**ttcn3**

A.1.6 TTCN-3 Syntax Productions

1. TTCN3Module ::= [TTCN3ModuleKeyword](#TTCN3ModuleKeyword) [ModuleId](#ModuleId) “{ ” [ [ModuleDefinitionsList](#ModuleDefinitionsList) ] [ [ModuleControlPart](#ModuleControlPart) ] “} ” [ [WithStatement](#WithStatement) ] [ [SemiColon](#SemiColon) ]

2. TTCN3ModuleKeyword ::= “module”

3. ModuleId ::= [Identifier](#Identifier) [ [LanguageSpec](#LanguageSpec) ]

4. LanguageSpec ::= [LanguageKeyword](#LanguageKeyword) [FreeText](#FreeText) { “,” [FreeText](#FreeText) }

5. LanguageKeyword ::= “language”

A.1.6.1 Module definitions part

A.1.6.1.0 General

6. ModuleDefinitionsList ::= { [ModuleDefinition](#ModuleDefinition) [ [SemiColon](#SemiColon) ] } +

7. ModuleDefinition ::= ( ( [ [Visibility](#Visibility) ] ( [TypeDef](#TypeDef) | [ConstDef](#ConstDef) | [TemplateDef](#TemplateDef) | [ModuleParDef](#ModuleParDef) | [FunctionDef](#FunctionDef) | [SignatureDef](#SignatureDef) | [TestcaseDef](#TestcaseDef) | [AltstepDef](#AltstepDef) | [ImportDef](#ImportDef) | [ExtFunctionDef](#ExtFunctionDef) | [ExtConstDef](#ExtConstDef) ) ) | ( [ “public” ] [GroupDef](#GroupDef) ) | ( [ “private” ] [FriendModuleDef](#FriendModuleDef) ) ) [ [WithStatement](#WithStatement) ]

8. Visibility ::= “public” | “friend” | “private”

A.1.6.1.1 Typedef definitions

9. TypeDef ::= [TypeDefKeyword](#TypeDefKeyword) [TypeDefBody](#TypeDefBody)

10. TypeDefBody ::= [StructuredTypeDef](#StructuredTypeDef) | [SubTypeDef](#SubTypeDef)

11. TypeDefKeyword ::= “type”

12. StructuredTypeDef ::= [RecordDef](#RecordDef) | [UnionDef](#UnionDef) | [SetDef](#SetDef) | [RecordOfDef](#RecordOfDef) | [SetOfDef](#SetOfDef) | [EnumDef](#EnumDef) | [PortDef](#PortDef) | [ComponentDef](#ComponentDef)

13. RecordDef ::= [RecordKeyword](#RecordKeyword) [StructDefBody](#StructDefBody)

14. RecordKeyword ::= “record”

15. StructDefBody ::= ( [Identifier](#Identifier) | [AddressKeyword](#AddressKeyword) ) “{ ” [ [StructFieldDef](#StructFieldDef) { “,” [StructFieldDef](#StructFieldDef) } ] “} ”

16. StructFieldDef ::= ( [Type](#Type) | [NestedTypeDef](#NestedTypeDef) ) [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ] [ [SubTypeSpec](#SubTypeSpec) ] [ [OptionalKeyword](#OptionalKeyword) ]

17. NestedTypeDef ::= [NestedRecordDef](#NestedRecordDef) | [NestedUnionDef](#NestedUnionDef) | [NestedSetDef](#NestedSetDef) | [NestedRecordOfDef](#NestedRecordOfDef) | [NestedSetOfDef](#NestedSetOfDef) | [NestedEnumDef](#NestedEnumDef)

18. NestedRecordDef ::= [RecordKeyword](#RecordKeyword) “{ ” [ [StructFieldDef](#StructFieldDef) { “,” [StructFieldDef](#StructFieldDef) } ] “} ”

19. NestedUnionDef ::= [UnionKeyword](#UnionKeyword) “{ ” [UnionFieldDef](#UnionFieldDef) { “,” [UnionFieldDef](#UnionFieldDef) } “} ”

20. NestedSetDef ::= [SetKeyword](#SetKeyword) “{ ” [ [StructFieldDef](#StructFieldDef) { “,” [StructFieldDef](#StructFieldDef) } ] “} ”

21. NestedRecordOfDef ::= [RecordKeyword](#RecordKeyword) [ [StringLength](#StringLength) ] [OfKeyword](#OfKeyword) ( [Type](#Type) | [NestedTypeDef](#NestedTypeDef) )

22. NestedSetOfDef ::= [SetKeyword](#SetKeyword) [ [StringLength](#StringLength) ] [OfKeyword](#OfKeyword) ( [Type](#Type) | [NestedTypeDef](#NestedTypeDef) )

23. NestedEnumDef ::= [EnumKeyword](#EnumKeyword) “{ ” [EnumerationList](#EnumerationList) “} ”

24. OptionalKeyword ::= “optional”

25. UnionDef ::= [UnionKeyword](#UnionKeyword) [UnionDefBody](#UnionDefBody)

26. UnionKeyword ::= “union”

27. UnionDefBody ::= ( [Identifier](#Identifier) | [AddressKeyword](#AddressKeyword) ) “{ ” [UnionFieldDef](#UnionFieldDef) { “,” [UnionFieldDef](#UnionFieldDef) } “} ”

28. UnionFieldDef ::= ( [Type](#Type) | [NestedTypeDef](#NestedTypeDef) ) [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ] [ [SubTypeSpec](#SubTypeSpec) ]

29. SetDef ::= [SetKeyword](#SetKeyword) [StructDefBody](#StructDefBody)

30. SetKeyword ::= “set”

31. RecordOfDef ::= [RecordKeyword](#RecordKeyword) [ [StringLength](#StringLength) ] [OfKeyword](#OfKeyword) [StructOfDefBody](#StructOfDefBody)

32. OfKeyword ::= “of”

33. StructOfDefBody ::= ( [Type](#Type) | [NestedTypeDef](#NestedTypeDef) ) ( [Identifier](#Identifier) | [AddressKeyword](#AddressKeyword) ) [ [SubTypeSpec](#SubTypeSpec) ]

34. SetOfDef ::= [SetKeyword](#SetKeyword) [ [StringLength](#StringLength) ] [OfKeyword](#OfKeyword) [StructOfDefBody](#StructOfDefBody)

35. EnumDef ::= [EnumKeyword](#EnumKeyword) ( [Identifier](#Identifier) | [AddressKeyword](#AddressKeyword) ) “{ ” [EnumerationList](#EnumerationList) “} ”

36. EnumKeyword ::= “enumerated”

37. EnumerationList ::= [Enumeration](#Enumeration) { “,” [Enumeration](#Enumeration) }

38. Enumeration ::= [Identifier](#Identifier) [ “(” [IntegerValueOrRange](#IntegerValueOrRange) { “,” [IntegerValueOrRange](#IntegerValueOrRange) } “)” ]

39. IntegerValueOrRange ::= [IntegerValue](#IntegerValue) [ “..” [IntegerValue](#IntegerValue) ]

40. IntegerValue ::= [ [Minus](#Minus) ] [Number](#Number)

41. SubTypeDef ::= [Type](#Type) ( [Identifier](#Identifier) | [AddressKeyword](#AddressKeyword) ) [ [ArrayDef](#ArrayDef) ] [ [SubTypeSpec](#SubTypeSpec) ]

42. SubTypeSpec ::= [AllowedValuesSpec](#AllowedValuesSpec) [ [StringLength](#StringLength) ] | [StringLength](#StringLength)

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

43. AllowedValuesSpec ::= “(” ( ( [TemplateOrRange](#TemplateOrRange) { “,” [TemplateOrRange](#TemplateOrRange) } ) | [CharStringMatch](#CharStringMatch) ) “)”

44. TemplateOrRange ::= [RangeDef](#RangeDef) | [TemplateBody](#TemplateBody) | [Type](#Type)

/\* 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 \*/

45. RangeDef ::= [Bound](#Bound) “..” [Bound](#Bound)

46. StringLength ::= [LengthKeyword](#LengthKeyword) “(” [SingleExpression](#SingleExpression) [ “..” ( [SingleExpression](#SingleExpression) | [InfinityKeyword](#InfinityKeyword) ) ] “)”

/\* 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) \*/

47. LengthKeyword ::= “length”

48. PortDef ::= [PortKeyword](#PortKeyword) [PortDefBody](#PortDefBody)

49. PortDefBody ::= [Identifier](#Identifier) [PortDefAttribs](#PortDefAttribs)

50. PortKeyword ::= “port”

51. PortDefAttribs ::= [MessageAttribs](#MessageAttribs) | [ProcedureAttribs](#ProcedureAttribs) | [MixedAttribs](#MixedAttribs)

52. MessageAttribs ::= [MessageKeyword](#MessageKeyword) “{ ” { ( [AddressDecl](#AddressDecl) | [MessageList](#MessageList) | [ConfigParamDef](#ConfigParamDef) ) [ [SemiColon](#SemiColon) ] } + “} ”

53. ConfigParamDef ::= [MapParamDef](#MapParamDef) | [UnmapParamDef](#UnmapParamDef)

54. MapParamDef ::= [MapKeyword](#MapKeyword) [ParamKeyword](#ParamKeyword) “(” [FormalValuePar](#FormalValuePar) { “,” [FormalValuePar](#FormalValuePar) } “)”

55. UnmapParamDef ::= [UnmapKeyword](#UnmapKeyword) [ParamKeyword](#ParamKeyword) “(” [FormalValuePar](#FormalValuePar) { “,” [FormalValuePar](#FormalValuePar) } “)”

56. AddressDecl ::= [AddressKeyword](#AddressKeyword) [Type](#Type)

57. MessageList ::= [Direction](#Direction) [AllOrTypeList](#AllOrTypeList)

58. Direction ::= [InParKeyword](#InParKeyword) | [OutParKeyword](#OutParKeyword) | [InOutParKeyword](#InOutParKeyword)

59. MessageKeyword ::= “message”

60. AllOrTypeList ::= [AllKeyword](#AllKeyword) | [TypeList](#TypeList)

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

61. AllKeyword ::= “all”

62. TypeList ::= [Type](#Type) { “,” [Type](#Type) }

63. ProcedureAttribs ::= [ProcedureKeyword](#ProcedureKeyword) “{ ” { ( [AddressDecl](#AddressDecl) | [ProcedureList](#ProcedureList) | [ConfigParamDef](#ConfigParamDef) ) [ [SemiColon](#SemiColon) ] } + “} ”

64. ProcedureKeyword ::= “procedure”

65. ProcedureList ::= [Direction](#Direction) [AllOrSignatureList](#AllOrSignatureList)

66. AllOrSignatureList ::= [AllKeyword](#AllKeyword) | [SignatureList](#SignatureList)

67. SignatureList ::= [Signature](#Signature) { “,” [Signature](#Signature) }

68. MixedAttribs ::= [MixedKeyword](#MixedKeyword) “{ ” { ( [AddressDecl](#AddressDecl) | [MixedList](#MixedList) | [ConfigParamDef](#ConfigParamDef) ) [ [SemiColon](#SemiColon) ] } + “} ”

