutStatement)

TimerOps ::= [ReadTimerOp](#TReadTimerOp) | [RunningTimerOp](#TRunningTimerOp)

StartTimerStatement ::= [Dot](#TDot) [StartKeyword](#TStartKeyword) ["(" [Expression](#TExpression) ")"]

StopTimerStatement ::= [TimerRefOrAll](#TTimerRefOrAll) [Dot](#TDot) [StopKeyword](#TStopKeyword)

TimerRefOrAll ::= | [AllKeyword](#TAllKeyword) [TimerKeyword](#TTimerKeyword)

ReadTimerOp ::= [Dot](#TDot) [ReadKeyword](#TReadKeyword)

ReadKeyword ::= "read"

RunningTimerOp ::= [TimerRefOrAny](#TTimerRefOrAny) [Dot](#TDot) [RunningKeyword](#TRunningKeyword) [[IndexAssignment](#TIndexAssignment)]

TimeoutStatement ::= [TimerRefOrAny](#TTimerRefOrAny) [Dot](#TDot) [TimeoutKeyword](#TTimeoutKeyword) [[IndexAssignment](#TIndexAssignment)]

TimerRefOrAny ::= |

([AnyKeyword](#TAnyKeyword) [TimerKeyword](#TTimerKeyword)) |

([AnyKeyword](#TAnyKeyword) [FromKeyword](#TFromKeyword) [Identifier](#TIdentifier))

TimeoutKeyword ::= "timeout"

#### A.1.6.4.4 Testcase operation

TestcaseOperation ::= [TestcaseKeyword](#TTestcaseKeyword) "." [StopKeyword](#TStopKeyword) ["(" { [LogItem](#TLogItem) [","] } ")"]

### A.1.6.5 Type

Type ::= [PredefinedType](#TPredefinedType) | [ReferencedType](#TReferencedType)

PredefinedType ::= [BitStringKeyword](#TBitStringKeyword) |

[BooleanKeyword](#TBooleanKeyword) |

[CharStringKeyword](#TCharStringKeyword) |

[UniversalCharString](#TUniversalCharString) |

[IntegerKeyword](#TIntegerKeyword) |

[OctetStringKeyword](#TOctetStringKeyword) |

[HexStringKeyword](#THexStringKeyword) |

[VerdictTypeKeyword](#TVerdictTypeKeyword) |

[FloatKeyword](#TFloatKeyword) |

[AddressKeyword](#TAddressKeyword) |

[DefaultKeyword](#TDefaultKeyword) |

[AnyTypeKeyword](#TAnyTypeKeyword) |

[TimerKeyword](#TTimerKeyword)

BitStringKeyword ::= "bitstring"

BooleanKeyword ::= "boolean"

IntegerKeyword ::= "integer"

OctetStringKeyword ::= "octetstring"

HexStringKeyword ::= "hexstring"

VerdictTypeKeyword ::= "verdicttype"

FloatKeyword ::= "float"

AddressKeyword ::= "address"

DefaultKeyword ::= "default"

AnyTypeKeyword ::= "anytype"

CharStringKeyword ::= "charstring"

UniversalCharString ::= [UniversalKeyword](#TUniversalKeyword) [CharStringKeyword](#TCharStringKeyword)

UniversalKeyword ::= "universal"

ReferencedType ::= [ExtendedIdentifier](#TExtendedIdentifier) [[ExtendedTypeFieldReference](#TExtendedFieldReference)]

TypeReference ::= [ExtendedIdentifier](#TExtendedIdentifier)

ArrayDef ::= {"[" [SingleExpression](#TSingleExpression) [".." [SingleExpression](#TSingleExpression)] "]"}+

ExtendedTypeFieldReference ::= {([Dot](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TDot) ([Identifier](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TIdentifier) | [PredefinedType](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TPredefinedType) | [FromKeyword](#TFromKeyword) | [ToKeyword](#TToKeyword))) |

("[" [Minus](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TMinus) "]") }+

/\* STATIC SEMANTICS - ArrayBounds will resolve to a non negative value of integer type \*/

### A.1.6.6 Value

Value ::= [PredefinedValue](#TPredefinedValue) | [ReferencedValue](#TReferencedValue)

