Chapter

39

Languages Supported in the TTCN Suite

The the TTCN suite editing tools support TTCN and ASN.1 as defined in the TTCN standard. The TTCN to C compiler on the other hand has limitations on its ASN.1 coverage. See "TTCN ASN.1 BER Encoding/Decoding" on page 56 in chapter 2, *Release Notes, in the Release Guide* for further details on the restrictions that apply.

The language supported by the TTCN suite covers the whole of TTCN and the whole of ASN.1 as it is used in TTCN.

This chapter describes the TTCN and ASN.1 EBNF grammar supported by the TTCN suite and the additional static semantics supported by the TTCN suite.

The Restrictions in the TTCN Suite

The TTCN suite editing tools support TTCN and ASN.1 as defined in the TTCN standard. The TTCN To C compiler on the other hand has limitations on its ASN.1 coverage. See "TTCN ASN.1 BER Encoding/Decoding" on page 56 in chapter 2, *Release Notes, in the Release Guide* for further details on the restrictions that apply. In a few other cases, the TTCN suite has introduced syntactical restrictions, but in all such cases there are simple work-around solutions.

This section describes the modifications in the TTCN and ASN.1 grammar, e.g. the differences between the TTCN/ASN.1 standards and the language supported by the TTCN suite. Modifications meaning that some constructs will not be analyzed correctly even though they have legal syntax according to the TTCN/ASN.1 standards.

External ASN.1 Types

The analysis of an ASN.1 Value requires access to its Type. In TTCN it is possible to refer to an external ASN.1 Type and define an ASN.1 Value of that type.

Therefore an extra field is included in the ASN.1 type by reference tables. The user provides the definition of an external ASN.1 type in this field. The allowed syntax/semantic for this field is the same as for regular ASN.1 type definition. ASN.1 values of such a reference type will be analyzed according to the provided type definition.

The extra field is exported if the option Telelogic MP format is chosen. It is, though, not printed together with the ASN.1 type by reference table.

Another solution to this problem is to open the referred ASN.1 module in the Organizer and define an dependencies link to it (see "Dependencies" on page 137 in chapter 2, *The Organizer* for more information). The type definition is fetched from the referred ASN.1 module before it is analyzed. For more information, see "ASN.1 External Type/Value References" on page 1188 in chapter 27, *Analyzing TTCN Documents (on UNIX)*.

ValueList

The problem is that a ValueList constraint (in MatchingSymbol) with only one element has the same construction as a parenthesized Expression in the top level. In addition, a ConstraintValue and an Expression may appear in the same places (since Expression is allowed in the ConstraintValue).

The Analyzer assumes that a ValueList with only one element is a parenthesized Expression wherever it appears.

Example 291 ––

Assuming the Test Suite Constant C1 (of type integer) with value (2 - X), C2 (also of type integer) with value 3 * (M - K) and C3 (also of type integer) with value (3, X). Both C1 and C2 are correct both syntactically and semantically, but C3 has an illegal value because value lists are not allowed in constants.

Example 292 ––

Assume a TTCN ASP Constraint C with parameters P1 and P2 (both of type integer). The parameter P1 has the value (2 + X) and P2 has the value (2 + Y, X). Both P1 and P2 have a correct value but the value of P1 is considered to be an Expression by the Analyzer.

––

ASN.1 AnyValue

This ASN.1 construct introduces ambiguities into the language. This is because it is hard to find the place where the Type ends and the Value starts.

A restricted form of AnyValue is supported where the Type in the AnyValue must be a reference to a user defined type. This form covers all cases for ASN.1 AnyValue.

Example 293 ––

```
Assume the ASN.1 Type definition T:
SEQUENCE {f1 INTEGER, f2 LT1 DEFAULT {s1 2, s2 LT2
FALSE}}
LT1 ::= SEQUENCE {s1 INTEGER, s2 ANY}
LT2 ::= BOOLEAN
```

––

ASN.1 NamedType & NamedValue

According to the ASN.1 standard (12.5), in some notation within which a type is referenced, the type may be named. In such cases, the ASN.1 standard specifies the use of the notation NamedType which is defined as:

```
NamedType ::= identifier Type | Type | SelectionType
```

The value of a type referenced using the NamedType notation shall be defined by the notation NamedValue which is defined as:

```
NamedValue ::= identifier Value | Value
```

According to the *Information Technology -OSI- ASN.1 (UDC 681.3:621.39)* the identifier in the NamedType definition shall be mandatory. The same modification is suggested for the NamedValue.

The TTCN Analyzer do not allow use of the second alternative of either of these constructs, i.e. the naming identifiers are **not** optional.

Example 294 ----------

```
Assume that the ASN.1 type T is defined as:
SEQUENCE { x INTEGER, y < Element }
Element ::= CHOICE { x BOOLEAN, y INTEGER }
This type definition is legal. The ASN.1 NamedType
in T (x & y) shall be unique (y is considered as a
NamedType in this case).
A legal value for T is: {x 2, y 4}
```

Example 295 ----------

```
Assume another ASN.1 type, T', defined as:
SEQUENCE { x INTEGER, y x < Element }
Element ::= CHOICE { x INTEGER, y BOOLEAN }
This type definition is also legal. The ASN.1
NamedType in T' (x & y) shall be unique (x is not
considered as a NamedType in this case).
The value {x 2, y 4} is not considered to be a legal
value for T' even though it is allowed according to
the ASN.1 standard. A legal value (which will be
accepted by the Analyzer) is {x 2, y x 4}
```

An important point to be noticed is that the TTCN standard does not allow the usage of an ASN.1 NamedValue within arithmetic expressions: (TTCN standard, 10.3.2.2).

Data Object Reference

A consequence of the above described restriction is that the TTCN suite does not permit the ComponentPosition mechanism for a DataObjectReference.

Another restriction for DataObjectReference is that the TTCN suite only permits Record referencing via ComponentIdentifiers. The mixed use of ComponentIdentifier, PDU_Identifier, and StructIdentifier for Record Reference is very vaguely described in the Standard. Almost any interpretation of the usage leads to possible ambiguities in the language.

The corresponding grammar for RecordRef (production 311) will in the TTCN suite be:

```
RecordRef ::= Dot ComponentIdentifier
```

The “path” leading down to the data object must be fully specified, i.e. all the component identifiers from the top to the bottom must be present.

Macro Value

The wildcard values “?”, “*”, or “-” may not in the TTCN suite be used in a Constraint where the identifier for the ASP Parameter, PDU Field, CM Element, or Structured Type Element is replaced by the macro symbol.

The Standard does not explicitly permit this nor does it allow this. The interpretation of using the wildcards for macros would however be very ambiguous. Rather than imposing one of the possible interpretations the TTCN suite will not allow the wildcards as macro values at all.

If wildcards are to be used as values for the elements that are inserted via macro expansion they must be written in the elements of a Structured Type Constraint that is referenced as macro value.

The TTCN-MP Syntax Productions in BNF

TTCN Specification

1. TTCN_Specification ::= TTCN_Module | Package | Suite

TTCN Module

2. TTCN_Module ::= **$TTCN_Module** TTCN_ModuleId TTCN_ModuleOverviewPart [TTCN_ModuleImportPart] [DeclarationsPart] [ConstraintsPart] [DynamicPart] **$End_TTCN_Module**
3. TTCN_ModuleId ::= **$TTCN_ModuleId** TTCN_ModuleIdentifier
4. TTCN_ModuleIdentifier ::= Identifier

The Module Overview

5. TTCN_ModuleOverviewPart ::= **$TTCN_ModuleOverviewPart** TTCN_ModuleExports [TTCN_ModuleStructure] [TestCaseIndex] [TestStepIndex] [DefaultIndex] **$End_TTCN_ModuleOverviewPart**

Module Exports

6. TTCN_ModuleExports ::= **$Begin_TTCN_ModuleExports** TTCN_ModuleId [TTCN_ModuleRef] [TTCN_ModuleObjective] [StandardsRef] [PICSref] [PIXITref] [TestMethods] [Comment] ExportedObjects [Comment] **$End_ModuleExports**
7. TTCN_ModuleRef ::= **$TTCN_ModuleRef** BoundedFreeText
8. TTCN_ModuleObjective ::= **$TTCN_ModuleObjective** BoundedFreeText
9. ExportedObjects ::= **$ExportedObjects** {ExportedObject} **$End_ExportedObjects**
10. ExportedObject ::= **$ExportedObject** ObjectId ObjectType [SourceInfo] [Comment] **$End_ExportedObject**
11. ObjectId ::= **$ObjectId** ObjectIdentifier
12. ObjectIdentifier ::= Identifier | ObjectTypeReference
13. ObjectTypeReference ::= Identifier '[' Identifier ']'
14. ObjectType ::= **$ObjectType** ObjectPredefinedType