69. MixedKeyword ::= “mixed”

70. MixedList ::= [Direction](#Direction) [ProcOrTypeList](#ProcOrTypeList)

71. ProcOrTypeList ::= [AllKeyword](#AllKeyword) | ( [ProcOrType](#ProcOrType) { “,” [ProcOrType](#ProcOrType) } )

72. ProcOrType ::= [Signature](#Signature) | [Type](#Type)

73. ComponentDef ::= [ComponentKeyword](#ComponentKeyword) [Identifier](#Identifier) [ [ExtendsKeyword](#ExtendsKeyword) [ComponentType](#ComponentType) { “,” [ComponentType](#ComponentType) } ] “{ ” [ [ComponentDefList](#ComponentDefList) ] “} ”

74. ComponentKeyword ::= “component”

75. ExtendsKeyword ::= “extends”

76. ComponentType ::= [ExtendedIdentifier](#ExtendedIdentifier)

77. ComponentDefList ::= { [ComponentElementDef](#ComponentElementDef) [ [WithStatement](#WithStatement) ] [ [SemiColon](#SemiColon) ] }

78. ComponentElementDef ::= [PortInstance](#PortInstance) | [VarInstance](#VarInstance) | [TimerInstance](#TimerInstance) | [ConstDef](#ConstDef) | [TemplateDef](#TemplateDef)

79. PortInstance ::= [PortKeyword](#PortKeyword) [ExtendedIdentifier](#ExtendedIdentifier) [PortElement](#PortElement) { “,” [PortElement](#PortElement) }

80. PortElement ::= [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ]

A.1.6.1.2 Constant definitions

81. ConstDef ::= [ConstKeyword](#ConstKeyword) [Type](#Type) [ConstList](#ConstList)

82. ConstList ::= [SingleConstDef](#SingleConstDef) { “,” [SingleConstDef](#SingleConstDef) }

83. SingleConstDef ::= [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ] [AssignmentChar](#AssignmentChar) [ConstantExpression](#ConstantExpression)

84. ConstKeyword ::= “const”

A.1.6.1.3 Template definitions

85. TemplateDef ::= [TemplateKeyword](#TemplateKeyword) [ [TemplateRestriction](#TemplateRestriction) ] [ [FuzzyModifier](#FuzzyModifier) ] [BaseTemplate](#BaseTemplate) [ [DerivedDef](#DerivedDef) ] [AssignmentChar](#AssignmentChar) [TemplateBody](#TemplateBody)

86. BaseTemplate ::= ( [Type](#Type) | [Signature](#Signature) ) [Identifier](#Identifier) [ “(” [TemplateOrValueFormalParList](#TemplateOrValueFormalParList) “)” ]

87. TemplateKeyword ::= “template”

88. DerivedDef ::= [ModifiesKeyword](#ModifiesKeyword) [ExtendedIdentifier](#ExtendedIdentifier)

89. ModifiesKeyword ::= “modifies”

90. TemplateOrValueFormalParList ::= [TemplateOrValueFormalPar](#TemplateOrValueFormalPar) { “,” [TemplateOrValueFormalPar](#TemplateOrValueFormalPar) }

91. TemplateOrValueFormalPar ::= [FormalValuePar](#FormalValuePar) | [FormalTemplatePar](#FormalTemplatePar)

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

92. TemplateBody ::= ( [SimpleSpec](#SimpleSpec) | [FieldSpecList](#FieldSpecList) | [ArrayValueOrAttrib](#ArrayValueOrAttrib) ) [ [ExtraMatchingAttributes](#ExtraMatchingAttributes) ]

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

93. SimpleSpec ::= ( [SingleExpression](#SingleExpression) [ “&” [SimpleTemplateSpec](#SimpleTemplateSpec) ] ) | [SimpleTemplateSpec](#SimpleTemplateSpec)

94. SimpleTemplateSpec ::= [SingleTemplateExpression](#SingleTemplateExpression) [ “&” [SimpleSpec](#SimpleSpec) ]

95. SingleTemplateExpression ::= [MatchingSymbol](#MatchingSymbol) | ( [TemplateRefWithParList](#TemplateRefWithParList) [ [ExtendedFieldReference](#ExtendedFieldReference) ] ) | [ExtendedIdentifier](#ExtendedIdentifier) [EnumTemplateExtension](#EnumTemplateExtension)

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

96. EnumTemplateExtension ::= “(” [TemplateBody](#TemplateBody) { “,” [TemplateBody](#TemplateBody) } “)”

/\*\* STATIC Semantics: each TemplateBody shall be an integer template \*/

97. FieldSpecList ::= “{ ” [FieldSpec](#FieldSpec) { “,” [FieldSpec](#FieldSpec) } “} ”

98. FieldSpec ::= [FieldReference](#FieldReference) [AssignmentChar](#AssignmentChar) ( [TemplateBody](#TemplateBody) | [Minus](#Minus) )

99. FieldReference ::= [StructFieldRef](#StructFieldRef) | [ArrayOrBitRef](#ArrayOrBitRef) | [ParRef](#ParRef)

100. StructFieldRef ::= [Identifier](#Identifier) | [PredefinedType](#PredefinedType) | [TypeReference](#TypeReference)

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

101. ParRef ::= [Identifier](#Identifier)

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

102. ArrayOrBitRef ::= “[” [FieldOrBitNumber](#FieldOrBitNumber) “]”

/\* 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 \*/

103. FieldOrBitNumber ::= [SingleExpression](#SingleExpression)

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

104. ArrayValueOrAttrib ::= “{ ” [ [ArrayElementSpecList](#ArrayElementSpecList) ] “} ”

105. ArrayElementSpecList ::= [ArrayElementSpec](#ArrayElementSpec) { “,” [ArrayElementSpec](#ArrayElementSpec) }

106. ArrayElementSpec ::= [Minus](#Minus) | [PermutationMatch](#PermutationMatch) | [TemplateBody](#TemplateBody)

107. MatchingSymbol ::= [Complement](#Complement) | ( [AnyValue](#AnyValue) [ [WildcardLengthMatch](#WildcardLengthMatch) ] ) | ( [AnyOrOmit](#AnyOrOmit) [ [WildcardLengthMatch](#WildcardLengthMatch) ] ) | [ListOfTemplates](#ListOfTemplates) | [Range](#Range) | [BitStringMatch](#BitStringMatch) | [HexStringMatch](#HexStringMatch) | [OctetStringMatch](#OctetStringMatch) | [CharStringMatch](#CharStringMatch) | [SubsetMatch](#SubsetMatch) | [SupersetMatch](#SupersetMatch) | [DecodedContentMatch](#DecodedContentMatch)

108. DecodedContentMatch ::= [DecodedMatchKeyword](#DecodedMatchKeyword) [ “(” [ [Expression](#Expression) ] “)” ] [TemplateInstance](#TemplateInstance)

108. 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. \*/

109. ExtraMatchingAttributes ::= [StringLength](#StringLength) | [IfPresentKeyword](#IfPresentKeyword) | ( [StringLength](#StringLength) [IfPresentKeyword](#IfPresentKeyword) )

110. BitStringMatch ::= “’” { [BinOrMatch](#BinOrMatch) } “’” “B”

111. BinOrMatch ::= [Bin](#Bin) | [AnyValue](#AnyValue) | [AnyOrOmit](#AnyOrOmit)

112. HexStringMatch ::= “’” { [HexOrMatch](#HexOrMatch) } “’” “H”

113. HexOrMatch ::= [Hex](#Hex) | [AnyValue](#AnyValue) | [AnyOrOmit](#AnyOrOmit)

114. OctetStringMatch ::= “’” { [OctOrMatch](#OctOrMatch) } “’” “O”

115. OctOrMatch ::= [Oct](#Oct) | [AnyValue](#AnyValue) | [AnyOrOmit](#AnyOrOmit)

116. CharStringMatch ::= [PatternKeyword](#PatternKeyword) [ [CaseInsenModifier](#CaseInsenModifier) ] [PatternParticle](#PatternParticle) { “&” [PatternParticle](#PatternParticle) }

117. PatternParticle ::= [Pattern](#Pattern) | [ReferencedValue](#ReferencedValue)

118. PatternKeyword ::= “pattern”

119. Pattern ::= “”” { [PatternElement](#PatternElement) } “””

120. PatternElement ::= ( ( “\ ” ( “?” | “\*” | “\ ” | “[” | “]” | “{ ” | “} ” | “”” | “|” | “(” | “)” | “#” | “+” | “d” | “w” | “t” | “n” | “r” | “s” | “b” ) ) | ( “?” | “\*” | “\ ” | “|” | “+” ) | ( “[” [ “^” ] [ { [PatternClassChar](#PatternClassChar) [ “-” [PatternClassChar](#PatternClassChar) ] } ] “]” ) | ( “{ ” [ “\ ” ] [ReferencedValue](#ReferencedValue) “} ” ) | ( “\ ” “N” “{ ” ( [ReferencedValue](#ReferencedValue) | [Type](#Type) ) “} ” ) | ( “”” “”” ) | ( “(” [PatternElement](#PatternElement) “)” ) | ( “#” ( [Num](#Num) | ( “(” [Number](#Number) “,” [ [Number](#Number) ] “)” ) | ( “(” “,” [Number](#Number) “)” ) | ( “(” [ “,” ] “)” ) [Num](#Num) “)” ) ) ) | [PatternChar](#PatternChar)

121. PatternChar ::= [NonSpecialPatternChar](#NonSpecialPatternChar) | [PatternQuadruple](#PatternQuadruple)

/\* 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 \*/

122. NonSpecialPatternChar ::= [Char](#Char)

123. PatternClassChar ::= [NonSpecialPatternClassChar](#NonSpecialPatternClassChar) | [PatternQuadruple](#PatternQuadruple) | “\ ” [EscapedPatternClassChar](#EscapedPatternClassChar)

124. NonSpecialPatternClassChar ::= [Char](#Char)

/\* 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 \*/

125. EscapedPatternClassChar ::= “[” | “-” | “^” | “]”

126. PatternQuadruple ::= “\ ” “q” “(” [Number](#Number) “,” [Number](#Number) “,” [Number](#Number) “,” [Number](#Number) “)”