PredefinedValue ::= [Bstring](#TBstring) |

[BooleanValue](#TBooleanValue) |

[CharStringValue](#TCharStringValue) |

[Number](#TNumber) | /\* IntegerValue \*/

[Ostring](#TOstring) |

[Hstring](#THstring) |

[VerdictTypeValue](#TVerdictTypeValue) |

[FloatValue](#TFloatValue) |

[AddressValue](#TAddressValue) |

[OmitKeyword](#TOmitKeyword)

BooleanValue ::= "true" | "false"

VerdictTypeValue ::= "pass" |

"fail" |

"inconc" |

"none" |

"error"

CharStringValue ::= [Cstring](#TCstring) | [Quadruple](#TQuadruple) | [USIlikeNotation](#TUSIlikeNotation)

Quadruple ::= [CharKeyword](#TCharKeyword) "(" [Number](#TNumber) "," [Number](#TNumber) "," [Number](#TNumber) "," [Number](#TNumber) ")"

USIlikeNotation ::= [CharKeyword](#TCharKeyword) "(" [UIDlike](#TUIDlike) { "," [UIDlike](#TUIDlike) } ")"

UIDlike ::= ("U"|"u") {"+"} {[Hex](#THex)}#(1,8)

CharKeyword ::= "char"

FloatValue ::= [FloatDotNotation](#TFloatDotNotation) |

[FloatENotation](#TFloatENotation) |

[NaNKeyword](#TNaNKeyword)

NaNKeyword ::= "not\_a\_number"

FloatDotNotation ::= [Number](#TNumber) [Dot](#TDot) [DecimalNumber](#TDecimalNumber)

FloatENotation ::= [Number](#TNumber) [[Dot](#TDot) [DecimalNumber](#TDecimalNumber)] [Exponential](#TExponential) [[Minus](#TMinus)] [Number](#TNumber)

Exponential ::= "E"

ReferencedValue ::= ([ExtendedIdentifier](#TExtendedIdentifier) [[ExtendedFieldReference](#TExtendedFieldReference)] ) | [ReferencedEnumValue](#TReferencedEnumValue)

/\* STATIC SEMANTICS – The second option is used only for referencing enumerated values, in all other cases, the first option is used.

ReferencedEnumValue ::= [[ReferencedType](#TReferencedType) [Dot](#TDot)] [Identifier](#TIdentifier) [[ExtendedEnumReference](#TExtendedEnumReference)]

/\* STATIC SEMANTICS - *ExtendedEnumReference* shall be present if and only if *Identifier* refers to an enumerated value with an attached value list \*/

ExtendedEnumReference ::= "(" [IntegerValue](#TIntegerValue) ")"

Number ::= ([NonZeroNum](#TNonZeroNum) {[Num](#TNum)}) | "0"

NonZeroNum ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

DecimalNumber ::= { [Num](#TNum) }+

Num ::= "0" | [NonZeroNum](#TNonZeroNum)

Bstring ::= "'" { [Bin](#TBin) | [BinSpace](#TBinSpace) } "'" "B"

Bin ::= "0" | "1"

Hstring ::= "'" { [Hex](#THex) | [BinSpace](#TBinSpace) } "'" "H"

Hex ::= [Num](#TNum) | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"

Ostring ::= "'" { [Oct](#TOct) | [BinSpace](#TBinSpace) } "'" "O"

Oct ::= [Hex](#THex) [Hex](#THex)

Cstring ::= """ {[Char](#TChar)} """

Char ::= /\* REFERENCE - A character defined by the relevant CharacterString type. For charstring a character from the character set defined in ITU-T T.50. For universal charstring a character from any character set defined in ISO/IEC 10646. \*/

Identifier ::= [Alpha](#TAlpha) {[AlphaNum](#TAlphaNum) | [Underscore](#TUnderscore)}

Alpha ::= [UpperAlpha](#TUpperAlpha) | [LowerAlpha](#TLowerAlpha)

AlphaNum ::= [Alpha](#TAlpha) | [Num](#TNum)

UpperAlpha ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |

"J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |

"S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

LowerAlpha ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |

"j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |

"s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

ExtendedAlphaNum ::= /\* REFERENCE - A graphical character from the BASIC LATIN or from the LATIN-1 SUPPLEMENT character sets defined in ISO/IEC 10646 (characters from char (0,0,0,32) to char (0,0,0,126), from char (0,0,0,161) to char (0,0,0,172) and from char (0,0,0,174) to char (0,0,0,255) \*/