15. ObjectPredefinedType ::= **SimpleType_Object** | **StructType_Object** | **ASN1_Type_Object** | **TS_Op_Object** | **TS_Proc_Object** | **TS_Par_Object** | **SelectExpr_Object** | **TS_Const_Object** | **TS_Var_Object** | **TC_Var_Object** | **PCO_Type_Object** | **PCO_Object** | **CP_Object** | **Timer_Object** | **TComp_Object** | **TCompConfig_Object** | **TTCN_ASP_Type_Object** | **ASN1_ASP_Type_Object** | **TTCN_PDU_Type_Object** | **ASN1_PDU_Type_Object** | **TTCN_CM_Type_Object** | **ASN1_CM_Type_Object** | **EncodingRule_Object** | **EncodingVariation_Object** | **InvalidFieldEncoding_Object** | **Alias_Object** | **StructTypeConstraint_Object** | **ASN1TypeConstraint_Object** | **TTCN_ASP_Constraint_Object** | **ASN1_ASP_Constraint_Object** | **TTCN_PDU_Constraint_Object** | **ASN1_PDU_Constraint_Object** | **TTCN_CM_Constraint_Object** | **ASN1_CM_Constraint_Object** | **TestCase_Object** | **TestStep_Object** | **Default_Object** | **NamedNumber_Object** | **Enumeration_Object**
16. SourceInfo ::= **$SourceInfo** (SourceIdentifier | ObjectDirective)
17. SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier | PackageIdentifier
18. ObjectDirective ::= '-' | OMIT | EXTERNAL

TTCN Module Structure

19. TTCN_ModuleStructure ::= **$Begin_TTCN_ModuleStructure** Structure&Objectives [Comment] **$End_TTCN_ModuleStructure**

TTCN Module Import Part

20. TTCN_ModuleImportPart ::= **$TTCN_ModuleImportPart** [ExternalObjects] [ImportDeclarations] **$End_TTCN_ModuleImportPart**

External Objects

21. ExternalObjects ::= **$Begin_ExternalObjects** {ExternalObject}+ [Comment] **$End_ExternalObjects**
22. ExternalObject ::= **$ExternalObject** ExternalObjectId ObjectType [Comment] **$End_ExternalObject**
23. ExternalObjectId ::= **$ExternalObjectId** ExternalObjectIdentifier
24. ExternalObjectIdentifier ::= ObjectIdentifier | TS_OpId&ParList | ConsId&ParList | TestStepId&ParList

Import Declarations

25. ImportDeclarations ::= **$ImportDeclarations** {Imports}+ **$End_ImportDeclarations**
26. Imports ::= **$Begin_Imports** SourceId [SourceRef] [StandardsRef] [Comment] ImportedObjects [Comment] **$End_Imports**
27. SourceId ::= **$SourceId** SourceIdentifier
28. SourceRef ::= **$SourceRef** BoundedFreeText

29. ImportedObjects ::= **$ImportedObjects** {ImportedObject}+ **$End_ImportedObjects**
30. ImportedObject ::= **$ImportedObject** ObjectId ObjectType [SourceInfo] [Comment] **$End_ImportedObject**

Source Package

31. Package ::= **$Package** PackageId PackageExports PackageImports **$End_Package**
32. PackageId ::= **$PackageId** PackageIdentifier
33. PackageIdentifier ::= Identifier

Package exports

34. PackageExports ::= **$Begin_PackageExports** PackageId [Comment] PackageExportedObjects [Comment] **$End_PackageExports**
35. PackageExportedObjects ::= **$PackageExportedObjects** {PackageExportedObject}+ **$End_PackageExportedObjects**
36. PackageExportedObject ::= **$PackageExportedObject** ObjectId ObjectType [SourceInfo] [Comment] **$End_PackageExportedObject**

Package imports and renames

37. PackageImports ::= **$PackageImports** {PackageImport} **$End_PackageImports**

Package Import

38. PackageImport ::= **$Begin_PackageImport** SourceId [Comment] PackageImportedObjects [Comment] **$End_PackageImport**
39. PackageImportedObjects ::= **$PackageImportedObjects** {PackageImportedObject}+ **$End_PackageImportedObjects**
40. PackageImportObject ::= **$PackageImportObject** ObjectId ObjectType [NewObjectDef] [NewObjectId] [Comment] **$End_PackageImportObject**
41. NewObjectDef ::= **$NewObjectDef** (NewObjectIdentifier | ObjectDirective)
42. NewObjectId ::= **$NewObjectId** NewObjectIdentifier
43. NewObjectIdentifier ::= ObjectIdentifier

Test suite

44. Suite ::= **$Suite** SuiteId SuiteOverviewPart [ImportPart] DeclarationsPart ConstraintsPart DynamicPart **$End_Suite**
45. SuiteId ::= **$SuiteId** SuiteIdentifier
46. SuiteIdentifier ::= Identifier

The Test Suite Overview

47. SuiteOverviewPart ::= **$SuiteOverviewPart** [TestSuiteIndex] SuiteStructure [TestCaseIndex] [TestStepIndex] [DefaultIndex] **$End_SuiteOverviewPart**

Test Suite Index

48. TestSuiteIndex ::= **$Begin_TestSuiteIndex** {ObjectInfo} [Comment] **$End_TestSuiteIndex**

The Imported Object Info

49. ObjectInfo ::= **$ObjectInfo** ObjectId ObjectType SourceId OrigObjectId [PageNum] [Comment] **$End_ObjectInfo**
50. PageNum ::= **$PageNum** PageNumber
51. PageNumber ::= Number
52. OrigObjectId ::= **$OrigObjectId** ObjectIdentifier

Test Suite Structure

53. SuiteStructure ::= **$Begin_SuiteStructure** SuiteId StandardsRef PICSref PIXITref TestMethods [Comment] Structure&Objectives [Comment] **$End_SuiteStructure**
54. StandardsRef ::= **$StandardsRef** BoundedFreeText
55. PICSref ::= **$PICSref** BoundedFreeText
56. PIXITref ::= **$PIXITref** BoundedFreeText
57. TestMethods ::= **$TestMethods** BoundedFreeText
58. Comment ::= **$Comment** [BoundedFreeText]
59. Structure&Objectives ::= **$Structure&Objectives** {Structure&Objective} **$End_Structure&Objectives**
60. Structure&Objective ::= **$Structure&Objective** TestGroupRef SelExprId Objective **$End_Structure&Objective**
61. SelExprId ::= **$SelectExprId** [SelectExprIdentifier]

Test Case Index

62. TestCaseIndex ::= **$Begin_TestCaseIndex** {CaseIndex} [Comment] **$End_TestCaseIndex**
63. CaseIndex ::= **$CaseIndex** TestGroupRef TestCaseId SelExprId Description **$End_CaseIndex**
64. Description ::= **$Description** BoundedFreeText

Test Step Index

65. TestStepIndex ::= **$Begin_TestStepIndex** {StepIndex} [Comment]
    $End_TestStepIndex
66. StepIndex ::= **$StepIndex** TestStepRef TestStepId Description **$End_StepIndex**

Default Index

67. DefaultIndex ::= **$Begin_DefaultIndex** {DefIndex} [Comment]
    $End_DefaultIndex
68. DefIndex ::= **$DefIndex** DefaultRef DefaultId Description **$End_DefIndex**

The Import Part

69. ImportPart ::= **$ImportPart** ImportDeclarations **$End_ImportPart**

The Declarations Part

70. DeclarationsPart ::= **$DeclarationsPart** Definitions Parameterization&Selection
    Declarations ComplexDefinitions **$End_DeclarationsPart**

Definitions

71. Definitions ::= [TS_TypeDefs] [EncodingDefs] [TS_OpDefs] [TS_ProcDefs]

Test Suite Type Definitions

72. TS_TypeDefs ::= **$TS_TypeDefs** [SimpleTypeDefs] [StructTypeDefs]
    [ASN1_TypeDefs] [ASN1_TypeRefs] **$End_TS_TypeDefs**