127. Complement ::= [ComplementKeyword](#ComplementKeyword) [ListOfTemplates](#ListOfTemplates)

128. ComplementKeyword ::= “complement”

129. ListOfTemplates ::= “(” [TemplateListItem](#TemplateListItem) { “,” [TemplateListItem](#TemplateListItem) } “)”

130. TemplateListItem ::= [TemplateBody](#TemplateBody) | [AllElementsFrom](#AllElementsFrom)

131. AllElementsFrom ::= [AllKeyword](#AllKeyword) [FromKeyword](#FromKeyword) [TemplateBody](#TemplateBody)

132. SubsetMatch ::= [SubsetKeyword](#SubsetKeyword) [ListOfTemplates](#ListOfTemplates)

133. SubsetKeyword ::= “subset”

134. SupersetMatch ::= [SupersetKeyword](#SupersetKeyword) [ListOfTemplates](#ListOfTemplates)

135. SupersetKeyword ::= “superset”

136. PermutationMatch ::= [PermutationKeyword](#PermutationKeyword) [ListOfTemplates](#ListOfTemplates)

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

137. PermutationKeyword ::= “permutation”

138. AnyValue ::= “?”

139. AnyOrOmit ::= “\*”

140. WildcardLengthMatch ::= [LengthKeyword](#LengthKeyword) “(” [SingleExpression](#SingleExpression) “)”

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

141. IfPresentKeyword ::= “ifpresent”

142. PresentKeyword ::= “present”

143. Range ::= “(” [Bound](#Bound) “..” [Bound](#Bound) “)”

144. Bound ::= ( [ “!” ] [SingleExpression](#SingleExpression) ) | ( [ [Minus](#Minus) ] [InfinityKeyword](#InfinityKeyword) )

/\* 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. \*/

145. InfinityKeyword ::= “infinity”

146. TemplateInstanceAssignment ::= [Identifier](#Identifier) “:=” [TemplateInstance](#TemplateInstance)

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

147. TemplateRefWithParList ::= [ExtendedIdentifier](#ExtendedIdentifier) [ [TemplateActualParList](#TemplateActualParList) ]

148. TemplateInstance ::= [ ( [Type](#Type) | [Signature](#Signature) ) [Colon](#Colon) ] [ [DerivedRefWithParList](#DerivedRefWithParList) [AssignmentChar](#AssignmentChar) ] [TemplateBody](#TemplateBody)

149. DerivedRefWithParList ::= [ModifiesKeyword](#ModifiesKeyword) [TemplateRefWithParList](#TemplateRefWithParList)

150. TemplateActualParList ::= “(” [ ( [TemplateInstanceActualPar](#TemplateInstanceActualPar) { “,” [TemplateInstanceActualPar](#TemplateInstanceActualPar) } ) | ( [TemplateInstanceAssignment](#TemplateInstanceAssignment) { “,” [TemplateInstanceAssignment](#TemplateInstanceAssignment) } ) ] “)”

151. TemplateInstanceActualPar ::= [TemplateInstance](#TemplateInstance) | [Minus](#Minus)

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

152. TemplateOps ::= [MatchOp](#MatchOp) | [ValueofOp](#ValueofOp)

153. MatchOp ::= [MatchKeyword](#MatchKeyword) “(” [Expression](#Expression) “,” [TemplateInstance](#TemplateInstance) “)”

154. MatchKeyword ::= “match”

155. ValueofOp ::= [ValueofKeyword](#ValueofKeyword) “(” [TemplateInstance](#TemplateInstance) “)”

156. ValueofKeyword ::= “valueof”

A.1.6.1.4 Function definitions

157. FunctionDef ::= [FunctionKeyword](#FunctionKeyword) [ [DeterministicModifier](#DeterministicModifier) ] [Identifier](#Identifier) “(” [ [FunctionFormalParList](#FunctionFormalParList) ] “)” [ [RunsOnSpec](#RunsOnSpec) ] [ [MtcSpec](#MtcSpec) ] [ [SystemSpec](#SystemSpec) ] [ [ReturnType](#ReturnType) ] [StatementBlock](#StatementBlock)

158. FunctionKeyword ::= “function”

159. FunctionFormalParList ::= [FunctionFormalPar](#FunctionFormalPar) { “,” [FunctionFormalPar](#FunctionFormalPar) }

160. FunctionFormalPar ::= [FormalValuePar](#FormalValuePar) | [FormalTimerPar](#FormalTimerPar) | [FormalTemplatePar](#FormalTemplatePar) | [FormalPortPar](#FormalPortPar)

161. ReturnType ::= [ReturnKeyword](#ReturnKeyword) [ [TemplateKeyword](#TemplateKeyword) | [RestrictedTemplate](#RestrictedTemplate) ] [Type](#Type)

162. ReturnKeyword ::= “return”

163. RunsOnSpec ::= [RunsKeyword](#RunsKeyword) [OnKeyword](#OnKeyword) [ComponentType](#ComponentType)

164. RunsKeyword ::= “runs”

165. OnKeyword ::= “on”

166. MtcSpec ::= [MTCKeyword](#MTCKeyword) [ComponentType](#ComponentType)

167. MTCKeyword ::= “mtc”

168. StatementBlock ::= “{ ” [ [FunctionDefList](#FunctionDefList) ] [ [FunctionStatementList](#FunctionStatementList) ] “} ”

169. FunctionDefList ::= { ( [FunctionLocalDef](#FunctionLocalDef) | [FunctionLocalInst](#FunctionLocalInst) ) [ [WithStatement](#WithStatement) ] [ [SemiColon](#SemiColon) ] } +

170. FunctionStatementList ::= { [FunctionStatement](#FunctionStatement) [ [SemiColon](#SemiColon) ] } +

171. FunctionLocalInst ::= [VarInstance](#VarInstance) | [TimerInstance](#TimerInstance)

172. FunctionLocalDef ::= [ConstDef](#ConstDef) | [TemplateDef](#TemplateDef)

173. FunctionStatement ::= [ConfigurationStatements](#ConfigurationStatements) | [TimerStatements](#TimerStatements) | [CommunicationStatements](#CommunicationStatements) | [BasicStatements](#BasicStatements) | [BehaviourStatements](#BehaviourStatements) | [SetLocalVerdict](#SetLocalVerdict) | [SUTStatements](#SUTStatements) | [TestcaseOperation](#TestcaseOperation)

174. FunctionInstance ::= [FunctionRef](#FunctionRef) “(” [ [FunctionActualParList](#FunctionActualParList) ] “)”

175. FunctionRef ::= [ [Identifier](#Identifier) [Dot](#Dot) ] ( [Identifier](#Identifier) | [PreDefFunctionIdentifier](#PreDefFunctionIdentifier) )

176. PreDefFunctionIdentifier ::= [Identifier](#Identifier) [ [CaseInsenModifier](#CaseInsenModifier) ]

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

177. FunctionActualParList ::= ( [FunctionActualPar](#FunctionActualPar) { “,” [FunctionActualPar](#FunctionActualPar) } ) | ( [FunctionActualParAssignment](#FunctionActualParAssignment) { “,” [FunctionActualParAssignment](#FunctionActualParAssignment) } )

178. FunctionActualPar ::= [ArrayIdentifierRef](#ArrayIdentifierRef) | [TemplateInstance](#TemplateInstance) | [ComponentRef](#ComponentRef) | [Minus](#Minus)

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

179. FunctionActualParAssignment ::= [TemplateInstanceAssignment](#TemplateInstanceAssignment) | [ComponentRefAssignment](#ComponentRefAssignment) | [ArrayIdentifierRefAssignment](#ArrayIdentifierRefAssignment)

180. ArrayIdentifierRefAssignment ::= [Identifier](#Identifier) “:=” [ArrayIdentifierRef](#ArrayIdentifierRef)

A.1.6.1.5 Signature definitions

181. SignatureDef ::= [SignatureKeyword](#SignatureKeyword) [Identifier](#Identifier) “(” [ [SignatureFormalParList](#SignatureFormalParList) ] “)” [ [ReturnType](#ReturnType) | [NoBlockKeyword](#NoBlockKeyword) ] [ [ExceptionSpec](#ExceptionSpec) ]

182. SignatureKeyword ::= “signature”

183. SignatureFormalParList ::= [FormalValuePar](#FormalValuePar) { “,” [FormalValuePar](#FormalValuePar) }

184. ExceptionSpec ::= [ExceptionKeyword](#ExceptionKeyword) “(” [TypeList](#TypeList) “)”

185. ExceptionKeyword ::= “exception”

186. Signature ::= [ExtendedIdentifier](#ExtendedIdentifier)

187. NoBlockKeyword ::= “noblock”

A.1.6.1.6 Testcase definitions

188. TestcaseDef ::= [TestcaseKeyword](#TestcaseKeyword) [Identifier](#Identifier) “(” [ [TemplateOrValueFormalParList](#TemplateOrValueFormalParList) ] “)” [ConfigSpec](#ConfigSpec) [StatementBlock](#StatementBlock)

189. TestcaseKeyword ::= “testcase”

190. ConfigSpec ::= [RunsOnSpec](#RunsOnSpec) [ [SystemSpec](#SystemSpec) ]

191. SystemSpec ::= [SystemKeyword](#SystemKeyword) [ComponentType](#ComponentType)

192. SystemKeyword ::= “system”

193. TestcaseInstance ::= [ExecuteKeyword](#ExecuteKeyword) “(” [ExtendedIdentifier](#ExtendedIdentifier) “(” [ [TestcaseActualParList](#TestcaseActualParList) ] “)” [ “,” ( [Expression](#Expression) | [Minus](#Minus) ) [ “,” [SingleExpression](#SingleExpression) ] ] “)”

194. ExecuteKeyword ::= “execute”

195. TestcaseActualParList ::= ( [TemplateInstanceActualPar](#TemplateInstanceActualPar) { “,” [TemplateInstanceActualPar](#TemplateInstanceActualPar) } ) | ( [TemplateInstanceAssignment](#TemplateInstanceAssignment) { “,” [TemplateInstanceAssignment](#TemplateInstanceAssignment) } )

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

A.1.6.1.7 Altstep definitions

196. AltstepDef ::= [AltstepKeyword](#AltstepKeyword) [Identifier](#Identifier) “(” [ [FunctionFormalParList](#FunctionFormalParList) ] “)” [ [RunsOnSpec](#RunsOnSpec) ] [ [MtcSpec](#MtcSpec) ] [ [SystemSpec](#SystemSpec) ] “{ ” [AltstepLocalDefList](#AltstepLocalDefList) [AltGuardList](#AltGuardList) “} ”

197. AltstepKeyword ::= “altstep”

198. AltstepLocalDefList ::= { [AltstepLocalDef](#AltstepLocalDef) [ [WithStatement](#WithStatement) ] [ [SemiColon](#SemiColon) ] }

199. AltstepLocalDef ::= [VarInstance](#VarInstance) | [TimerInstance](#TimerInstance) | [ConstDef](#ConstDef) | [TemplateDef](#TemplateDef)

200. AltstepInstance ::= [ExtendedIdentifier](#ExtendedIdentifier) “(” [ [FunctionActualParList](#FunctionActualParList) ] “)”

A.1.6.1.8 Import definitions

201. ImportDef ::= [ImportKeyword](#ImportKeyword) [ImportFromSpec](#ImportFromSpec) ( [AllWithExcepts](#AllWithExcepts) | ( “{ ” [ImportSpec](#ImportSpec) “} ” ) )