FreeText ::= """ {[ExtendedAlphaNum](#TExtendedAlphaNum)} """

AddressValue ::= "null"

OmitKeyword ::= "omit"

BinSpace ::= " " | "\" [NLChar](#TNLChar)

NLChar ::= /\* REFERENCE - Any sequence of newline characters that constitute a newline by using the following C0 control characters: LF(10), VT(11), FF(12), CR(13) (jointly called newline characters, see clause A.1.5.1) from the character set defined in Recommendation ITU‑T T.50 [4]. \*/

### A.1.6.7 Parameterization

InParKeyword ::= "in"

OutParKeyword ::= "out"

InOutParKeyword ::= "inout"

FormalValuePar ::= [([InParKeyword](#TInParKeyword) | [InOutParKeyword](#TInOutParKeyword) | [OutParKeyword](#TOutParKeyword) )]

[([LazyModifier](#TLazyModifier) | [FuzzyModifier](#TFuzzyModifier)) [[DeterministicModifier](#TDeterministicModifier)] ]

[Type](#TType) [Identifier](#TIdentifier) [[ArrayDef](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TArrayDef)] [":=" ([Expression](#TExpression) | [Minus](#TMinus))]

FormalTemplatePar ::= [([InParKeyword](#TInParKeyword) | [OutParKeyword](#TOutParKeyword) | [InOutParKeyword](#TInOutParKeyword))] [TemplateModifier](#TTemplateModifier)

[([LazyModifier](#TLazyModifier) | [FuzzyModifier](#TFuzzyModifier)) [[DeterministicModifier](#TDeterministicModifier)] ]

[Type](#TType) [Identifier](#TIdentifier) [[ArrayDef](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TArrayDef)] [":=" ([TemplateInstance](#TTemplateInstance) | [Minus](#TMinus)) ]

TemplateModifier ::= [TemplateKeyword](#TTemplateKeyword) | [RestrictedTemplate](#TRestrictedTemplate)

RestrictedTemplate ::= [OmitKeyword](#TOmitKeyword) | ([TemplateKeyword](#TTemplateKeyword) [TemplateRestriction](#TTemplateRestriction))

TemplateRestriction ::= "(" ([OmitKeyword](#TOmitKeyword) |

[ValueKeyword](#TValueKeyword) |

[PresentKeyword](#TPresentKeyword)

) ")"

### A.1.6.8 Statements

#### A.1.6.8.1 With statement

WithStatement ::= [WithKeyword](#TWithKeyword) [WithAttribList](#TWithAttribList)

WithKeyword ::= "with"

WithAttribList ::= "{" [MultiWithAttrib](#TMultiWithAttrib) "}"

MultiWithAttrib ::= {[SingleWithAttrib](#TSingleWithAttrib) [[SemiColon](#TSemiColon)]}

SingleWithAttrib ::= [StandardAttribute](#TStandardAttribute) |

[VariantAttribute](#TVariantAttribute)

StandardAttribute ::= [AttribKeyword](#TAttribKeyword) [[OverrideKeyword](#TOverrideKeyword) | [LocalModifier](#localmodifier)] [[AttribQualifier](#TAttribQualifier)]

[FreeText](#TFreeText)

VariantAttribute ::= [VariantKeyword](#TVariantKeyword) [( [OverrideKeyword](#TOverrideKeyword) | [LocalModifier](#localmodifier) )]

[[AttribQualifier](#TAttribQualifier)] [ [RelatedEncoding](#TRelatedEncoding) "." ] [FreeText](#TFreeText)

RelatedEncoding ::= [FreeText](#TFreeText) | ( "{" [FreeText](#TFreeText) { "," [FreeText](#TFreeText) } "}" )

AttribKeyword ::= [EncodeKeyword](#TEncodeKeyword) |

[DisplayKeyword](#TDisplayKeyword) |

[ExtensionKeyword](#TExtensionKeyword) |

[OptionalKeyword](#TOptionalKeyword)

EncodeKeyword ::= "encode"

VariantKeyword ::= "variant"

DisplayKeyword ::= "display"

ExtensionKeyword ::= "extension"

OverrideKeyword ::= "override"

LocalModifier ::= "@local"

AttribQualifier ::= "(" [DefOrFieldRefList](#TDefOrFieldRefList) ")"

DefOrFieldRefList ::= [DefOrFieldRef](#TDefOrFieldRef) {"," [DefOrFieldRef](#TDefOrFieldRef)}