Simple Type Definitions

73. SimpleTypeDefs ::= **$Begin_SimpleTypeDefs** {SimpleTypeDef}+ [Comment] **$End_SimpleTypeDefs**
74. SimpleTypeDef ::= **$SimpleTypeDef** SimpleTypeId SimpleTypeDefinition [PDU_FieldEncoding] [Comment] **$End_SimpleTypeDef**
75. SimpleTypeId ::= **$SimpleTypeId** SimpleTypeIdentifier
76. SimpleTypeIdentifier ::= Identifier
77. SimpleTypeDefinition ::= **$SimpleTypeDefinition** Type&Restriction
78. Type&Restriction ::= Type [Restriction]
79. Restriction ::= LengthRestriction | IntegerRange | SimpleValueList
80. LengthRestriction ::= SingleTypeLength | RangeTypeLength
81. SingleTypeLength ::= "["Number "]"
82. RangeTypeLength ::= "[" LowerTypeBound To UpperTypeBound "]"
83. IntegerRange ::= "(" LowerTypeBound To UpperTypeBound ")"
84. LowerTypeBound ::= [Minus] Number | Minus **INFINITY**
85. UpperTypeBound ::= [Minus] Number | **INFINITY**
86. To ::= **TO** | ".."
87. SimpleValueList ::= "(" [Minus] LiteralValue {Comma [Minus] LiteralValue} ")"

Structured Type Definitions

88. StructTypeDefs ::= **$StructTypeDefs** {StructTypeDef}+ **$End_StructTypeDefs**
89. StructTypeDef ::= **$Begin_StructTypeDef** StructId [EncVariationId] [Comment] ElemDcls [Comment] **$End_StructTypeDef**
90. StructId ::= **$StructId** StructId&FullId
91. StructId&FullId ::= StructIdentifier [FullIdentifier]
92. FullIdentifier ::= "(" BoundedFreeText ")"
93. StructIdentifier ::= Identifier
94. ElemDcls ::= **$ElemDcls** {ElemDcl}+ **$End_ElemDcls**
95. ElemDcl ::= **$ElemDcl** ElemId ElemType [PDU_FieldEncoding] [Comment] **$End_ElemDcl**
96. ElemId ::= **$ElemId** ElemId&FullId
97. ElemId&FullId ::= ElemIdentifier [FullIdentifier]
98. ElemIdentifier ::= Identifier
99. ElemType ::= **$ElemType** Type&Attributes

ASN.1 Type Definitions

100.ASN1_TypeDefs ::= **$ASN1_TypeDefs** {ASN1_TypeDef}+
$End_ASN1_TypeDefs
101.ASN1_TypeDef ::= **$Begin_ASN1_TypeDef** ASN1_TypeId [EncVariationId] [Comment] ASN1_TypeDefinition [Comment] **$End_ASN1_TypeDef**
102.ASN1_TypeId ::= **$ASN1_TypeId** ASN1_TypeId&FullId
103.ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
104.ASN1_TypeIdentifier ::= Identifier
105.ASN1_TypeDefinition ::= **$ASN1_TypeDefinition** ASN1_Type&LocalTypes
$End_ASN1_TypeDefinition
106.ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}
107.ASN1_Type ::= ASN1_main_Type [ASN1_Encoding]
108.ASN1_LocalType ::= TypeAssignment

ASN.1 Type Definitions by Reference

109.ASN1_TypeRefs ::= **$Begin_ASN1_TypeRefs** {ASN1_TypeRef}+ [Comment]
$End_ASN1_TypeRefs
110.ASN1_TypeRef ::= **$ASN1_TypeRef** ASN1_TypeId ASN1_TypeReference ASN1_ModuleId [EncVariationId] [Comment] [*ASN1_TypeDefinition*]
$End_ASN1_TypeRef
111.ASN1_TypeReference ::= **$ASN1_TypeReference** TypeReference
112.TypeReference ::= typereference
113.ASN1_ModuleId ::= **$ASN1_ModuleId** ModuleIdentifier
114.ModuleIdentifier ::= moduleidentifier

Test Suite Operation Definitions

115.TS_OpDefs ::= **$TS_OpDefs** {TS_OpDef}+ **$End_TS_OpDefs**
116.TS_OpDef ::= **$Begin_TS_OpDef** TS_OpId TS_OpResult [Comment] TS_OpDescription [Comment] **$End_TS_OpDef**
117.TS_OpId ::= **$TS_OpId** TS_OpId&ParList
118.TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]
119.TS_OpIdentifier ::= Identifier
120.TS_OpResult ::= **$TS_OpResult** TypeOrPDU
121.TS_OpDescription ::= **$TS_OpDescription** BoundedFreeText

Test Suite Operation Procedural Definitions

122.TS_ProcDefs ::= **$TS_ProcDefs** {TS_ProcDef}+ **$End_TS_ProcDefs**
123.TS_ProcDef ::= **$Begin_TS_ProcDef** TS_ProcId TS_ProcResult [Comment] TS_ProcDescription [Comment] **$End_TS_ProcDef**
124.TS_ProcId ::= **$TS_ProcId** TS_ProcId&ParList
125.TS_ProcId&ParList ::= TS_ProcIdentifier [FormalParList]
126.TS_ProcIdentifier ::= Identifier
127.TS_ProcResult ::= **$TS_ProcResult** TypeOrPDU
128.TS_ProcDescription ::= **$TS_ProcDescription** TS_OpProcDef **$End_TS_ProcDescription**
129.TS_OpProcDef ::= [VarBlock] ProcStatement
130.VarBlock ::= **VAR** VarDcls **ENDVAR**
131.VarDcls ::= {VarDcl SemiColon}
132.VarDcl ::= [**STATIC**] VarIdentifiers Colon TypeOrPDU [Colon Value]
133.VarIdentifiers ::= VarIdentifier {Comma VarIdentifier}
134.VarIdentifier ::= Identifier
135.ProcStatement ::= ReturnValueStatement | Assignment | IfStatement | WhileLoop | CaseStatement | ProcBlock
136.ReturnValueStatement ::= **RETURNVALUE** Expression
137.IfStatement ::= **IF** Expression **THEN** {ProcStatement SemiColon}+ [**ELSE** {ProcStatement SemiColon}+] **ENDIF**
138.WhileLoop ::= **WHILE** Expression **DO** {ProcStatement SemiColon}+ **END-WHILE**
139.CaseStatement ::= **CASE** Expression **OF** {CaseClause SemiColon}+ [**ELSE** {ProcStatement SemiColon}+] **ENDCASE**
140.CaseClause ::= IntegerLabel Colon ProcStatement
141.IntegerLabel ::= Number | TS_ParIdentifier | TS_ConstIdentifier
142.ProcBlock ::= **BEGIN** {ProcStatement SemiColon}+ **END**

Parameterization and Selection

143.Parameterization&Selection ::= [TS_ParDcls] [SelectExprDefs]

Test Suite Parameter Declarations

144.TS_ParDcls ::= **$Begin_TS_ParDcls** {TS_ParDcl}+ [Comment]
$End_TS_ParDcls
145.TS_ParDcl ::= **$TS_ParDcl** TS_ParId TS_ParType PICS_PIXITref [Comment]
$End_TS_ParDcl
146.TS_ParId ::= **$TS_ParId** TS_ParIdentifier
147.TS_ParIdentifier ::= Identifier
148.TS_ParType ::= **$TS_ParType** TypeOrPDU
149.PICS_PIXITref ::= **$PICS_PIXITref** BoundedFreeText

Test Case Selection Expression Definitions

150.SelectExprDefs ::= **$Begin_SelectExprDefs** {SelectExprDef}+ [Comment]
$End_SelectExprDefs
151.SelectExprDef ::= **$SelectExprDef** SelectExprId SelectExpr [Comment]
$End_SelectExprDef
152.SelectExprId ::= **$SelectExprId** SelectExprIdentifier
153.SelectExprIdentifier ::= Identifier
154.SelectExpr ::= **$SelectExpr** SelectionExpression
155.SelectionExpression ::= Expression

Declarations

156.Declarations ::= [TS_ConstDcls] [TS_ConstRefs] [TS_VarDcls] [TC_VarDcls] [PCO_TypeDcls] [PCO_Dcls] [CP_Dcls] [TimerDcls] [TCompDcls TCompConfigDcls]

Test Suite Constant Declarations

157.TS_ConstDcls ::= **$Begin_TS_ConstDcls** {TS_ConstDcl}+ [Comment]
$End_TS_ConstDcls
158.TS_ConstDcl ::= **$TS_ConstDcl** TS_ConstId TS_ConstType TS_ConstValue [Comment] **$End_TS_ConstDcl**
159.TS_ConstId ::= **$TS_ConstId** TS_ConstIdentifier
160.TS_ConstIdentifier ::= Identifier
161.TS_ConstType ::= **$TS_ConstType** Type
162.TS_ConstValue ::= **$TS_ConstValue** DeclarationValue
163.DeclarationValue ::= Expression