202. ImportKeyword ::= “import”

203. AllWithExcepts ::= [AllKeyword](#AllKeyword) [ [ExceptsDef](#ExceptsDef) ]

204. ExceptsDef ::= [ExceptKeyword](#ExceptKeyword) “{ ” [ExceptSpec](#ExceptSpec) “} ”

205. ExceptKeyword ::= “except”

206. ExceptSpec ::= { [ExceptElement](#ExceptElement) [ [SemiColon](#SemiColon) ] }

207. ExceptElement ::= [ExceptGroupSpec](#ExceptGroupSpec) | [ExceptTypeDefSpec](#ExceptTypeDefSpec) | [ExceptTemplateSpec](#ExceptTemplateSpec) | [ExceptConstSpec](#ExceptConstSpec) | [ExceptTestcaseSpec](#ExceptTestcaseSpec) | [ExceptAltstepSpec](#ExceptAltstepSpec) | [ExceptFunctionSpec](#ExceptFunctionSpec) | [ExceptSignatureSpec](#ExceptSignatureSpec) | [ExceptModuleParSpec](#ExceptModuleParSpec)

208. ExceptGroupSpec ::= [GroupKeyword](#GroupKeyword) ( [QualifiedIdentifierList](#QualifiedIdentifierList) | [AllKeyword](#AllKeyword) )

209. IdentifierListOrAll ::= [IdentifierList](#IdentifierList) | [AllKeyword](#AllKeyword)

210. ExceptTypeDefSpec ::= [TypeDefKeyword](#TypeDefKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

211. ExceptTemplateSpec ::= [TemplateKeyword](#TemplateKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

212. ExceptConstSpec ::= [ConstKeyword](#ConstKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

213. ExceptTestcaseSpec ::= [TestcaseKeyword](#TestcaseKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

214. ExceptAltstepSpec ::= [AltstepKeyword](#AltstepKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

215. ExceptFunctionSpec ::= [FunctionKeyword](#FunctionKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

216. ExceptSignatureSpec ::= [SignatureKeyword](#SignatureKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

217. ExceptModuleParSpec ::= [ModuleParKeyword](#ModuleParKeyword) [IdentifierListOrAll](#IdentifierListOrAll)

218. ImportSpec ::= { [ImportElement](#ImportElement) [ [SemiColon](#SemiColon) ] }

219. ImportElement ::= [ImportGroupSpec](#ImportGroupSpec) | [ImportTypeDefSpec](#ImportTypeDefSpec) | [ImportTemplateSpec](#ImportTemplateSpec) | [ImportConstSpec](#ImportConstSpec) | [ImportTestcaseSpec](#ImportTestcaseSpec) | [ImportAltstepSpec](#ImportAltstepSpec) | [ImportFunctionSpec](#ImportFunctionSpec) | [ImportSignatureSpec](#ImportSignatureSpec) | [ImportModuleParSpec](#ImportModuleParSpec) | [ImportImportSpec](#ImportImportSpec)

220. ImportFromSpec ::= [FromKeyword](#FromKeyword) [ModuleId](#ModuleId) [ [RecursiveKeyword](#RecursiveKeyword) ]

221. RecursiveKeyword ::= “recursive”

222. ImportGroupSpec ::= [GroupKeyword](#GroupKeyword) ( [GroupRefListWithExcept](#GroupRefListWithExcept) | [AllGroupsWithExcept](#AllGroupsWithExcept) )

223. GroupRefListWithExcept ::= [QualifiedIdentifierWithExcept](#QualifiedIdentifierWithExcept) { “,” [QualifiedIdentifierWithExcept](#QualifiedIdentifierWithExcept) }

224. AllGroupsWithExcept ::= [AllKeyword](#AllKeyword) [ [ExceptKeyword](#ExceptKeyword) [QualifiedIdentifierList](#QualifiedIdentifierList) ]

225. QualifiedIdentifierWithExcept ::= [QualifiedIdentifier](#QualifiedIdentifier) [ [ExceptsDef](#ExceptsDef) ]

226. IdentifierListOrAllWithExcept ::= [IdentifierList](#IdentifierList) | [AllWithExcept](#AllWithExcept)

227. ImportTypeDefSpec ::= [TypeDefKeyword](#TypeDefKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

228. AllWithExcept ::= [AllKeyword](#AllKeyword) [ [ExceptKeyword](#ExceptKeyword) [IdentifierList](#IdentifierList) ]

229. ImportTemplateSpec ::= [TemplateKeyword](#TemplateKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

230. ImportConstSpec ::= [ConstKeyword](#ConstKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

231. ImportAltstepSpec ::= [AltstepKeyword](#AltstepKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

232. ImportTestcaseSpec ::= [TestcaseKeyword](#TestcaseKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

233. ImportFunctionSpec ::= [FunctionKeyword](#FunctionKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

234. ImportSignatureSpec ::= [SignatureKeyword](#SignatureKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

235. ImportModuleParSpec ::= [ModuleParKeyword](#ModuleParKeyword) [IdentifierListOrAllWithExcept](#IdentifierListOrAllWithExcept)

236. ImportImportSpec ::= [ImportKeyword](#ImportKeyword) [AllKeyword](#AllKeyword)

A.1.6.1.9 Group definitions

237. GroupDef ::= [GroupKeyword](#GroupKeyword) [Identifier](#Identifier) “{ ” [ [ModuleDefinitionsList](#ModuleDefinitionsList) ] “} ”

238. GroupKeyword ::= “group”

A.1.6.1.10 External function definitions

239. ExtFunctionDef ::= [ExtKeyword](#ExtKeyword) [FunctionKeyword](#FunctionKeyword) [ [DeterministicModifier](#DeterministicModifier) ] [Identifier](#Identifier) “(” [ [FunctionFormalParList](#FunctionFormalParList) ] “)” [ [ReturnType](#ReturnType) ]

240. ExtKeyword ::= “external”

A.1.6.1.11 External constant definitions

241. ExtConstDef ::= [ExtKeyword](#ExtKeyword) [ConstKeyword](#ConstKeyword) [Type](#Type) [IdentifierList](#IdentifierList)

A.1.6.1.12 Module parameter definitions

242. ModuleParDef ::= [ModuleParKeyword](#ModuleParKeyword) ( [ModulePar](#ModulePar) | ( “{ ” [MultitypedModuleParList](#MultitypedModuleParList) “} ” ) )

243. ModuleParKeyword ::= “modulepar”

244. MultitypedModuleParList ::= { [ModulePar](#ModulePar) [ [SemiColon](#SemiColon) ] }

245. ModulePar ::= [Type](#Type) [ModuleParList](#ModuleParList)

246. ModuleParList ::= [Identifier](#Identifier) [ [AssignmentChar](#AssignmentChar) [ConstantExpression](#ConstantExpression) ] { “,” [Identifier](#Identifier) [ [AssignmentChar](#AssignmentChar) [ConstantExpression](#ConstantExpression) ] }

A.1.6.1.13 Friend module definitions

247. FriendModuleDef ::= “friend” “module” [IdentifierList](#IdentifierList) [ [SemiColon](#SemiColon) ]

A.1.6.2 Control part

248. ModuleControlPart ::= [ControlKeyword](#ControlKeyword) “{ ” [ModuleControlBody](#ModuleControlBody) “} ” [ [WithStatement](#WithStatement) ] [ [SemiColon](#SemiColon) ]

249. ControlKeyword ::= “control”

250. ModuleControlBody ::= [ [ControlStatementOrDefList](#ControlStatementOrDefList) ]

251. ControlStatementOrDefList ::= { [ControlStatementOrDef](#ControlStatementOrDef) [ [SemiColon](#SemiColon) ] } +

252. ControlStatementOrDef ::= ( [FunctionLocalDef](#FunctionLocalDef) | [FunctionLocalInst](#FunctionLocalInst) ) [ [WithStatement](#WithStatement) ] | [ControlStatement](#ControlStatement)

253. ControlStatement ::= [TimerStatements](#TimerStatements) | [BasicStatements](#BasicStatements) | [BehaviourStatements](#BehaviourStatements) | [SUTStatements](#SUTStatements) | [StopKeyword](#StopKeyword)

A.1.6.3 Local definitions

A.1.6.3.1 Variable instantiation

254. VarInstance ::= [VarKeyword](#VarKeyword) ( ( [ [LazyModifier](#LazyModifier) | [FuzzyModifier](#FuzzyModifier) ] [Type](#Type) [VarList](#VarList) ) | ( ( [TemplateKeyword](#TemplateKeyword) | [RestrictedTemplate](#RestrictedTemplate) ) [ [LazyModifier](#LazyModifier) | [FuzzyModifier](#FuzzyModifier) ] [Type](#Type) [TempVarList](#TempVarList) ) )

255. VarList ::= [SingleVarInstance](#SingleVarInstance) { “,” [SingleVarInstance](#SingleVarInstance) }

256. SingleVarInstance ::= [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ] [ [AssignmentChar](#AssignmentChar) [Expression](#Expression) ]

257. VarKeyword ::= “var”

258. TempVarList ::= [SingleTempVarInstance](#SingleTempVarInstance) { “,” [SingleTempVarInstance](#SingleTempVarInstance) }

259. SingleTempVarInstance ::= [Identifier](#Identifier) [ [ArrayDef](#ArrayDef) ] [ [AssignmentChar](#AssignmentChar) [TemplateBody](#TemplateBody) ]

260. VariableRef ::= [Identifier](#Identifier) [ [ExtendedFieldReference](#ExtendedFieldReference) ]

A.1.6.3.2 Timer instantiation

261. TimerInstance ::= [TimerKeyword](#TimerKeyword) [VarList](#VarList)

262. TimerKeyword ::= “timer”

263. ArrayIdentifierRef ::= [Identifier](#Identifier) { [ArrayOrBitRef](#ArrayOrBitRef) }

A.1.6.4 Operations

A.1.6.4.1 Component operations

264. ConfigurationStatements ::= [ConnectStatement](#ConnectStatement) | [MapStatement](#MapStatement) | [DisconnectStatement](#DisconnectStatement) | [UnmapStatement](#UnmapStatement) | [DoneStatement](#DoneStatement) | [KilledStatement](#KilledStatement) | [StartTCStatement](#StartTCStatement) | [StopTCStatement](#StopTCStatement) | [KillTCStatement](#KillTCStatement)

265. ConfigurationOps ::= [CreateOp](#CreateOp) | [SelfOp](#SelfOp) | [SystemKeyword](#SystemKeyword) | [MTCKeyword](#MTCKeyword) | [RunningOp](#RunningOp) | [AliveOp](#AliveOp)

266. CreateOp ::= [ComponentType](#ComponentType) [Dot](#Dot) [CreateKeyword](#CreateKeyword) [ “(” ( [SingleExpression](#SingleExpression) | [Minus](#Minus) ) [ “,” [SingleExpression](#SingleExpression) ] “)” ] [ [AliveKeyword](#AliveKeyword) ]

267. SelfOp ::= “self”

268. DoneStatement ::= [ComponentOrAny](#ComponentOrAny) [Dot](#Dot) [DoneKeyword](#DoneKeyword) [ [PortRedirectSymbol](#PortRedirectSymbol) [ [ValueStoreSpec](#ValueStoreSpec) ] [ [IndexSpec](#IndexSpec) ] ]