DefOrFieldRef ::= [QualifiedIdentifier](#TQualifiedIdentifier) |

(([FieldReference](#TFieldReference) | "[" [Minus](#TMinus) "]") [[ExtendedFieldOrTypeReference](#TExtendedFieldReference)]) |

[AllRef](#TAllRef)

QualifiedIdentifier ::= {[Identifier](#TIdentifier) [Dot](#TDot)} [Identifier](#TIdentifier)

ExtendedFieldOrTypeReference ::= {([Dot](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TDot) ([Identifier](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TIdentifier) | [PredefinedType](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TPredefinedType))) |

[IndexRef](#TIndexRef) | ("[" [Minus](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/0EG46CRK/CR7496-v1.docx#TMinus) "]") }+

/\* STATIC SEMANTIC - The *Identifier* refers to a type definition if the type of the *VarInstance* or *ReferencedValue* in which the *ExtendedFieldOrReference* is used is *anytype*. *IndexRef* shall be used when referencing elements of values or arrays. The square brackets with dash shall be used when referencing inner types of a record of, set of or array type. \*/

AllRef ::= ([GroupKeyword](#TGroupKeyword) [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword) "{" [QualifiedIdentifierList](#TQualifiedIdentifierList)

"}"]) | (([TypeDefKeyword](#TTypeDefKeyword) |

[TemplateKeyword](#TTemplateKeyword) |

[ConstKeyword](#TConstKeyword) |

[AltstepKeyword](#TAltstepKeyword) |

[TestcaseKeyword](#TTestcaseKeyword) |

[FunctionKeyword](#TFunctionKeyword) |

[SignatureKeyword](#TSignatureKeyword) |

[ModuleParKeyword](#TModuleParKeyword)

) [AllKeyword](#TAllKeyword) [[ExceptKeyword](#TExceptKeyword)

"{" [IdentifierList](#TIdentifierList)

"}"])

#### A.1.6.8.2 Behaviour statements

BehaviourStatements ::= [TestcaseInstance](#TTestcaseInstance) |

[FunctionInstance](#TFunctionInstance) |

[ReturnStatement](#TReturnStatement) |

[AltConstruct](#TAltConstruct) |

[InterleavedConstruct](#TInterleavedConstruct) |

[LabelStatement](#TLabelStatement) |

[GotoStatement](#TGotoStatement) |

[RepeatStatement](#TRepeatStatement) |

[DeactivateStatement](#TDeactivateStatement) |

[AltstepInstance](#TAltstepInstance) |

[ActivateOp](#TActivateOp) |

[BreakStatement](#TBreakStatement) |

[ContinueStatement](#TContinueStatement)

SetLocalVerdict ::= [SetVerdictKeyword](#TSetVerdictKeyword) "(" [SingleExpression](#TSingleExpression) {"," [LogItem](#TLogItem)} ")"

SetVerdictKeyword ::= "setverdict"

GetLocalVerdict ::= "getverdict"

SUTStatements ::= [ActionKeyword](#TActionKeyword) "(" [ActionText](#TActionText) {[StringOp](#TStringOp) [ActionText](#TActionText)} ")"

ActionKeyword ::= "action"

ActionText ::= [FreeText](#TFreeText) | [Expression](#TExpression)

ReturnStatement ::= [ReturnKeyword](#TReturnKeyword) [[TemplateInstance](#TTemplateInstance)]

/\* STATIC SEMANTICS - *TemplateInstance* shall evaluate to a value of a type compatible with the return type for functions returning a value. It shall evaluate to a value, template (literal or template instance), or a matching mechanism compatible with the return type for functions returning a template.\*/

AltConstruct ::= [AltKeyword](#TAltKeyword) [ [NoDefaultModifier](#TNoDefaultModifier) ] "{" [AltstepLocalDefList](#TAltstepLocalDefList) [AltGuardList](#TAltGuardList) "}"

AltKeyword ::= "alt"

AltGuardList ::= {[GuardStatement](#TGuardStatement) | [ElseStatement](#TElseStatement) [[SemiColon](#TSemiColon)]}

GuardStatement ::= [AltGuardChar](#TAltGuardChar) ([AltstepInstance](#TAltstepInstance) [[StatementBlock](#TStatementBlock)] |

[GuardOp](#TGuardOp) [StatementBlock](#TStatementBlock))