Test Suite Constant Declarations by Reference

164.TS_ConstRefs ::= **$Begin_TS_ConstRefs** {TS_ConstRef}+ [Comment]
$End_TS_ConstRefs
165.TS_ConstRef ::= **$TS_ConstRef** TS_ConstId TS_ConstType
ASN1_ValueReference ASN1_ModuleId [Comment] *[ASN1_Value]*
$End_TS_ConstRef
166.ASN1_ValueReference ::= **$ASN1_ValueReference** ValueReference
167.ValueReference ::= valuereference

Test Suite Variable Declarations

168.TS_VarDcls ::= **$Begin_TS_VarDcls** {TS_VarDcl}+ [Comment]
$End_TS_VarDcls
169.TS_VarDcl ::= **$TS_VarDcl** TS_VarId TS_VarType TS_VarValue [Comment]
$End_TS_VarDcl
170.TS_VarId ::= **$TS_VarId** TS_VarIdentifier
171.TS_VarIdentifier ::= Identifier
172.TS_VarType ::= **$TS_VarType** TypeOrPDU
173.TS_VarValue ::= **$TS_VarValue** [DeclarationValue]

Test Case Variable Declarations

174.TC_VarDcls ::= **$Begin_TC_VarDcls** {TC_VarDcl}+ [Comment]
$End_TC_VarDcls
175.TC_VarDcl ::= **$TC_VarDcl** TC_VarId TC_VarType TC_VarValue [Comment]
$End_TC_VarDcl
176.TC_VarId ::= **$TC_VarId** TC_VarIdentifier
177.TC_VarIdentifier ::= Identifier
178.TC_VarType ::= **$TC_VarType** TypeOrPDU
179.TC_VarValue ::= **$TC_VarValue** [DeclarationValue]

PCO Type Declarations

180.PCO_TypeDcls ::= **$Begin_PCO_TypeDcls** {PCO_TypeDcl}+ [Comment] **$End_PCO_TypeDcls**
181.PCO_TypeDcl ::= **$PCO_TypeDcl** PCO_TypeId P_Role [Comment] **$End_PCO_TypeDcl**
182.PCO_TypeId ::= **$PCO_TypeId** PCO_TypeIdentifier
183.PCO_TypeIdentifier ::= Identifier

PCO Declarations

184.PCO_Dcls ::= **$Begin_PCO_Dcls** {PCO_Dcl}+ [Comment] **$End_PCO_Dcls**
185.PCO_Dcl ::= **$PCO_Dcl** PCO_Id PCO_TypeId&MuxValue P_Role [Comment] **$End_PCO_Dcl**
186.PCO_Id ::= **$PCO_Id** PCO_Identifier
187.PCO_Identifier ::= Identifier
188.PCO_TypeId&MuxValue ::= **$PCO_TypeId** PCO_TypeIdentifier [“(“ MuxValue “)”]
189.MuxValue ::= TS_ParIdentifier
190.P_Role ::= **$PCO_Role** PCO_Role
191.PCO_Role ::= **UT** | **LT**

Coordination Points Declaration

192.CP_Dcls ::= **$Begin_CP_Dcls** {CP_Dcl}+ [Comment] **$End_CP_Dcls**
193.CP_Dcl ::= **$CP_Dcl** CP_Id [Comment] **$End_CP_Dcl**
194.CP_Id ::= **$CP_Id** CP_Identifier
195.CP_Identifier ::= Identifier

Timer Declarations

196.TimerDcls ::= **$Begin_TimerDcls** {TimerDcl}+ [Comment] **$End_TimerDcls**
197.TimerDcl ::= **$TimerDcl** TimerId Duration Unit [Comment] **$End_TimerDcl**
198.TimerId ::= **$TimerId** TimerIdentifier
199.TimerIdentifier ::= Identifier
200.Duration ::= **$Duration** [DeclarationValue]
201.Unit ::= **$Unit** TimeUnit
202.TimeUnit ::= **ps** | **ns** | **us** | **ms** | **s** | **min**

Test Component Declarations

203.TCompDcls ::= **$Begin_TCompDcls** {TCompDcl}+ [Comment] **$End_TCompDcls**
204.TCompDcl ::= **$TCompDcl** TCompId TCompRole NumOf PCOs NumOf CPs [Comment] **$EndTCompDcl**
205.TCompId ::= **$TCompId** TCompIdentifier
206.TCompIdentifier ::= Identifier
207.TCompRole ::= **$TCompRole** TC_Role
208.TC_Role ::= **MTC** | **PTC**
209.NumOf_PCOs ::= **$NumOf_PCOs** Num_PCOs
210.Num_PCOs ::= Number
211.NumOf_Cps ::= **$NumOf_CPs** Num_CPs
212.Num_CPs ::= Number

Test Component Configuration Declarations

213.TCompConfigDcls ::= **$TCompConfigDcls** {TCompConfigDcl}+ **$End_TCompConfigDcls**
214.TCompConfigDcl ::= **$Begin_TCompConfigDcl** TCompConfigId [Comment] TCompConfigInfos [Comment] **$EndTCompConfigDcl**
215.TCompConfigId ::= **$TCompConfigId** TCompConfigIdentifier
216.TCompConfigIdentifier ::= Identifier
217.TCompConfigInfos ::= **$TCompConfigInfos** {TCompConfigInfo}+ **$End_TCompConfigInfos**
218.TCompConfigInfo ::= **$TCompConfigInfo** TCompUsed PCOs_Used CPs_Used [Comment] **$End_TCompConfigInfo**
219.TCompUsed ::= **$TCompUsed** TCompIdentifier
220.PCOs_Used ::= **$PCOs_Used** [PCO_List]
221.PCO_List ::= PCO_Identifier {Comma PCO_Identifier}
222.CPs_Used ::= **$CPs_Used** [CP_List]
223.CP_List ::= CP_Identifier {Comma CP_Identifier}

ASP, PDU and CM Type Definitions

224.ComplexDefinitions ::= [ASP_TypeDefs] [PDU_TypeDefs] [CM_TypeDefs] [AliasDefs]

ASP Type Definitions

225.ASP_TypeDefs ::= **$ASP_TypeDefs** [TTCN_ASP_TypeDefs] [ASN1_ASP_TypeDefs] [ASN1_ASP_TypeDefsByRef] **$End_ASP_TypeDefs**

Tabular ASP Type Definitions

226.TTCN_ASP_TypeDefs ::= **$TTCN_ASP_TypeDefs** {TTCN_ASP_TypeDef}+ **$End_TTCN_ASP_TypeDefs**
227.TTCN_ASP_TypeDef ::= **$Begin_TTCN_ASP_TypeDef** ASP_Id PCO_Type [Comment] [ASP_ParDcls] [Comment] **$End_TTCN_ASP_TypeDef**
228.PCO_Type ::= **$PCO_Type** [PCO_TypeIdentifier]
229.ASP_Id ::= **$ASP_Id** ASP_Id&FullId
230.ASP_Id&FullId ::= ASP_Identifier [FullIdentifier]
231.ASP_Identifier ::= Identifier
232.ASP_ParDcls ::= **$ASP_ParDcls** {ASP_ParDcl} **$End_ASP_ParDcls**
233.ASP_ParDcl ::= **$ASP_ParDcl** ASP_ParId ASP_ParType [Comment] **$End_ASP_ParDcl**
234.ASP_ParId ::= **$ASP_ParId** ASP_ParIdOrMacro
235.ASP_ParIdOrMacro ::= ASP_ParId&FullId | MacroSymbol
236.ASP_ParId&FullId ::= ASP_ParIdentifier [FullIdentifier]
237.ASP_ParIdentifier ::= Identifier
238.ASP_ParType ::= **$ASP_ParType** Type&Attributes

ASN.1 ASP Type Definitions

239.ASN1_ASP_TypeDefs ::= **$ASN1_ASP_TypeDefs** {ASN1_ASP_TypeDef} **$End_ASN1_ASP_TypeDefs**
240.ASN1_ASP_TypeDef ::= **$Begin_ASN1_ASP_TypeDef** ASP_Id PCO_Type [Comment] [ASN1_TypeDefinition] [Comment] **$End_ASN1_ASP_TypeDef**