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

269. ComponentOrAny ::= [ComponentOrDefaultReference](#ComponentOrDefaultReference) | ( [AnyKeyword](#AnyKeyword) ( [ComponentKeyword](#ComponentKeyword) | [FromKeyword](#FromKeyword) [VariableRef](#VariableRef) ) ) | ( [AllKeyword](#AllKeyword) [ComponentKeyword](#ComponentKeyword) )

270. ValueStoreSpec ::= [ValueKeyword](#ValueKeyword) [VariableRef](#VariableRef)

271. IndexAssignment ::= [PortRedirectSymbol](#PortRedirectSymbol) [IndexSpec](#IndexSpec)

272. IndexSpec ::= [IndexModifier](#IndexModifier) [ValueStoreSpec](#ValueStoreSpec)

273. KilledStatement ::= [ComponentOrAny](#ComponentOrAny) [Dot](#Dot) [KilledKeyword](#KilledKeyword) [ [PortRedirectSymbol](#PortRedirectSymbol) [ [ValueStoreSpec](#ValueStoreSpec) ] [ [IndexSpec](#IndexSpec) ] ]

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

274. DoneKeyword ::= “done”

275. KilledKeyword ::= “killed”

276. RunningOp ::= [ComponentOrAny](#ComponentOrAny) [Dot](#Dot) [RunningKeyword](#RunningKeyword) [ [IndexAssignment](#IndexAssignment) ]

277. RunningKeyword ::= “running”

278. AliveOp ::= [ComponentOrAny](#ComponentOrAny) [Dot](#Dot) [AliveKeyword](#AliveKeyword) [ [IndexAssignment](#IndexAssignment) ]

279. CreateKeyword ::= “create”

280. AliveKeyword ::= “alive”

281. ConnectStatement ::= [ConnectKeyword](#ConnectKeyword) [SingleConnectionSpec](#SingleConnectionSpec)

282. ConnectKeyword ::= “connect”

283. SingleConnectionSpec ::= “(” [PortRef](#PortRef) “,” [PortRef](#PortRef) “)”

284. PortRef ::= [ComponentRef](#ComponentRef) [Colon](#Colon) [ArrayIdentifierRef](#ArrayIdentifierRef)

285. ComponentRef ::= [ComponentOrDefaultReference](#ComponentOrDefaultReference) | [SystemKeyword](#SystemKeyword) | [SelfOp](#SelfOp) | [MTCKeyword](#MTCKeyword)

286. ComponentRefAssignment ::= [Identifier](#Identifier) “:=” [ComponentRef](#ComponentRef)

287. DisconnectStatement ::= [DisconnectKeyword](#DisconnectKeyword) [ [SingleConnectionSpec](#SingleConnectionSpec) | [AllConnectionsSpec](#AllConnectionsSpec) | [AllPortsSpec](#AllPortsSpec) | [AllCompsAllPortsSpec](#AllCompsAllPortsSpec) ]

288. AllConnectionsSpec ::= “(” [PortRef](#PortRef) “)”

289. AllPortsSpec ::= “(” [ComponentRef](#ComponentRef) “:” [AllKeyword](#AllKeyword) [PortKeyword](#PortKeyword) “)”

290. AllCompsAllPortsSpec ::= “(” [AllKeyword](#AllKeyword) [ComponentKeyword](#ComponentKeyword) “:” [AllKeyword](#AllKeyword) [PortKeyword](#PortKeyword) “)”

291. DisconnectKeyword ::= “disconnect”

292. MapStatement ::= [MapKeyword](#MapKeyword) [SingleConnectionSpec](#SingleConnectionSpec) [ [ParamClause](#ParamClause) ]

293. ParamClause ::= [ParamKeyword](#ParamKeyword) [FunctionActualParList](#FunctionActualParList)

294. MapKeyword ::= “map”

295. UnmapStatement ::= [UnmapKeyword](#UnmapKeyword) [ [SingleConnectionSpec](#SingleConnectionSpec) [ [ParamClause](#ParamClause) ] | [AllConnectionsSpec](#AllConnectionsSpec) [ [ParamClause](#ParamClause) ] | [AllPortsSpec](#AllPortsSpec) | [AllCompsAllPortsSpec](#AllCompsAllPortsSpec) ]

296. UnmapKeyword ::= “unmap”

297. StartTCStatement ::= [ComponentOrDefaultReference](#ComponentOrDefaultReference) [Dot](#Dot) [StartKeyword](#StartKeyword) “(” ( [FunctionInstance](#FunctionInstance) | [AltstepInstance](#AltstepInstance) ) “)”

298. StartKeyword ::= “start”

299. StopTCStatement ::= [StopKeyword](#StopKeyword) | ( [ComponentReferenceOrLiteral](#ComponentReferenceOrLiteral) | [AllKeyword](#AllKeyword) [ComponentKeyword](#ComponentKeyword) ) [Dot](#Dot) [StopKeyword](#StopKeyword)

300. ComponentReferenceOrLiteral ::= [ComponentOrDefaultReference](#ComponentOrDefaultReference) | [MTCKeyword](#MTCKeyword) | [SelfOp](#SelfOp)

301. KillTCStatement ::= [KillKeyword](#KillKeyword) | ( ( [ComponentReferenceOrLiteral](#ComponentReferenceOrLiteral) | [AllKeyword](#AllKeyword) [ComponentKeyword](#ComponentKeyword) ) [Dot](#Dot) [KillKeyword](#KillKeyword) )

302. ComponentOrDefaultReference ::= [VariableRef](#VariableRef) | [FunctionInstance](#FunctionInstance)

303. KillKeyword ::= “kill”

A.1.6.4.2 Port operations

304. CommunicationStatements ::= [SendStatement](#SendStatement) | [CallStatement](#CallStatement) | [ReplyStatement](#ReplyStatement) | [RaiseStatement](#RaiseStatement) | [ReceiveStatement](#ReceiveStatement) | [TriggerStatement](#TriggerStatement) | [GetCallStatement](#GetCallStatement) | [GetReplyStatement](#GetReplyStatement) | [CatchStatement](#CatchStatement) | [CheckStatement](#CheckStatement) | [ClearStatement](#ClearStatement) | [StartStatement](#StartStatement) | [StopStatement](#StopStatement) | [HaltStatement](#HaltStatement) | [CheckStateStatement](#CheckStateStatement)

305. SendStatement ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [PortSendOp](#PortSendOp)

306. PortSendOp ::= [SendOpKeyword](#SendOpKeyword) “(” [TemplateInstance](#TemplateInstance) “)” [ [ToClause](#ToClause) ]

307. SendOpKeyword ::= “send”

308. ToClause ::= [ToKeyword](#ToKeyword) ( [TemplateInstance](#TemplateInstance) | [AddressRefList](#AddressRefList) | [AllKeyword](#AllKeyword) [ComponentKeyword](#ComponentKeyword) )

309. AddressRefList ::= “(” [TemplateInstance](#TemplateInstance) { “,” [TemplateInstance](#TemplateInstance) } “)”

310. ToKeyword ::= “to”

311. CallStatement ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [PortCallOp](#PortCallOp) [ [PortCallBody](#PortCallBody) ]

312. PortCallOp ::= [CallOpKeyword](#CallOpKeyword) “(” [CallParameters](#CallParameters) “)” [ [ToClause](#ToClause) ]

313. CallOpKeyword ::= “call”

314. CallParameters ::= [TemplateInstance](#TemplateInstance) [ “,” [CallTimerValue](#CallTimerValue) ]

315. CallTimerValue ::= [Expression](#Expression) | [NowaitKeyword](#NowaitKeyword)

316. NowaitKeyword ::= “nowait”

317. PortCallBody ::= “{ ” [CallBodyStatementList](#CallBodyStatementList) “} ”

318. CallBodyStatementList ::= { [CallBodyStatement](#CallBodyStatement) [ [SemiColon](#SemiColon) ] } +

319. CallBodyStatement ::= [CallBodyGuard](#CallBodyGuard) [StatementBlock](#StatementBlock)

320. CallBodyGuard ::= [AltGuardChar](#AltGuardChar) [CallBodyOps](#CallBodyOps)

321. CallBodyOps ::= [GetReplyStatement](#GetReplyStatement) | [CatchStatement](#CatchStatement)

322. ReplyStatement ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [PortReplyOp](#PortReplyOp)

323. PortReplyOp ::= [ReplyKeyword](#ReplyKeyword) “(” [TemplateInstance](#TemplateInstance) [ [ReplyValue](#ReplyValue) ] “)” [ [ToClause](#ToClause) ]

324. ReplyKeyword ::= “reply”

325. ReplyValue ::= [ValueKeyword](#ValueKeyword) [TemplateBody](#TemplateBody)

/\* 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. \*/

326. RaiseStatement ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [PortRaiseOp](#PortRaiseOp)

327. PortRaiseOp ::= [RaiseKeyword](#RaiseKeyword) “(” [Signature](#Signature) “,” [TemplateInstance](#TemplateInstance) “)” [ [ToClause](#ToClause) ]

328. RaiseKeyword ::= “raise”

329. ReceiveStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortReceiveOp](#PortReceiveOp)

330. PortOrAny ::= [ArrayIdentifierRef](#ArrayIdentifierRef) | ( [AnyKeyword](#AnyKeyword) ( [PortKeyword](#PortKeyword) | [FromKeyword](#FromKeyword) [VariableRef](#VariableRef) ) )

331. PortReceiveOp ::= [ReceiveOpKeyword](#ReceiveOpKeyword) [ “(” [TemplateInstance](#TemplateInstance) “)” ] [ [FromClause](#FromClause) ] [ [PortRedirect](#PortRedirect) ]

332. ReceiveOpKeyword ::= “receive”

333. FromClause ::= [FromKeyword](#FromKeyword) ( [TemplateInstance](#TemplateInstance) | [AddressRefList](#AddressRefList) | [AnyKeyword](#AnyKeyword) [ComponentKeyword](#ComponentKeyword) )

334. FromKeyword ::= “from”

335. PortRedirect ::= [PortRedirectSymbol](#PortRedirectSymbol) ( ( [ValueSpec](#ValueSpec) [ [SenderSpec](#SenderSpec) ] [ [IndexSpec](#IndexSpec) ] ) | ( [SenderSpec](#SenderSpec) [ [IndexSpec](#IndexSpec) ] ) | [IndexSpec](#IndexSpec) )

336. PortRedirectSymbol ::= “->”

337. ValueSpec ::= [ValueKeyword](#ValueKeyword) ( [VariableRef](#VariableRef) | ( “(” [SingleValueSpec](#SingleValueSpec) { “,” [SingleValueSpec](#SingleValueSpec) } “)” ) )

338. SingleValueSpec ::= [VariableRef](#VariableRef) [ [AssignmentChar](#AssignmentChar) [ [DecodedModifier](#DecodedModifier) [ “(” [ [Expression](#Expression) ] “)” ] ] [FieldReference](#FieldReference) [ExtendedFieldReference](#ExtendedFieldReference) ]