ElseStatement ::= "[" [ElseKeyword](#TElseKeyword) "]" [StatementBlock](#TStatementBlock)

AltGuardChar ::= "[" [[BooleanExpression](#TBooleanExpression)] "]"

GuardOp ::= [TimeoutStatement](#TTimeoutStatement) |

[ReceiveStatement](#TReceiveStatement) |

[TriggerStatement](#TTriggerStatement) |

[GetCallStatement](#TGetCallStatement) |

[CatchStatement](#TCatchStatement) |

[CheckStatement](#TCheckStatement) |

[GetReplyStatement](#TGetReplyStatement) |

[DoneStatement](#TDoneStatement) |

[KilledStatement](#TKilledStatement)

InterleavedConstruct ::= [InterleavedKeyword](#TInterleavedKeyword) [ [NoDefaultModifier](#TNoDefaultModifier) ] "{" [InterleavedGuardList](#TInterleavedGuardList) "}"

InterleavedKeyword ::= "interleave"

InterleavedGuardList ::= {[InterleavedGuardElement](#TInterleavedGuardElement) [[SemiColon](#TSemiColon)]}+

InterleavedGuardElement ::= [InterleavedGuard](#TInterleavedGuard) [StatementBlock](#TStatementBlock)

InterleavedGuard ::= "[" "]" [GuardOp](#TGuardOp)

LabelStatement ::= [LabelKeyword](#TLabelKeyword) [Identifier](#TIdentifier)

LabelKeyword ::= "label"

GotoStatement ::= [GotoKeyword](#TGotoKeyword) [Identifier](#TIdentifier)

GotoKeyword ::= "goto"

RepeatStatement ::= "repeat"

ActivateOp ::= [ActivateKeyword](#TActivateKeyword) "(" [AltstepInstance](#TAltstepInstance) ")"

ActivateKeyword ::= "activate"

DeactivateStatement ::= [DeactivateKeyword](#TDeactivateKeyword) ["(" ")"]

DeactivateKeyword ::= "deactivate"

BreakStatement ::= "break"

ContinueStatement ::= "continue"

#### A.1.6.8.3 Basic statements

BasicStatements ::= [Assignment](#TAssignment) |

[LogStatement](#TLogStatement) |

[LoopConstruct](#TLoopConstruct) |

[ConditionalConstruct](#TConditionalConstruct) |

[SelectCaseConstruct](#TSelectCaseConstruct) |

[StatementBlock](#TStatementBlock)

Expression ::= [SingleExpression](#TSingleExpression) | [CompoundExpression](#TCompoundExpression)

CompoundExpression ::= [FieldExpressionList](#TFieldExpressionList) | [ArrayOrMixedExpression](#TArrayExpression)

/\* STATIC SEMANTICS - Within *CompoundExpression* the *ArrayOrMixedExpression* can be used for Arrays, record, record of, set and set of types.\*/

FieldExpressionList ::= "{" [FieldExpressionSpec](#TFieldExpressionSpec) {"," [FieldExpressionSpec](#TFieldExpressionSpec)} "}"

FieldExpressionSpec ::= [FieldReference](#TFieldReference) [AssignmentChar](#TAssignmentChar) [NotUsedOrExpression](#TNotUsedOrExpression)

ArrayOrMixedExpression ::= "{" [[ArrayElementExpressionList](#TArrayElementExpressionList) {"," [FieldExpressionSpec](file:///C:/Users/ethgry/AppData/Local/Microsoft/Windows/Temporary%20Internet%20Files/Content.IE5/SJHEOEMY/CR7655.docx#TFieldExpressionSpec)}] "}"

ArrayElementExpressionList ::= [NotUsedOrExpression](#TNotUsedOrExpression) {"," [NotUsedOrExpression](#TNotUsedOrExpression)}

NotUsedOrExpression ::= [Expression](#TExpression) | [Minus](#TMinus)

ConstantExpression ::= [SingleExpression](#TSingleExpression) | [CompoundConstExpression](#TCompoundConstExpression)

BooleanExpression ::= [SingleExpression](#TSingleExpression)

/\* STATIC SEMANTICS - *BooleanExpression* shall resolve to a Value of type Boolean \*/

CompoundConstExpression ::= [FieldConstExpressionList](#TFieldConstExpressionList) | [ArrayConstExpression](#TArrayConstExpression)