ASN.1 ASP Type Definitions by Reference

241.ASN1_ASP_TypeDefsByRef ::= **$Begin_ASN1_ASP_TypeDefsByRef** {ASN1_ASP_TypeDefByRef}+ [Comment] **$End_ASN1_ASP_TypeDefsByRef**
242.ASN1_ASP_TypeDefByRef ::= **$ASN1_ASP_TypeDefByRef** ASP_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [Comment] *[ASN1_TypeDefinition]* **$End_ASN1_ASP_TypeDefByRef**

PDU Type Definitions

243.PDU_TypeDefs ::= **$PDU_TypeDefs** [TTCN_PDU_TypeDefs] [ASN1_PDU_TypeDefs] [ASN1_PDU_TypeDefsByRef] **$End_PDU_TypeDefs**

Tabular PDU Type Definitions

244.TTCN_PDU_TypeDefs ::= **$TTCN_PDU_TypeDefs** {TTCN_PDU_TypeDef}+ **$End_TTCN_PDU_TypeDefs**
245.TTCN_PDU_TypeDef ::= **$Begin_TTCN_PDU_TypeDef** PDU_Id PCO_Type [PDU_EncodingId] [EncVariationId] [Comment] [PDU_FieldDcls] [Comment] **$End_TTCN_PDU_TypeDef**
246.PDU_Id ::= **$PDU_Id** PDU_Id&FullId
247.PDU_Id&FullId ::= PDU_Identifier [FullIdentifier]
248.PDU_Identifier ::= Identifier
249.PDU_EncodingId ::= **$PDU_EncodingId** [EncodingRuleIdentifier]
250.PDU_FieldDcls ::= **$PDU_FieldDcls** {PDU_FieldDcl} **$End_PDU_FieldDcls**
251.PDU_FieldDcl ::= **$PDU_FieldDcl** PDU_FieldId PDU_FieldType [PDU_FieldEncoding] [Comment] **$End_PDU_FieldDcl**
252.PDU_FieldId ::= **$PDU_FieldId** PDU_FieldIdOrMacro
253.PDU_FieldIdOrMacro ::= PDU_FieldId&FullId | MacroSymbol
254.MacroSymbol ::= "<-"
255.PDU_FieldId&FullId ::= PDU_FieldIdentifier [FullIdentifier]
256.PDU_FieldIdentifier ::= Identifier
257.PDU_FieldType ::= **$PDU_FieldType** Type&Attributes
258.Type&Attributes ::= (Type [LengthAttribute]) | **PDU**
259.LengthAttribute ::= SingleLength | RangeLength
260.SingleLength ::= "[" Bound "]"
261.Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
262.RangeLength ::= "[" LowerBound To UpperBound "]"
263.LowerBound ::= Bound
264.UpperBound ::= Bound | **INFINITY**

ASN.1 PDU Type Definitions

265.ASN1_PDU_TypeDefs ::= **$ASN1_PDU_TypeDefs** {ASN1_PDU_TypeDef} **$End_ASN1_PDU_TypeDefs**
266.ASN1_PDU_TypeDef ::= **$Begin_ASN1_PDU_TypeDef** PDU_Id PCO_Type [PDU_EncodingId] [EncVariationId] [Comment] [ASN1_TypeDefinition] [Comment] **$End_ASN1_PDU_TypeDef**

ASN.1 PDU Type Definitions by Reference

267.ASN1_PDU_TypeDefsByRef ::= **$Begin_ASN1_PDU_TypeDefsByRef** {ASN1_PDU_TypeDefByRef}+ [Comment] **$End_ASN1_PDU_TypeDefsByRef**

268.ASN1_PDU_TypeDefByRef ::= **$ASN1_PDU_TypeDefByRef** PDU_Id PCO_Type ASN1_TypeReference ASN1_ModuleId [PDU_EncodingId] [Enc-VariationId] [Comment] *[ASN1_TypeDefinition]* **$End_ASN1_PDU_TypeDefByRef**

CM Type Definitions

269.CM_TypeDefs ::= **$CM_TypeDefs** [TTCN_CM_TypeDefs] [ASN1_CM_TypeDefs] **$End_CM_TypeDefs**

Tabular CM Type Definitions

270.TTCN_CM_TypeDefs ::= **$TTCN_CM_TypeDefs** {TTCN_CM_TypeDefs}+ **$End_TTCN_CM_TypeDefs**

271.TTCN_CM_TypeDef ::= **$Begin_TTCN_CM_TypeDef** CM_Id [Comment] [CM_ParDcls] [Comment] **$End_TTCN_CM_TypeDef**

272.CM_Id ::= **$CM_Id** CM_Identifier

273.CM_Identifier ::= Identifier

274.CM_ParDcls ::= **$CM_ParDcls** {CM_ParDcl} **$End_CM_ParDcls**

275.CM_ParDcl ::= **$CM_ParDcl** CM_ParId CM_ParType [Comment] **$End_CM_ParDcl**

276.CM_ParId ::= **$CM_ParId** CM_ParIdOrMacro

277.CM_ParIdOrMacro ::= CM_ParIdentifier | MacroSymbol

278.CM_ParIdentifier ::= Identifier

279.CM_ParType ::= **$CM_ParType** Type&Attributes

ASN1 CM Type Definitions

280.ASN1_CM_TypeDefs ::= **$ASN1_CM_TypeDefs** {ASN1_CM_TypeDefs}+ **$End_ASN1_CM_TypeDefs**

281.ASN1_CM_TypeDef ::= **$Begin_ASN1_CM_TypeDef** CM_Id [Comment] [ASN1_TypeDefinition] [Comment] **$End_ASN1_CM_TypeDef**

Varieties of Encoding Definition

282.EncodingDefs ::= **$EncodingDefs** [EncodingDefinitions] [EncodingVariations] [InvalidFieldEncodingDefs] **$End_EncodingDefs**

Encoding Definitions

283.EncodingDefinitions ::= **$Begin_EncodingDefinitions** {EncodingDefinition}+ [Comment] **$End_EncodingDefinitions**
284.EncodingDefinition ::= **$EncodingDefinition** EncodingRuleId EncodingRef EncodingDefault [Comment] **$End_EncodingDefinition**
285.EncodingRuleId ::= **$EncodingRuleId** EncodingRuleIdentifier
286.EncodingRuleIdentifier ::= Identifier
287.EncodingRef ::= **$EncodingRef** EncodingReference
288.EncodingReference ::= BoundedFreeText
289.EncodingDefault ::= **$EncodingDefault** [DefaultExpression]
290.DefaultExpression ::= Expression

Encoding Variations

291.EncodingVariations ::= **$EncodingVariations** {EncodingVariationSet}+ **$End_EncodingVariations**
292.EncodingVariationSet ::= **$Begin_EncodingVariationSet** EncodingRuleId Encoding_TypeList [Comment] {EncodingVariation}+ [Comment] **$End_EncodingVariationSet**
293.Encoding_TypeList ::= **$Encoding_TypeList** [TypeList]
294.TypeList ::= Type {Comma Type}
295.EncodingVariation ::= **$EncodingVariation** EncodingVariationId VariationRef VariationDefault [Comment] **$End_EncodingVariation**
296.EncodingVariationId ::= **$EncodingVariationId** EncVariationId&ParList
297.EncVariationId&ParList ::= EncVariationIdentifier [FormalParList]
298.EncVariationIdentifier ::= Identifier
299.VariationRef ::= **$VariationRef** VariationReference
300.VariationReference ::= BoundedFreeText
301.VariationDefault ::= **$VariationDefault** [DefaultExpression]

Invalid Encoding Definitions

302.InvalidFieldEncodingDefs ::= **$InvalidFieldEncodingDefs** {InvalidFieldEncodingDef}+ **$End_InvalidFieldEncodingDefs**
303.InvalidFieldEncodingDef ::= **$Begin_InvalidFieldEncodingDef** InvalidFieldEncodingId Encoding_TypeList [Comment] InvalidFieldEncodingDefinition [Comment] **$End_InvalidFieldEncodingDef**
304.InvalidFieldEncodingId ::= **$InvalidFieldEncodingId** InvalidFieldEncodingId&ParList

305.InvalidFieldEncodingId&ParList ::= InvalidFieldEncodingIdentifier [FormalParList]
306.InvalidFieldEncodingIdentifier ::= Identifier
307.InvalidFieldEncodingDefinition ::= **$InvalidFieldEncodingDefinition** TS_OpProcDef **$End_InvalidFieldEncodingDefinition**