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

339. ValueKeyword ::= “value”

340. SenderSpec ::= [SenderKeyword](#SenderKeyword) [VariableRef](#VariableRef)

341. SenderKeyword ::= “sender”

342. TriggerStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortTriggerOp](#PortTriggerOp)

343. PortTriggerOp ::= [TriggerOpKeyword](#TriggerOpKeyword) [ “(” [TemplateInstance](#TemplateInstance) “)” ] [ [FromClause](#FromClause) ] [ [PortRedirect](#PortRedirect) ]

344. TriggerOpKeyword ::= “trigger”

345. GetCallStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortGetCallOp](#PortGetCallOp)

346. PortGetCallOp ::= [GetCallOpKeyword](#GetCallOpKeyword) [ “(” [TemplateInstance](#TemplateInstance) “)” ] [ [FromClause](#FromClause) ] [ [PortRedirectWithParam](#PortRedirectWithParam) ]

347. GetCallOpKeyword ::= “getcall”

348. PortRedirectWithParam ::= [PortRedirectSymbol](#PortRedirectSymbol) [RedirectWithParamSpec](#RedirectWithParamSpec)

349. RedirectWithParamSpec ::= ( [ParamSpec](#ParamSpec) [ [SenderSpec](#SenderSpec) ] [ [IndexSpec](#IndexSpec) ] ) | ( [SenderSpec](#SenderSpec) [ [IndexSpec](#IndexSpec) ] ) | [IndexSpec](#IndexSpec)

350. ParamSpec ::= [ParamKeyword](#ParamKeyword) [ParamAssignmentList](#ParamAssignmentList)

351. ParamKeyword ::= “param”

352. ParamAssignmentList ::= “(” ( [AssignmentList](#AssignmentList) | [VariableList](#VariableList) ) “)”

353. AssignmentList ::= [VariableAssignment](#VariableAssignment) { “,” [VariableAssignment](#VariableAssignment) }

354. VariableAssignment ::= [VariableRef](#VariableRef) [AssignmentChar](#AssignmentChar) [ [DecodedModifier](#DecodedModifier) [ “(” [Expression](#Expression) ] “)” ] [Identifier](#Identifier)

355. VariableList ::= [VariableEntry](#VariableEntry) { “,” [VariableEntry](#VariableEntry) }

356. VariableEntry ::= [VariableRef](#VariableRef) | [Minus](#Minus)

357. GetReplyStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortGetReplyOp](#PortGetReplyOp)

358. PortGetReplyOp ::= [GetReplyOpKeyword](#GetReplyOpKeyword) [ “(” [TemplateInstance](#TemplateInstance) [ [ValueMatchSpec](#ValueMatchSpec) ] “)” ] [ [FromClause](#FromClause) ] [ [PortRedirectWithValueAndParam](#PortRedirectWithValueAndParam) ]

359. PortRedirectWithValueAndParam ::= [PortRedirectSymbol](#PortRedirectSymbol) [RedirectWithValueAndParamSpec](#RedirectWithValueAndParamSpec)

360. RedirectWithValueAndParamSpec ::= ( [ValueSpec](#ValueSpec) [ [ParamSpec](#ParamSpec) ] [ [SenderSpec](#SenderSpec) ] [ [IndexSpec](#IndexSpec) ] ) | [RedirectWithParamSpec](#RedirectWithParamSpec)

361. GetReplyOpKeyword ::= “getreply”

362. ValueMatchSpec ::= [ValueKeyword](#ValueKeyword) [TemplateInstance](#TemplateInstance)

363. CheckStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortCheckOp](#PortCheckOp)

364. PortCheckOp ::= [CheckOpKeyword](#CheckOpKeyword) [ “(” [CheckParameter](#CheckParameter) “)” ]

365. CheckOpKeyword ::= “check”

366. CheckParameter ::= [CheckPortOpsPresent](#CheckPortOpsPresent) | [FromClausePresent](#FromClausePresent) | [RedirectPresent](#RedirectPresent)

367. FromClausePresent ::= [FromClause](#FromClause) [ [PortRedirectSymbol](#PortRedirectSymbol) ( ( [SenderSpec](#SenderSpec) [ [IndexSpec](#IndexSpec) ] ) | [IndexSpec](#IndexSpec) ) ]

368. RedirectPresent ::= [PortRedirectSymbol](#PortRedirectSymbol) ( ( [SenderSpec](#SenderSpec) [ [IndexSpec](#IndexSpec) ] ) | [IndexSpec](#IndexSpec) )

369. CheckPortOpsPresent ::= [PortReceiveOp](#PortReceiveOp) | [PortGetCallOp](#PortGetCallOp) | [PortGetReplyOp](#PortGetReplyOp) | [PortCatchOp](#PortCatchOp)

370. CatchStatement ::= [PortOrAny](#PortOrAny) [Dot](#Dot) [PortCatchOp](#PortCatchOp)

371. PortCatchOp ::= [CatchOpKeyword](#CatchOpKeyword) [ “(” [CatchOpParameter](#CatchOpParameter) “)” ] [ [FromClause](#FromClause) ] [ [PortRedirect](#PortRedirect) ]

372. CatchOpKeyword ::= “catch”

373. CatchOpParameter ::= [Signature](#Signature) “,” [TemplateInstance](#TemplateInstance) | [TimeoutKeyword](#TimeoutKeyword)

374. ClearStatement ::= [PortOrAll](#PortOrAll) [Dot](#Dot) [ClearOpKeyword](#ClearOpKeyword)

375. PortOrAll ::= [ArrayIdentifierRef](#ArrayIdentifierRef) | [AllKeyword](#AllKeyword) [PortKeyword](#PortKeyword)

376. ClearOpKeyword ::= “clear”

377. StartStatement ::= [PortOrAll](#PortOrAll) [Dot](#Dot) [StartKeyword](#StartKeyword)

378. StopStatement ::= [PortOrAll](#PortOrAll) [Dot](#Dot) [StopKeyword](#StopKeyword)

379. StopKeyword ::= “stop”

380. HaltStatement ::= [PortOrAll](#PortOrAll) [Dot](#Dot) [HaltKeyword](#HaltKeyword)

381. HaltKeyword ::= “halt”

382. AnyKeyword ::= “any”

383. CheckStateStatement ::= [PortOrAllAny](#PortOrAllAny) [Dot](#Dot) [CheckStateKeyword](#CheckStateKeyword) “(” [SingleExpression](#SingleExpression) “)”

384. PortOrAllAny ::= [PortOrAll](#PortOrAll) | [AnyKeyword](#AnyKeyword) [PortKeyword](#PortKeyword)

385. CheckStateKeyword ::= “checkstate”

A.1.6.4.3 Timer operations

386. TimerStatements ::= [StartTimerStatement](#StartTimerStatement) | [StopTimerStatement](#StopTimerStatement) | [TimeoutStatement](#TimeoutStatement)

387. TimerOps ::= [ReadTimerOp](#ReadTimerOp) | [RunningTimerOp](#RunningTimerOp)

388. StartTimerStatement ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [StartKeyword](#StartKeyword) [ “(” [Expression](#Expression) “)” ]

389. StopTimerStatement ::= [TimerRefOrAll](#TimerRefOrAll) [Dot](#Dot) [StopKeyword](#StopKeyword)

390. TimerRefOrAll ::= [ArrayIdentifierRef](#ArrayIdentifierRef) | [AllKeyword](#AllKeyword) [TimerKeyword](#TimerKeyword)

391. ReadTimerOp ::= [ArrayIdentifierRef](#ArrayIdentifierRef) [Dot](#Dot) [ReadKeyword](#ReadKeyword)

392. ReadKeyword ::= “read”

393. RunningTimerOp ::= [TimerRefOrAny](#TimerRefOrAny) [Dot](#Dot) [RunningKeyword](#RunningKeyword) [ [IndexAssignment](#IndexAssignment) ]

394. TimeoutStatement ::= [TimerRefOrAny](#TimerRefOrAny) [Dot](#Dot) [TimeoutKeyword](#TimeoutKeyword) [ [IndexAssignment](#IndexAssignment) ]

395. TimerRefOrAny ::= [ArrayIdentifierRef](#ArrayIdentifierRef) | ( [AnyKeyword](#AnyKeyword) [TimerKeyword](#TimerKeyword) ) | ( [AnyKeyword](#AnyKeyword) [FromKeyword](#FromKeyword) [Identifier](#Identifier) )

396. TimeoutKeyword ::= “timeout”

A.1.6.4.4 Testcase operation

397. TestcaseOperation ::= [TestcaseKeyword](#TestcaseKeyword) “.” [StopKeyword](#StopKeyword) [ “(” { [LogItem](#LogItem) [ “,” ] } “)” ]

A.1.6.5 Type

398. Type ::= [PredefinedType](#PredefinedType) | [ReferencedType](#ReferencedType)

399. PredefinedType ::= [BitStringKeyword](#BitStringKeyword) | [BooleanKeyword](#BooleanKeyword) | [CharStringKeyword](#CharStringKeyword) | [UniversalCharString](#UniversalCharString) | [IntegerKeyword](#IntegerKeyword) | [OctetStringKeyword](#OctetStringKeyword) | [HexStringKeyword](#HexStringKeyword) | [VerdictTypeKeyword](#VerdictTypeKeyword) | [FloatKeyword](#FloatKeyword) | [AddressKeyword](#AddressKeyword) | [DefaultKeyword](#DefaultKeyword) | [AnyTypeKeyword](#AnyTypeKeyword)

400. BitStringKeyword ::= “bitstring”

401. BooleanKeyword ::= “boolean”

402. IntegerKeyword ::= “integer”

403. OctetStringKeyword ::= “octetstring”

404. HexStringKeyword ::= “hexstring”

405. VerdictTypeKeyword ::= “verdicttype”

406. FloatKeyword ::= “float”

407. AddressKeyword ::= “address”

408. DefaultKeyword ::= “default”

409. AnyTypeKeyword ::= “anytype”

410. CharStringKeyword ::= “charstring”

411. UniversalCharString ::= [UniversalKeyword](#UniversalKeyword) [CharStringKeyword](#CharStringKeyword)

412. UniversalKeyword ::= “universal”

413. ReferencedType ::= [ExtendedIdentifier](#ExtendedIdentifier) [ [ExtendedFieldReference](#ExtendedFieldReference) ]

414. TypeReference ::= [ExtendedIdentifier](#ExtendedIdentifier)

415. ArrayDef ::= { “[” [SingleExpression](#SingleExpression) [ “..” [SingleExpression](#SingleExpression) ] “]” } +

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

A.1.6.6 Value

416. Value ::= [PredefinedValue](#PredefinedValue) | [ReferencedValue](#ReferencedValue)

417. PredefinedValue ::= [Bstring](#Bstring) | [BooleanValue](#BooleanValue) | [CharStringValue](#CharStringValue) | [Number](#Number) | [Ostring](#Ostring) | [Hstring](#Hstring) | [VerdictTypeValue](#VerdictTypeValue) | [FloatValue](#FloatValue) | [AddressValue](#AddressValue) | [OmitKeyword](#OmitKeyword)