/\* STATIC SEMANTICS - Within *CompoundConstExpression* the *ArrayConstExpression* can be used for arrays, record, record of and set of types. \*/

FieldConstExpressionList ::= "{" [FieldConstExpressionSpec](#TFieldConstExpressionSpec) {"," [FieldConstExpressionSpec](#TFieldConstExpressionSpec)} "}"

FieldConstExpressionSpec ::= [FieldReference](#TFieldReference) [AssignmentChar](#TAssignmentChar) [ConstantExpression](#TConstantExpression)

ArrayConstExpression ::= "{" [[ArrayElementConstExpressionList](#TArrayElementConstExpressionList)] "}"

ArrayElementConstExpressionList ::= [ConstantExpression](#TConstantExpression) {"," [ConstantExpression](#TConstantExpression)}

Assignment ::= [ValueRef](#TComponentRef) [AssignmentChar](#TAssignmentChar) [TemplateBody](#TTemplateBody)

/\* STATIC SEMANTICS - The *TemplateBody* on the right hand side of Assignment shall evaluate to an explicit value of a type compatible with the type of the left hand side for value variables and shall evaluate to an explicit value, template (literal or a template instance) or a matching mechanism compatible with the type of the left hand side for template variables. \*/

SingleExpression ::= [XorExpression](#TXorExpression) {"or" [XorExpression](#TXorExpression)}

/\* STATIC SEMANTICS - If more than one *XorExpression* exists, then the *XorExpressions* shall evaluate to specific values of compatible types \*/

XorExpression ::= [AndExpression](#TAndExpression) {"xor" [AndExpression](#TAndExpression)}

/\* STATIC SEMANTICS - If more than one *AndExpression* exists, then the *AndExpressions* shall evaluate to specific values of compatible types \*/

AndExpression ::= [NotExpression](#TNotExpression) {"and" [NotExpression](#TNotExpression)}

/\* STATIC SEMANTICS - If more than one *NotExpression* exists, then the *NotExpressions* shall evaluate to specific values of compatible types \*/

NotExpression ::= ["not"] [EqualExpression](#TEqualExpression)

/\* STATIC SEMANTICS - Operands of the not operator shall be of type boolean or derivatives of type Boolean. \*/

EqualExpression ::= [RelExpression](#TRelExpression) {[EqualOp](#TEqualOp) [RelExpression](#TRelExpression)}

/\* STATIC SEMANTICS - If more than one *RelExpression* exists, then the *RelExpressions* shall evaluate to specific values of compatible types. If only one RelExpression exists, it shall not derive to a *CompoundExpression*. \*/

RelExpression ::= [ShiftExpression](#TShiftExpression) [[RelOp](#TRelOp) [ShiftExpression](#TShiftExpression)] | [CompoundExpression](#TCompoundExpression)

/\* STATIC SEMANTICS - If both *ShiftExpressions* exist, then each *ShiftExpression* shall evaluate to a specific integer, Enumerated or float Value or derivatives of these types. \*/

ShiftExpression ::= [BitOrExpression](#TBitOrExpression) {[ShiftOp](#TShiftOp) [BitOrExpression](#TBitOrExpression)}

/\* STATIC SEMANTICS - Each Result shall resolve to a specific Value. If more than one Result exists the right-hand operand shall be of type integer or derivatives and if the shift op is "<<" or ">>" then the left-hand operand shall resolve to either bitstring, hexstring or octetstring type or derivatives of these types. If the shift op is " \*/

BitOrExpression ::= [BitXorExpression](#TBitXorExpression) {"or4b" [BitXorExpression](#TBitXorExpression)}

/\* STATIC SEMANTICS - If more than one *BitXorExpression* exists, then the *BitXorExpressions* shall evaluate to specific values of compatible types \*/

BitXorExpression ::= [BitAndExpression](#TBitAndExpression) {"xor4b" [BitAndExpression](#TBitAndExpression)}

/\* STATIC SEMANTICS - If more than one *BitAndExpression* exists, then the *BitAndExpressions* shall evaluate to specific values of compatible types \*/

BitAndExpression ::= [BitNotExpression](#TBitNotExpression) {"and4b" [BitNotExpression](#TBitNotExpression)}

/\* STATIC SEMANTICS - If more than one *BitNotExpression* exists, then the *BitNotExpressions* shall evaluate to specific values of compatible types \*/