Alias Definitions

308.AliasDefs ::= **$Begin_AliasDefs** {AliasDef}+ [Comment] **$End_AliasDefs**
309.AliasDef ::= **$AliasDef** AliasId ExpandedId [Comment] **$End_AliasDef**
310.AliasId ::= **$AliasId** AliasIdentifier
311.AliasIdentifier ::= Identifier
312.ExpandedId ::= **$ExpandedId** Expansion
313.Expansion ::= ASP_Identifier | PDU_Identifier

The Constraints Part

314.ConstraintsPart ::= **$ConstraintsPart** [TS_TypeConstraints] [ASP_Constraints] [PDU_Constraints] [CM_Constraints] **$End_ConstraintsPart**

Test Suite Type Constraint Declarations

315.TS_TypeConstraints ::= **$TS_TypeConstraints** [StructTypeConstraints] [ASN1_TypeConstraints] **$End_TS_TypeConstraints**

Structured Type Constraint Declarations

316.StructTypeConstraints ::= **$StructTypeConstraints** {StructTypeConstraint}+ **$End_StructTypeConstraints**
317.StructTypeConstraint ::= **$Begin_StructTypeConstraint** ConsId StructId DerivPath [EncVariationId] [Comment] ElemValues [Comment] **$End_StructTypeConstraint**
318.EncVariationId ::= **$EncVariationId** [EncVariationCall]
319.EncVariationCall ::= EncVariationIdentifier [ActualParList]
320.ElemValues ::= **$ElemValues** {ElemValue}+ **$End_ElemValues**
321.ElemValue ::= **$ElemValue** ElemId ConsValue [PDU_FieldEncoding] [Comment] **$End_ElemValue**
322.PDU_FieldEncoding ::= **$PDU_FieldEncoding** [PDU_FieldEncodingCall]
323.PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall
324.InvalidFieldEncodingCall ::= InvalidFieldEncodingIdentifier (ActualParList | "(" ")")

ASN.1 Type Constraint Declarations

325.ASN1_TypeConstraints ::= **$ASN1_TypeConstraints** {ASN1_TypeConstraint}+ **$End_ASN1_TypeConstraints**

326.ASN1_TypeConstraint ::= **$Begin_ASN1_TypeConstraint** ConsId ASN1_TypeId DerivPath [EncVariationId] [Comment] ASN1_ConsValue [Comment] **$End_ASN1_TypeConstraint**

ASP Constraint Declarations

327.ASP_Constraints ::= **$ASP_Constraints** [TTCN_ASP_Constraints] [ASN1_ASP_Constraints] **$End_ASP_Constraints**

Tabular ASP Constraint Declarations

328.TTCN_ASP_Constraints ::= **$TTCN_ASP_Constraints** {TTCN_ASP_Constraint}+ **$End_TTCN_ASP_Constraints**

329.TTCN_ASP_Constraint ::= **$Begin_TTCN_ASP_Constraint** ConsId ASP_Id DerivPath [Comment] [ASP_ParValues] [Comment] **$End_TTCN_ASP_Constraint**

330.ASP_ParValues ::= **$ASP_ParValues** {ASP_ParValue}+ **$End_ASP_ParValues**

331.ASP_ParValue ::= **$ASP_ParValue** ASP_ParId ConsValue [Comment] **$End_ASP_ParValue**

ASN.1 ASP Constraint Declarations

332.ASN1_ASP_Constraints ::= **$ASN1_ASP_Constraints** {ASN1_ASP_Constraint}+ **$End_ASN1_ASP_Constraints**

333.ASN1_ASP_Constraint ::= **$Begin_ASN1_ASP_Constraint** ConsId ASP_Id DerivPath [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_ASP_Constraint**

PDU Constraint Declarations

334.PDU_Constraints ::= **$PDU_Constraints** [TTCN_PDU_Constraints] [ASN1_PDU_Constraints] **$End_PDU_Constraints**

Tabular PDU Constraint Declarations

335.TTCN_PDU_Constraints ::= **$TTCN_PDU_Constraints** {TTCN_PDU_Constraint}+ **$End_TTCN_PDU_Constraints**
336.TTCN_PDU_Constraint ::= **$Begin_TTCN_PDU_Constraint** ConsId PDU_Id DerivPath [EncRuleId] [EncVariationId] [Comment] [PDU_FieldValues] [Comment] **$End_TTCN_PDU_Constraint**
337.EncRuleId ::= **$EncRuleId** [EncodingRuleIdentifier]
338.ConsId ::= **$ConsId** ConsId&ParList
339.ConsId&ParList ::= ConstraintIdentifier [FormalParList]
340.ConstraintIdentifier ::= Identifier
341.DerivPath ::= **$DerivPath** [DerivationPath]
342.DerivationPath ::= {ConstraintIdentifier Dot}+
343.PDU_FieldValues ::= **$PDU_FieldValues** {PDU_FieldValue}+ **$End_PDU_FieldValues**
344.PDU_FieldValue ::= **$PDU_FieldValue** PDU_FieldId ConsValue [PDU_FieldEncoding] [Comment] **$End_PDU_FieldValue**
345.ConsValue ::= **$ConsValue** ConstraintValue&Attributes
346.ConstraintValue&Attributes ::= ConstraintValue ValueAttributes
347.ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef
348.ConstraintExpression ::= Expression
349.MatchingSymbol ::= Complement | Omit | AnyValue | AnyOrOmit | ValueList | ValueRange | SuperSet | SubSet | Permutation
350.Complement ::= **COMPLEMENT** ValueList
351.Omit ::= Dash | **OMIT**
352.AnyValue ::= "?"
353.AnyOrOmit ::= "*"
354.ValueList ::= "(" ConstraintValue&Attributes { Comma ConstraintValue&Attributes }")"
355.ValueRange ::= "(" ValRange ")"
356.ValRange ::= (LowerRangeBound To UpperRangeBound)
357.LowerRangeBound ::= ConstraintExpression | Minus **INFINITY**
358.UpperRangeBound ::= ConstraintExpression | **INFINITY**
359.SuperSet ::= **SUPERSET** "(" ConstraintValue&Attributes ")"
360.SubSet ::= **SUBSET** "(" ConstraintValue&Attributes ")"
361.Permutation ::= **PERMUTATION** ValueList
362.ValueAttributes ::= [ValueLength] [**IF_PRESENT**] [ASN1_Encoding]
363.ASN1_Encoding ::= **ENC** PDU_FieldEncodingCall
364.ValueLength ::= SingleValueLength | RangeValueLength
365.SingleValueLength ::= "[" ValueBound "]"
366.ValueBound ::= Number | TS_ParIdentifier | TS_ConstIdentifier | FormalParIdentifier

367.RangeValueLength ::= "[" LowerValueBound To UpperValueBound "]"
368.LowerValueBound ::= ValueBound
369.UpperValueBound ::= ValueBound | **INFINITY**

ASN.1 PDU Constraint Declarations

370.ASN1_PDU_Constraints ::= **$ASN1_PDU_Constraints** {ASN1_PDU_Constraint}+ **$End_ASN1_PDU_Constraints**
371.ASN1_PDU_Constraint ::= **$Begin_ASN1_PDU_Constraint** ConsId PDU_Id DerivPath [EncRuleId] [EncVariationId] [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_PDU_Constraint**
372.ASN1_ConsValue ::= **$ASN1_ConsValue** ConstraintValue&AttributesOrReplace **$End_ASN1_ConsValue**
373.ConstraintValue&AttributesOrReplace ::= ConstraintValue&Attributes | Replacement {Comma Replacement}
374.Replacement ::= (**REPLACE** ReferenceList **BY** ConstraintValue&Attributes) | (**OMIT** ReferenceList)
375.ReferenceList ::= (ArrayRef | ComponentIdentifier | ComponentPosition) {ComponentReference}

CM Constraint Declarations

376.CM_Constraints ::= **$CM_Constraints** [TTCN_CM_Constraints] [ASN1_CM_Constraints] **$End_CM_Constraints**

Tabular CM Constraint Declaration

377.TTCN_CM_Constraints ::= **$TTCN_CM_Constraints** {TTCN_CM_Constraint}+ **$End_TTCN_CM_Constraints**
378.TTCN_CM_Constraint ::= **$Begin_TTCN_CM_Constraint** ConsId CM_Id DerivPath [Comment] [CM_ParValues] [Comment] **$End_TTCN_CM_Constraint**
379.CM_ParValues ::= **$CM_ParValues** {CM_ParValue} **$End_CM_ParValues**
380.CM_ParValue ::= **$CM_ParValue** CM_ParId ConsValue [Comment] **$End_CM_ParValue**