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

418. BooleanValue ::= “true” | “false”

419. VerdictTypeValue ::= “pass” | “fail” | “inconc” | “none” | “error”

420. CharStringValue ::= [Cstring](#Cstring) | [Quadruple](#Quadruple) | [USIlikeNotation](#USIlikeNotation)

421. Quadruple ::= [CharKeyword](#CharKeyword) “(” [Number](#Number) “,” [Number](#Number) “,” [Number](#Number) “,” [Number](#Number) “)”

422. USIlikeNotation ::= [CharKeyword](#CharKeyword) “(” [UIDlike](#UIDlike) { “,” [UIDlike](#UIDlike) } “)”

423. UIDlike ::= ( “U” | “u” ) { “+” } { [Hex](#Hex) } #(1,8)

424. CharKeyword ::= “char”

425. FloatValue ::= [FloatDotNotation](#FloatDotNotation) | [FloatENotation](#FloatENotation) | [NaNKeyword](#NaNKeyword)

426. NaNKeyword ::= “not\_a\_number”

427. FloatDotNotation ::= [Number](#Number) [Dot](#Dot) [DecimalNumber](#DecimalNumber)

428. FloatENotation ::= [Number](#Number) [ [Dot](#Dot) [DecimalNumber](#DecimalNumber) ] [Exponential](#Exponential) [ [Minus](#Minus) ] [Number](#Number)

429. Exponential ::= “E”

430. ReferencedValue ::= [ExtendedIdentifier](#ExtendedIdentifier) [ [ExtendedFieldReference](#ExtendedFieldReference) | [ExtendedEnumReference](#ExtendedEnumReference) ]

/\*\* STATIC Semantics: ExtendedEnumReference shall be present if and only if ExtendedIdentifier refers to an enumerated value with an attached value list \*/

431. ExtendedEnumReference ::= “(” [IntegerValue](#IntegerValue) “)”

432. Number ::= ( [NonZeroNum](#NonZeroNum) { [Num](#Num) } ) | “0”

433. NonZeroNum ::= “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

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

435. Num ::= “0” | [NonZeroNum](#NonZeroNum)

436. Bstring ::= “’” { [Bin](#Bin) | [BinSpace](#BinSpace) } “’” “B”

437. Bin ::= “0” | “1”

438. Hstring ::= “’” { [Hex](#Hex) | [BinSpace](#BinSpace) } “’” “H”

439. Hex ::= [Num](#Num) | “A” | “B” | “C” | “D” | “E” | “F” | “a” | “b” | “c” | “d” | “e” | “f”

440. Ostring ::= “’” { [Oct](#Oct) | [BinSpace](#BinSpace) } “’” “O”

441. Oct ::= [Hex](#Hex) [Hex](#Hex)

442. Cstring ::= “”” { [Char](#Char) } “””

443. 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 \*/

444. Identifier ::= [Alpha](#Alpha) { [AlphaNum](#AlphaNum) | [Underscore](#Underscore) }

445. Alpha ::= [UpperAlpha](#UpperAlpha) | [LowerAlpha](#LowerAlpha)

446. AlphaNum ::= [Alpha](#Alpha) | [Num](#Num)

447. 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”

448. 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”

449. 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) \*/

450. FreeText ::= “”” { [ExtendedAlphaNum](#ExtendedAlphaNum) } “””

451. AddressValue ::= “null”

452. OmitKeyword ::= “omit”

453. BinSpace ::= “” | “\ ” [NLChar](#NLChar)

454. 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) (see Recommendation ITU T T.50 [4]) (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

455. InParKeyword ::= “in”

456. OutParKeyword ::= “out”

457. InOutParKeyword ::= “inout”

458. FormalValuePar ::= [ ( [InParKeyword](#InParKeyword) | [InOutParKeyword](#InOutParKeyword) | [OutParKeyword](#OutParKeyword) ) ] [ [LazyModifier](#LazyModifier) | [FuzzyModifier](#FuzzyModifier) ] [Type](#Type) [Identifier](#Identifier) [ “:=” ( [Expression](#Expression) | [Minus](#Minus) ) ]

459. FormalPortPar ::= [ [InOutParKeyword](#InOutParKeyword) ] [Identifier](#Identifier) [Identifier](#Identifier)

/\* The first Identifier refers to the port type. The second Identifier refers to the port parameter identifier \*/

460. FormalTimerPar ::= [ [InOutParKeyword](#InOutParKeyword) ] [TimerKeyword](#TimerKeyword) [Identifier](#Identifier)

461. FormalTemplatePar ::= [ ( [InParKeyword](#InParKeyword) | [OutParKeyword](#OutParKeyword) | [InOutParKeyword](#InOutParKeyword) ) ] ( [TemplateKeyword](#TemplateKeyword) | [RestrictedTemplate](#RestrictedTemplate) ) [ [LazyModifier](#LazyModifier) | [FuzzyModifier](#FuzzyModifier) ] [Type](#Type) [Identifier](#Identifier) [ “:=” ( [TemplateInstance](#TemplateInstance) | [Minus](#Minus) ) ]

462. RestrictedTemplate ::= [OmitKeyword](#OmitKeyword) | ( [TemplateKeyword](#TemplateKeyword) [TemplateRestriction](#TemplateRestriction) )

463. TemplateRestriction ::= “(” ( [OmitKeyword](#OmitKeyword) | [ValueKeyword](#ValueKeyword) | [PresentKeyword](#PresentKeyword) ) “)”

A.1.6.8 Statements

A.1.6.8.1 With statement

464. WithStatement ::= [WithKeyword](#WithKeyword) [WithAttribList](#WithAttribList)

465. WithKeyword ::= “with”

466. WithAttribList ::= “{ ” [MultiWithAttrib](#MultiWithAttrib) “} ”

467. MultiWithAttrib ::= { [SingleWithAttrib](#SingleWithAttrib) [ [SemiColon](#SemiColon) ] }

468. SingleWithAttrib ::= [AttribKeyword](#AttribKeyword) [ [OverrideKeyword](#OverrideKeyword) ] [ [AttribQualifier](#AttribQualifier) ] [FreeText](#FreeText)

469. AttribKeyword ::= [EncodeKeyword](#EncodeKeyword) | [VariantKeyword](#VariantKeyword) | [DisplayKeyword](#DisplayKeyword) | [ExtensionKeyword](#ExtensionKeyword) | [OptionalKeyword](#OptionalKeyword)

470. EncodeKeyword ::= “encode”

471. VariantKeyword ::= “variant”

472. DisplayKeyword ::= “display”

473. ExtensionKeyword ::= “extension”

474. OverrideKeyword ::= “override”

475. AttribQualifier ::= “(” [DefOrFieldRefList](#DefOrFieldRefList) “)”

476. DefOrFieldRefList ::= [DefOrFieldRef](#DefOrFieldRef) { “,” [DefOrFieldRef](#DefOrFieldRef) }

477. DefOrFieldRef ::= [QualifiedIdentifier](#QualifiedIdentifier) | ( ( [FieldReference](#FieldReference) | “[” [Minus](#Minus) “]” ) [ [ExtendedFieldReference](#ExtendedFieldReference) ] ) | [AllRef](#AllRef)

478. QualifiedIdentifier ::= { [Identifier](#Identifier) [Dot](#Dot) } [Identifier](#Identifier)

479. AllRef ::= ( [GroupKeyword](#GroupKeyword) [AllKeyword](#AllKeyword) [ [ExceptKeyword](#ExceptKeyword) “{ ” [QualifiedIdentifierList](#QualifiedIdentifierList) “} ” ] ) | ( ( [TypeDefKeyword](#TypeDefKeyword) | [TemplateKeyword](#TemplateKeyword) | [ConstKeyword](#ConstKeyword) | [AltstepKeyword](#AltstepKeyword) | [TestcaseKeyword](#TestcaseKeyword) | [FunctionKeyword](#FunctionKeyword) | [SignatureKeyword](#SignatureKeyword) | [ModuleParKeyword](#ModuleParKeyword) ) [AllKeyword](#AllKeyword) [ [ExceptKeyword](#ExceptKeyword) “{ ” [IdentifierList](#IdentifierList) “} ” ] )

A.1.6.8.2 Behaviour statements

480. BehaviourStatements ::= [TestcaseInstance](#TestcaseInstance) | [FunctionInstance](#FunctionInstance) | [ReturnStatement](#ReturnStatement) | [AltConstruct](#AltConstruct) | [InterleavedConstruct](#InterleavedConstruct) | [LabelStatement](#LabelStatement) | [GotoStatement](#GotoStatement) | [RepeatStatement](#RepeatStatement) | [DeactivateStatement](#DeactivateStatement) | [AltstepInstance](#AltstepInstance) | [ActivateOp](#ActivateOp) | [BreakStatement](#BreakStatement) | [ContinueStatement](#ContinueStatement)

481. SetLocalVerdict ::= [SetVerdictKeyword](#SetVerdictKeyword) “(” [SingleExpression](#SingleExpression) { “,” [LogItem](#LogItem) } “)”

482. SetVerdictKeyword ::= “setverdict”

483. GetLocalVerdict ::= “getverdict”

484. SUTStatements ::= [ActionKeyword](#ActionKeyword) “(” [ActionText](#ActionText) { [StringOp](#StringOp) [ActionText](#ActionText) } “)”

485. ActionKeyword ::= “action”

486. ActionText ::= [FreeText](#FreeText) | [Expression](#Expression)

487. ReturnStatement ::= [ReturnKeyword](#ReturnKeyword) [ [TemplateInstance](#TemplateInstance) ]

/\* 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. \*/

488. AltConstruct ::= [AltKeyword](#AltKeyword) “{ ” [AltGuardList](#AltGuardList) “} ”

489. AltKeyword ::= “alt”

490. AltGuardList ::= { [GuardStatement](#GuardStatement) | [ElseStatement](#ElseStatement) [ [SemiColon](#SemiColon) ] }

491. GuardStatement ::= [AltGuardChar](#AltGuardChar) ( [AltstepInstance](#AltstepInstance) [ [StatementBlock](#StatementBlock) ] | [GuardOp](#GuardOp) [StatementBlock](#StatementBlock) )

492. ElseStatement ::= “[” [ElseKeyword](#ElseKeyword) “]” [StatementBlock](#StatementBlock)

493. AltGuardChar ::= “[” [ [BooleanExpression](#BooleanExpression) ] “]”

494. GuardOp ::= [TimeoutStatement](#TimeoutStatement) | [ReceiveStatement](#ReceiveStatement) | [TriggerStatement](#TriggerStatement) | [GetCallStatement](#GetCallStatement) | [CatchStatement](#CatchStatement) | [CheckStatement](#CheckStatement) | [GetReplyStatement](#GetReplyStatement) | [DoneStatement](#DoneStatement) | [KilledStatement](#KilledStatement)

495. InterleavedConstruct ::= [InterleavedKeyword](#InterleavedKeyword) “{ ” [InterleavedGuardList](#InterleavedGuardList) “} ”