BitNotExpression ::= ["not4b"] [AddExpression](#TAddExpression)

/\* STATIC SEMANTICS - If the *not4b* operator exists, the operand shall be of type bitstring, octetstring or hexstring or derivatives of these types. \*/

AddExpression ::= [MulExpression](#TMulExpression) {[AddOp](#TAddOp) [MulExpression](#TMulExpression)}

/\* STATIC SEMANTICS - Each *MulExpression* shall resolve to a specific Value. If more than one *MulExpression* exists and the *AddOp* resolves to *StringOp* then the *MulExpressions* shall be valid operands for *StringOp*. If more than one *MulExpression* exists and the *AddOp* does not resolve to *StringOp* then the *MulExpression* shall both resolve to type integer or float or derivatives of these types. If only one *MulExpression* exists, it shall not derive to a *CompoundExpression*. \*/

MulExpression ::= [UnaryExpression](#TUnaryExpression) {[MultiplyOp](#TMultiplyOp) [UnaryExpression](#TUnaryExpression)} | [CompoundExpression](#TCompoundExpression)

/\* STATIC SEMANTICS - Each *UnaryExpression* shall resolve to a specific Value. If more than one *UnaryExpression* exists then the *UnaryExpressions* shall resolve to type integer or float or derivatives of these types. \*/

UnaryExpression ::= [[UnaryOp](#TUnaryOp)] [Primary](#TPrimary)

/\* STATIC SEMANTICS - *Primary* shall resolve to a specific Value of type integer or float or derivatives of these types. \*/

Primary ::= [OpCall](#TOpCall) |

[PresenceCheckingOp](#TPresenceCheckingOp) |

[Value](#TValue) |

"(" [SingleExpression](#TSingleExpression) ")"

ExtendedFieldReference ::= {([Dot](#TDot) ([Identifier](#TIdentifier) | [PredefinedType](#TPredefinedType) | [FromKeyword](#TFromKeyword) | [ToKeyword](#TToKeyword))) |

[IndexRef](#TIndexRef) |

[DecodedFieldReference](#TDecodedFieldReference)

}+

/\* STATIC SEMANTIC - The *Identifier* refers to a type definition if the type of the *VarInstance* or *ReferencedValue* in which the *ExtendedFieldReference* is used is anytype. *IndexRef* shall be used when referencing elements of values or arrays. *DecodedFieldReference* shall not appear on the LHS of assignments and in type references\*/

DecodedFieldReference ::= "=>" [DecodedFieldType](#TDecodedFieldType)

DecodedFieldType ::= [PredefinedType](#TPredefinedType) |

[Identifier](#TIdentifier) |

"(" [Type](#TType) [ "," [Expression](#TExpression) ] ")"

/\* STATIC SEMANTIC - The *Identifier* shall resolve into a type \*/

OpCall ::= [ConfigurationOps](#TConfigurationOps) |

[GetLocalVerdict](#TGetLocalVerdict) |

[TimerOps](#TTimerOps) |

[TestcaseInstance](#TTestcaseInstance) |

([FunctionInstance](#TFunctionInstance) [[ExtendedFieldReference](#TExtendedFieldReference)]) |

([TemplateOps](#TTemplateOps) [[ExtendedFieldReference](#TExtendedFieldReference)]) |

[ActivateOp](#TActivateOp) |

[GetAttributeOp](#TGetAttributeOp)

PresenceCheckingOp := ("ispresent" | "ischosen" | "isvalue" | "isbound")

"(" [TemplateInstance](#TInLineTemplate) ")"

AddOp ::= "+" |

"-" |

[StringOp](#TStringOp)

/\* STATIC SEMANTICS - Operands of the "+" or "-" operators shall be of type integer or float or derivations of integer or float (i.e. subrange) \*/

MultiplyOp ::= "\*" | "/" | "mod" | "rem"

/\* STATIC SEMANTICS - Operands of the "\*", "/", rem or mod operators shall be of type integer or float or derivations of integer or float (i.e. subrange) \*/

UnaryOp ::= "+" | "-"

/\* STATIC SEMANTICS - Operands of the "+" or "-" operators shall be of type integer or float or derivations of integer or float (i.e. subrange) \*/

RelOp ::= "<" | ">" | ">=" | "<="

/\* STATIC SEMANTICS - the precedence of the operators is defined in Table 6 \*/

EqualOp ::= "==" | "!="

StringOp ::= "&"