ASN.1 CM Constraint Declaration

381.ASN1_CM_Constraints ::= **$ASN1_CM_Constraints** {ASN1_CM_Constraint}+ **$End_ASN1_CM_Constraints**
382.ASN1_CM_Constraint ::= **$Begin_ASN1_CM_Constraint** ConsId CM_Id DerivPath [Comment] [ASN1_ConsValue] [Comment] **$End_ASN1_CM_Constraint**

The Dynamic Part

383.DynamicPart ::= **$DynamicPart** [TestCases] [TestStepLibrary] [DefaultsLibrary] **$End_DynamicPart**

Test Cases

384.TestCases ::= **$TestCases** {TestGroup | TestCase | *CompactTestGroup*}+ **$End_TestCases**
385.TestGroup ::= **$TestGroup** TestGroupId *[SelExprId] [Objective]* {TestGroup | TestCase}+ **$End_TestGroup**
386.TestGroupId ::= **$TestGroupId** TestGroupIdentifier
387.TestGroupIdentifier ::= Identifier
388.TestCase ::= **$Begin_TestCase** TestCaseId TestGroupRef TestPurpose [Configuration] DefaultsRef [Comment] *[SelExprId] [Description]* BehaviourDescription [Comment] **$End_TestCase**
389.TestCaseId ::= **$TestCaseId** TestCaseIdentifier
390.TestCaseIdentifier ::= Identifier
391.TestGroupRef ::= **$TestGroupRef** TestGroupReference
392.TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
393.TestPurpose ::= **$TestPurpose** BoundedFreeText
394.Configuration ::= **$Configuration** TCompConfigIdentifier
395.DefaultsRef ::= **$DefaultsRef** [DefaultRefList]
396.DefaultRefList ::= DefaultReference {Comma DefaultReference}
397.DefaultReference ::= DefaultIdentifier [ActualParList]

397a.CompactTestGroup ::= **$Begin_CompactTestGroup** TestGroupId DefaultsRef *[SelExprId] [Objective]* {CompactTestCase} **$End_CompactTestGroup**

397b.CompactTestCase ::= **$CompactTestCase** TestCaseId TestPurpose TestStepAttachment [Comment] *[SelExprId] [Description]* **$End_CompactTestCase**

397c.TestStepAttachment ::= **$TestStepAttachment** Attach

Test Step Library

398.TestStepLibrary ::= **$TestStepLibrary** {TestStepGroup | TestStep}+ **$End_TestStepLibrary**
399.TestStepGroup ::= **$TestStepGroup** TestStepGroupId {TestStepGroup | TestStep}+ **$End_TestStepGroup**
400.TestStepGroupId ::= **$TestStepGroupId** TestStepGroupIdentifier
401.TestStepGroupIdentifier ::= Identifier
402.TestStep ::= **$Begin_TestStep** TestStepId TestStepRef Objective DefaultsRef [Comment] *[Description]* BehaviourDescription [Comment] **$End_TestStep**
403.TestStepId ::= **$TestStepId** TestStepId&ParList
404.TestStepId&ParList ::= TestStepIdentifier [FormalParList]
405.TestStepIdentifier ::= Identifier
406.TestStepRef ::= **$TestStepRef** TestStepGroupReference
407.TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}
408.Objective ::= **$Objective** BoundedFreeText

Default Library

409.DefaultsLibrary ::= **$DefaultsLibrary** {DefaultGroup | Default}+
$End_DefaultsLibrary
410.DefaultGroup ::= **$DefaultGroup** DefaultGroupId {DefaultGroup | Default}+
$End_DefaultGroup
411.DefaultGroupId ::= **$DefaultGroupId** DefaultGroupIdentifier
412.Default ::= **$Begin_Default** DefaultId DefaultRef Objective [Comment] *[Description]* BehaviourDescription [Comment] **$End_Default**
413.DefaultRef ::= **$DefaultRef** DefaultGroupReference
414.DefaultId ::= **$DefaultId** DefaultId&ParList
415.DefaultId&ParList ::= DefaultIdentifier [FormalParList]
416.DefaultIdentifier ::= Identifier
417.DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/"}
418.DefaultGroupIdentifier ::= Identifier

Behaviour descriptions

419.BehaviourDescription ::= **$BehaviourDescription** RootTree {LocalTree}
$End_BehaviourDescription
420.RootTree ::= {BehaviourLine}+
421.LocalTree ::= Header {BehaviourLine}+
422.Header ::= **$Header** TreeHeader
423.TreeHeader ::= TreeIdentifier [FormalParList]
424.TreeIdentifier ::= Identifier
425.FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"
426.FormalPar&Type ::= FormalParIdentifier {Comma FormalParIdentifier} Colon FormalParType
427.FormalParIdentifier ::= Identifier
428.FormalParType ::= Type | PCO_TypeIdentifier | **PDU** | **CP** | **TIMER**

Behaviour lines

429.BehaviourLine ::= **$BehaviourLine** LabelId Line Cref VerdictId [Comment]
$End_BehaviourLine
430.Line ::= **$Line** Indentation StatementLine
431.Indentation ::= "[" Number "]"
432.LabelId ::= **$LabelId** [Label]
433.Label ::= Identifier
434.Cref ::= **$Cref** [ConstraintReference]
435.ConstraintReference ::= ConsRef | FormalParIdentifier | AnyValue
436.ConsRef ::= ConstraintIdentifier [ActualCrefParList]
437.ActualCrefParList ::= "(" ActualCrefPar {Comma ActualCrefPar} ")"
438.ActualCrefPar ::= Value
439.VerdictId ::= **$VerdictId** [Verdict]
440.Verdict ::= Pass | Fail | Inconclusive | Result
441.Pass ::= **PASS** | **P** | "(" **PASS** ")" | "(" **P** ")"
442.Fail ::= **FAIL** | **F** | "(" **FAIL** ")" | "(" **F** ")"
443.Inconclusive ::= **INCONC** | **I** | "(" **INCONC** ")" | "(" **I** ")"
444.Result ::= **R**

TTCN statements

445.StatementLine ::= (Event [Qualifier] [AssignmentList] [TimerOps]) | (Qualifier [AssignmentList] [TimerOps]) | (AssignmentList [TimerOps]) | TimerOps | Construct | ImplicitSend
446.Event ::= Send | Receive | Otherwise | Timeout | Done
447.Qualifier ::= "[" Expression "]"
448.Send ::= [PCO_Identifier | CP_Identifier | FormalParIdentifier] "!" (ASP_Identifier | PDU_Identifier | CM_Identifier)
449.ImplicitSend ::= "<" **IUT** "!" (ASP_Identifier | PDU_Identifier) ">"
450.Receive ::= [PCO_Identifier | CP_Identifier | FormalParIdentifier] "?" (ASP_Identifier | PDU_Identifier | CM_Identifier)
451.Otherwise ::= [PCO_Identifier | FormalParIdentifier] "?" **OTHERWISE**
452.Timeout ::= "?" **TIMEOUT** [TimerIdentifier | FormalParIdentifier]
453.Done ::= "?" **DONE** "(" [TCompIdList] ")"
454.TCompIdList ::= TCompIdentifier {Comma TCompIdentifier}
455.Construct ::= GoTo | Attach | Repeat | Return | Activate | Create
456.Activate ::= **ACTIVATE** "(" [DefaultRefList] ")"
457.Return ::= **RETURN**
458.Create ::= **CREATE** "(" CreateList ")"
459.CreateList ::= CreateTComp {Comma CreateTComp}
460.CreateTComp ::= TCompIdentifier Colon TreeReference [ActualParList]
461.GoTo ::= ("->" | **GOTO**) Label

462.Attach ::= "+" TreeReference [ActualParList]
463.Repeat ::= **REPEAT** TreeReference [ActualParList] **UNTIL** Qualifier
464.TreeReference ::= TestStepIdentifier | TreeIdentifier
465.ActualParList ::= "(" ActualPar {Comma ActualPar} ")"
466.ActualPar ::= Value | PCO_Identifier | CP_Identifier | TimerIdentifier