496. InterleavedKeyword ::= “interleave”

497. InterleavedGuardList ::= { [InterleavedGuardElement](#InterleavedGuardElement) [ [SemiColon](#SemiColon) ] } +

498. InterleavedGuardElement ::= [InterleavedGuard](#InterleavedGuard) [StatementBlock](#StatementBlock)

499. InterleavedGuard ::= “[” “]” [GuardOp](#GuardOp)

500. LabelStatement ::= [LabelKeyword](#LabelKeyword) [Identifier](#Identifier)

501. LabelKeyword ::= “label”

502. GotoStatement ::= [GotoKeyword](#GotoKeyword) [Identifier](#Identifier)

503. GotoKeyword ::= “goto”

504. RepeatStatement ::= “repeat”

505. ActivateOp ::= [ActivateKeyword](#ActivateKeyword) “(” [AltstepInstance](#AltstepInstance) “)”

506. ActivateKeyword ::= “activate”

507. DeactivateStatement ::= [DeactivateKeyword](#DeactivateKeyword) [ “(” [ComponentOrDefaultReference](#ComponentOrDefaultReference) “)” ]

508. DeactivateKeyword ::= “deactivate”

509. BreakStatement ::= “break”

510. ContinueStatement ::= “continue”

A.1.6.8.3 Basic statements

511. BasicStatements ::= [Assignment](#Assignment) | [LogStatement](#LogStatement) | [LoopConstruct](#LoopConstruct) | [ConditionalConstruct](#ConditionalConstruct) | [SelectCaseConstruct](#SelectCaseConstruct) | [StatementBlock](#StatementBlock)

512. Expression ::= [SingleExpression](#SingleExpression) | [CompoundExpression](#CompoundExpression)

513. CompoundExpression ::= [FieldExpressionList](#FieldExpressionList) | [ArrayExpression](#ArrayExpression)

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

514. FieldExpressionList ::= “{ ” [FieldExpressionSpec](#FieldExpressionSpec) { “,” [FieldExpressionSpec](#FieldExpressionSpec) } “} ”

515. FieldExpressionSpec ::= [FieldReference](#FieldReference) [AssignmentChar](#AssignmentChar) [NotUsedOrExpression](#NotUsedOrExpression)

516. ArrayExpression ::= “{ ” [ [ArrayElementExpressionList](#ArrayElementExpressionList) ] “} ”

517. ArrayElementExpressionList ::= [NotUsedOrExpression](#NotUsedOrExpression) { “,” [NotUsedOrExpression](#NotUsedOrExpression) }

518. NotUsedOrExpression ::= [Expression](#Expression) | [Minus](#Minus)

519. ConstantExpression ::= [SingleExpression](#SingleExpression) | [CompoundConstExpression](#CompoundConstExpression)

520. BooleanExpression ::= [SingleExpression](#SingleExpression)

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

521. CompoundConstExpression ::= [FieldConstExpressionList](#FieldConstExpressionList) | [ArrayConstExpression](#ArrayConstExpression)

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

522. FieldConstExpressionList ::= “{ ” [FieldConstExpressionSpec](#FieldConstExpressionSpec) { “,” [FieldConstExpressionSpec](#FieldConstExpressionSpec) } “} ”

523. FieldConstExpressionSpec ::= [FieldReference](#FieldReference) [AssignmentChar](#AssignmentChar) [ConstantExpression](#ConstantExpression)

524. ArrayConstExpression ::= “{ ” [ [ArrayElementConstExpressionList](#ArrayElementConstExpressionList) ] “} ”

525. ArrayElementConstExpressionList ::= [ConstantExpression](#ConstantExpression) { “,” [ConstantExpression](#ConstantExpression) }

526. Assignment ::= [VariableRef](#VariableRef) [AssignmentChar](#AssignmentChar) [TemplateBody](#TemplateBody)

/\* 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. \*/

527. SingleExpression ::= [XorExpression](#XorExpression) { “or” [XorExpression](#XorExpression) }

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

528. XorExpression ::= [AndExpression](#AndExpression) { “xor” [AndExpression](#AndExpression) }

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

529. AndExpression ::= [NotExpression](#NotExpression) { “and” [NotExpression](#NotExpression) }

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

530. NotExpression ::= [ “not” ] [EqualExpression](#EqualExpression)

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

531. EqualExpression ::= [RelExpression](#RelExpression) { [EqualOp](#EqualOp) [RelExpression](#RelExpression) }

/\* 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. \*/

532. RelExpression ::= [ShiftExpression](#ShiftExpression) [ [RelOp](#RelOp) [ShiftExpression](#ShiftExpression) ] | [CompoundExpression](#CompoundExpression)

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

533. ShiftExpression ::= [BitOrExpression](#BitOrExpression) { [ShiftOp](#ShiftOp) [BitOrExpression](#BitOrExpression) }

/\* 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 “ \*/

534. BitOrExpression ::= [BitXorExpression](#BitXorExpression) { “or4b” [BitXorExpression](#BitXorExpression) }

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

535. BitXorExpression ::= [BitAndExpression](#BitAndExpression) { “xor4b” [BitAndExpression](#BitAndExpression) }

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

536. BitAndExpression ::= [BitNotExpression](#BitNotExpression) { “and4b” [BitNotExpression](#BitNotExpression) }

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

537. BitNotExpression ::= [ “not4b” ] [AddExpression](#AddExpression)

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

538. AddExpression ::= [MulExpression](#MulExpression) { [AddOp](#AddOp) [MulExpression](#MulExpression) }

/\* 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. \*/

539. MulExpression ::= [UnaryExpression](#UnaryExpression) { [MultiplyOp](#MultiplyOp) [UnaryExpression](#UnaryExpression) } | [CompoundExpression](#CompoundExpression)

/\* 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. \*/

540. UnaryExpression ::= [ [UnaryOp](#UnaryOp) ] [Primary](#Primary)

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

541. Primary ::= [OpCall](#OpCall) | [Value](#Value) | “(” [SingleExpression](#SingleExpression) “)”

542. ExtendedFieldReference ::= { ( [Dot](#Dot) ( [Identifier](#Identifier) | [PredefinedType](#PredefinedType) ) ) | [ArrayOrBitRef](#ArrayOrBitRef) | ( “[” [Minus](#Minus) “]” ) } +

/\* 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. ArrayOrBitRef 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 or set of type. \*/

543. OpCall ::= [ConfigurationOps](#ConfigurationOps) | [GetLocalVerdict](#GetLocalVerdict) | [TimerOps](#TimerOps) | [TestcaseInstance](#TestcaseInstance) | ( [FunctionInstance](#FunctionInstance) [ [ExtendedFieldReference](#ExtendedFieldReference) ] ) | ( [TemplateOps](#TemplateOps) [ [ExtendedFieldReference](#ExtendedFieldReference) ] ) | [ActivateOp](#ActivateOp)

544. AddOp ::= “+” | “-” | [StringOp](#StringOp)

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

545. 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) \*/

546. UnaryOp ::= “+” | “-”

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

547. RelOp ::= “<” | “>” | “>=” | “<=”

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

548. EqualOp ::= “==” | “!=”

549. 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 \*/

550. ShiftOp ::= “<<” | “>>” | “<@” | “@>”

551. LogStatement ::= [LogKeyword](#LogKeyword) “(” [LogItem](#LogItem) { “,” [LogItem](#LogItem) } “)”

552. LogKeyword ::= “log”

553. LogItem ::= [FreeText](#FreeText) | [TemplateInstance](#TemplateInstance)

554. LoopConstruct ::= [ForStatement](#ForStatement) | [WhileStatement](#WhileStatement) | [DoWhileStatement](#DoWhileStatement)

555. ForStatement ::= [ForKeyword](#ForKeyword) “(” [Initial](#Initial) [SemiColon](#SemiColon) [BooleanExpression](#BooleanExpression) [SemiColon](#SemiColon) [Assignment](#Assignment) “)” [StatementBlock](#StatementBlock)

556. ForKeyword ::= “for”

557. Initial ::= [VarInstance](#VarInstance) | [Assignment](#Assignment)

558. WhileStatement ::= [WhileKeyword](#WhileKeyword) “(” [BooleanExpression](#BooleanExpression) “)” [StatementBlock](#StatementBlock)

559. WhileKeyword ::= “while”

560. DoWhileStatement ::= [DoKeyword](#DoKeyword) [StatementBlock](#StatementBlock) [WhileKeyword](#WhileKeyword) “(” [BooleanExpression](#BooleanExpression) “)”

561. DoKeyword ::= “do”

562. ConditionalConstruct ::= [IfKeyword](#IfKeyword) “(” [BooleanExpression](#BooleanExpression) “)” [StatementBlock](#StatementBlock) { [ElseIfClause](#ElseIfClause) } [ [ElseClause](#ElseClause) ]

563. IfKeyword ::= “if”

564. ElseIfClause ::= [ElseKeyword](#ElseKeyword) [IfKeyword](#IfKeyword) “(” [BooleanExpression](#BooleanExpression) “)” [StatementBlock](#StatementBlock)

565. ElseKeyword ::= “else”

566. ElseClause ::= [ElseKeyword](#ElseKeyword) [StatementBlock](#StatementBlock)

567. SelectCaseConstruct ::= [SelectKeyword](#SelectKeyword) [ [UnionKeyword](#UnionKeyword) ] “(” [SingleExpression](#SingleExpression) “)” [SelectCaseBody](#SelectCaseBody)

568. SelectKeyword ::= “select”

569. SelectCaseBody ::= “{ ” { [SelectCase](#SelectCase) } + [ [CaseElse](#CaseElse) ] “} ”

570. SelectCase ::= [CaseKeyword](#CaseKeyword) ( “(” [TemplateInstance](#TemplateInstance) { “,” [TemplateInstance](#TemplateInstance) } “)” | [ElseKeyword](#ElseKeyword) ) [StatementBlock](#StatementBlock)

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

571. CaseElse ::= [CaseKeyword](#CaseKeyword) [ElseKeyword](#ElseKeyword) [StatementBlock](#StatementBlock)

572. CaseKeyword ::= “case”

573. ExtendedIdentifier ::= [ [Identifier](#Identifier) [Dot](#Dot) ] [Identifier](#Identifier)

/\*\* STATIC SEMANTICS The optional Identifier Dot part shall not be used for enumerated values\*/

574. IdentifierList ::= [Identifier](#Identifier) { “,” [Identifier](#Identifier) }

575. QualifiedIdentifierList ::= [QualifiedIdentifier](#QualifiedIdentifier) { “,” [QualifiedIdentifier](#QualifiedIdentifier) }

A.1.6.9 Miscellaneous productions

576. Dot ::= “.”

577. Minus ::= “-”

578. SemiColon ::= “;”

579. Colon ::= “:”

580. Underscore ::= “\_”

581. AssignmentChar ::= “:=”

582. IndexModifier ::= “@index”

583. DeterministicModifier ::= “@deterministic”

584. LazyModifier ::= “@lazy”

585. FuzzyModifier ::= “@fuzzy”

586. CaseInsenModifier ::= “@nocase”

587. DecodedModifier ::= “@decoded”