/\* STATIC SEMANTICS - Operands of the list operator shall be bitstring, hexstring, octetstring, (universal) character string, record of, set of, or array types, or derivates of these types. \*/

ShiftOp ::= "<<" | ">>" | "<@" | "@>"

LogStatement ::= [LogKeyword](#TLogKeyword) "(" [LogItem](#TLogItem) {"," [LogItem](#TLogItem)} ")"

LogKeyword ::= "log"

LogItem ::= [FreeText](#TFreeText) | [TemplateInstance](#TTemplateInstance)

LoopConstruct ::= [ForStatement](#TForStatement) |

[WhileStatement](#TWhileStatement) |

[DoWhileStatement](#TDoWhileStatement)

ForStatement ::= [ForKeyword](#TForKeyword) "(" [Initial](#TInitial) [SemiColon](#TSemiColon) [BooleanExpression](#TBooleanExpression)

[SemiColon](#TSemiColon) [Assignment](#TAssignment) ")" [StatementBlock](#TStatementBlock)

ForKeyword ::= "for"

Initial ::= [VarInstance](#TVarInstance) | [Assignment](#TAssignment)

WhileStatement ::= [WhileKeyword](#TWhileKeyword) "(" [BooleanExpression](#TBooleanExpression) ")" [StatementBlock](#TStatementBlock)

WhileKeyword ::= "while"

DoWhileStatement ::= [DoKeyword](#TDoKeyword) [StatementBlock](#TStatementBlock) [WhileKeyword](#TWhileKeyword) "(" [BooleanExpression](#TBooleanExpression)

")"

DoKeyword ::= "do"

ConditionalConstruct ::= [IfKeyword](#TIfKeyword) "(" [BooleanExpression](#TBooleanExpression) ")" [StatementBlock](#TStatementBlock)

{[ElseIfClause](#TElseIfClause)} [[ElseClause](#TElseClause)]

IfKeyword ::= "if"

ElseIfClause ::= [ElseKeyword](#TElseKeyword) [IfKeyword](#TIfKeyword) "(" [BooleanExpression](#TBooleanExpression) ")" [StatementBlock](#TStatementBlock)

ElseKeyword ::= "else"

ElseClause ::= [ElseKeyword](#TElseKeyword) [StatementBlock](#TStatementBlock)

SelectCaseConstruct ::= [SelectKeyword](#TSelectKeyword) [[UnionKeyword](#TUnionKeyword)] "(" [SingleExpression](#TSingleExpression) ")" [SelectCaseBody](#TSelectCaseBody)

SelectKeyword ::= "select"

SelectCaseBody ::= "{" {[SelectCase](#TSelectCase)}+ [[CaseElse](#TCaseElse)] "}"

SelectCase ::= [CaseKeyword](#TCaseKeyword) ("("[TemplateInstance](#TTemplateInstance) {"," [TemplateInstance](#TTemplateInstance)}

")" | [ElseKeyword](#TElseKeyword)) [StatementBlock](#TStatementBlock)

/\* STATIC SEMANTIC - *TemplateInstance*-s shall be *Identifier*-s if the *UnionKeyword* is present in the surrounding *SelectCaseConstruct* (see clause 19.3.2) \*/

CaseElse ::= [CaseKeyword](#TCaseKeyword) [ElseKeyword](#TElseKeyword) [StatementBlock](#TStatementBlock)

CaseKeyword ::= "case"

ExtendedIdentifier ::= [[Identifier](#TIdentifier) [Dot](#TDot)] [Identifier](#TIdentifier)

IdentifierList ::= [Identifier](#TIdentifier) {"," [Identifier](#TIdentifier)}

QualifiedIdentifierList ::= [QualifiedIdentifier](#TQualifiedIdentifier) {"," [QualifiedIdentifier](#TQualifiedIdentifier)}

GetAttributeOp ::= ([Type](#TType) | [TemplateInstance](#TTemplateInstance)) "." [GetAttributeSpec](#TGetAttributeSpec)

GetAttributeSpec ::= [EncodeKeyword](#TEncodeKeyword) |

[VariantKeyword](#TVariantKeyword) ["(" [FreeText](#TFreeText) ")"] |

[DisplayKeyword](#TDisplayKeyword) |

[ExtensionKeyword](#TExtensionKeyword) |

[OptionalKeyword](#TOptionalKeyword)

### 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"