Expressions

467.AssignmentList ::= "(" Assignment {Comma Assignment} ")"
468.Assignment ::= DataObjectReference ":=" Expression
469.Expression ::= SimpleExpression [RelOp SimpleExpression]
470.SimpleExpression ::= Term {AddOp Term}
471.Term ::= Factor {MultiplyOp Factor}
472.Factor ::= [UnaryOp] Primary
473.Primary ::= Value | DataObjectReference | OpCall | SelectExprIdentifier | "(" Expression ")"
474.DataObjectReference ::= DataObjectIdentifier {ComponentReference}
475.DataObjectIdentifier ::= TS_ParIdentifier | TS_ConstIdentifier | TS_VarIdentifier | TC_VarIdentifier | FormalParIdentifier | ASP_Identifier | PDU_Identifier | CM_Identifier
476.ComponentReference ::= RecordRef | ArrayRef | BitRef
477.RecordRef ::= Dot (ComponentIdentifier | PDU_Identifier | StructIdentifier | ComponentPosition)
478.ComponentIdentifier ::= ASP_ParIdentifier | PDU_FieldIdentifier | CM_ParIdentifier | ElemIdentifier | ASN1_Identifier
479.ASN1_Identifier ::= Identifier
480.ComponentPosition ::= ("Number")"
481.ArrayRef ::= Dot "[" ComponentNumber "]"
482.ComponentNumber ::= Expression
483.BitRef ::= Dot (BitIdentifier | "[" BitNumber "]")
484.BitIdentifier ::= Identifier
485.BitNumber ::= Expression
486.OpCall ::= OpIdentifier (ActualParList | "(" ")")
487.OpIdentifier ::= TS_OpIdentifier | PredefinedOpIdentifier
488.PredefinedOpIdentifier ::= BIT_TO_INT | HEX_TO_INT | INT_TO_BIT | INT_TO_HEX | IS_CHOSEN | IS_PRESENT | LENGTH_OF | NUMBER_OF_ELEMENT
489.AddOp ::= "+" | "-" | **OR**
490.MultiplyOp ::= "*" | "/" | **MOD** | **AND**
491.UnaryOp ::= "+" | "-" | **NOT**
492.RelOp ::= "=" | "<" | ">" | "<>" | ">=" | "<="

Timer operations

493.TimerOps ::= TimerOp {Comma TimerOp}
494.TimerOp ::= StartTimer | CancelTimer | ReadTimer
495.StartTimer ::= **START** TimerIdentifier ["(" TimerValue ")"]
496.CancelTimer ::= **CANCEL** [TimerIdentifier]
497.TimerValue ::= Expression
498.ReadTimer ::= **READTIMER** TimerIdentifier "(" DataObjectReference")"

Types

499.TypeOrPDU ::= Type | PDU
500.Type ::= PredefinedType | ReferenceType

Predefined types

501.PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | **R_Type** | CharacterString
502.CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString** | **T61String** | **ISO646String**

Referenced types

503.ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier | CM_Identifier
504.TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier

Values

505.Value ::= LiteralValue | ASN1_Value [ASN1_Encoding]
506.LiteralValue ::= Number | BooleanValue | Bstring | Hstring | Ostring | Cstring | R_Value
507.Number ::= (NonZeroNum {Num}) | **0**
508.NonZeroNum ::= **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
509.Num ::= **0** | NonZeroNum
510.BooleanValue ::= **TRUE** | **FALSE**
511.Bstring ::= "'" {Bin | Wildcard} "'" **B**
512.Bin ::= **0** | **1**
513.Hstring ::= "'" {Hex | Wildcard} "'" **H**
514.Hex ::= Num **| A | B | C | D | E | F**
515.Ostring ::= "'" {Oct | Wildcard} "'" **O**
516.Oct ::= Hex Hex
517.Cstring ::= """ {Char | Wildcard | "\"} """
518.Char ::= */* REFERENCE - A character defined by the relevant character string type */*

519.Wildcard ::= AnyOne | AnyOrNone
520.AnyOne ::= "?"
521.AnyOrNone ::= "*"
522.R_Value ::= **pass** | **fail** | **inconc** | **none**
523.Identifier ::= Alpha{AlphaNum | Underscore}
524.Alpha ::= UpperAlpha | LowerAlpha
525.AlphaNum ::= Alpha | Num
526.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**
527.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**
528.ExtendedAlphaNum ::= */* REFERENCE - A character from any character set defined in ISO/IEC 10646 */*
529.BoundedFreeText ::= *"/*"* FreeText *"*/"*
530.FreeText ::= {ExtendedAlphaNum}

Miscellaneous productions

531.Comma ::= ","
532.Dot ::= "."
533.Dash ::= "-"
534.Minus ::= "-"
535.SemiColon ::= ";"
536.Colon ::= ":"
537.Underscore ::= "_"

The ASN1 Syntax Productions in BNF

1. moduleidentifier ::= BoundedFreeText
2. typereference ::= Identifier
3. valuereference ::= Identifier
4. TypeAssignment ::= typereference "::=" ASN1_main_Type
5. ASN1_main_Type ::= BuiltinType | DefinedType | SubType
6. BuiltinType ::= BooleanType | IntegerType | BitStringType | OctetStringType | NullType | SequenceType | SequenceOfType | SetType | SetOfType | ChoiceType | SelectionType | TaggedType | AnyType | ObjectIdentifierType | CharacterStringType | UsefulType | EnumeratedType | RealType
7. NamedType ::= Identifier ASN1_main_Type | ASN1_main_Type | SelectionType
8. ASN1_Value ::= BuiltinValue | DefinedValue
9. BuiltinValue ::= BooleanValue | IntegerValue | BitStringValue | OctetStringValue | NullValue | SequenceValue | SequenceOfValue | SetValue | SetOfValue | ChoiceValue | SelectionValue | TaggedValue | ASN1_AnyValue | ObjectIdentifierValue | CharacterStringValue | EnumeratedValue | RealValue
10. NamedValue ::= Identifier ASN1_Value | ASN1_Value
11. BooleanType ::= BOOLEAN
12. IntegerType ::= INTEGER | INTEGER "{" NamedNumberList "}"
13. NamedNumberList ::= NamedNumber | NamedNumberList Comma NamedNumber
14. NamedNumber ::= Identifier "(" Minus Number ")" | Identifier "(" Number ")" | Identifier "(" DefinedValue ")"
15. IntegerValue ::= Minus Number | Number | Identifier
16. EnumeratedType ::= ENUMERATED "{" Enumeration "}"
17. Enumeration ::= NamedNumber | Enumeration Comma NamedNumber
18. EnumeratedValue ::= Identifier
19. RealType ::= REAL
20. RealValue ::= NumericalRealValue | SpecialRealValue
21. NumericalRealValue ::= "{" Mantissa Comma Base Comma Exponent "}"
22. Mantissa ::= Minus Number | Number
23. Base ::= Two | Ten
24. Exponent ::= Minus Number | Number
25. SpecialRealValue ::= PLUS_INFINITY | MINUS_INFINITY
26. BitStringType ::= BIT STRING | BIT STRING "{" NamedBitList "}"
27. NamedBitList ::= NamedBit | NamedBitList Comma NamedBitList
28. NamedBit ::= Identifier "(" Number ")" | Identifier "(" DefinedValue ")"
29. BitStringValue ::= Bstring | Hstring | "{" IdentifierList "}" | "{" "}"
30. IdentifierList ::= Identifier | IdentifierList Comma Identifier
31. OctetStringType ::= OCTET STRING
32. OctetStringValue ::= Bstring | Hstring

33. NullType ::= NULL
34. NullValue ::= NULL
35. SequenceType ::= SEQUENCE "{" ElementTypeList "}"
36. ElementTypeList ::= | ElementType | ElementTypeList Comma ElementType
37. ElementType ::= NamedType | NamedType OPTIONAL | NamedType DEFAULT ASN1_Value | COMPONENTS OF ASN1_main_Type
38. SequenceValue ::= "{" ElementValueList "}" | "{" "}"
39. ElementValueList ::= NamedValue | ElementValueList Comma NamedValue
40. SequenceOfType ::= SEQUENCE OF ASN1_main_Type | SEQUENCE
41. SequenceOfValue ::= LBRACE ASN1_ValueList RBRACE | LBRACE RBRACE
42. ASN1_ValueList ::= ASN1_Value | ASN1_ValueList Comma ASN1_Value
43. SetType ::= SET "{" ElementTypeList "}"
44. SetValue ::= "{" ElementValueList "}" | "{" "}"
45. SetOfType ::= SET OF ASN1_main_Type | SET
46. SetOfValue ::= "{" ASN1_ValueList "}" | "{" "}"
47. ChoiceType ::= CHOICE "{" AlternativeTypeList "}"
48. AlternativeTypeList ::= NamedType | AlternativeTypeList Comma NamedType
49. ChoiceValue ::= NamedValue
50. SelectionType ::= Identifier "<" ASN1_main_Type
51. SelectionValue ::= NamedValue
52. TaggedType ::= Tag ASN1_main_Type | Tag IMPLICIT ASN1_main_Type | Tag EXPLICIT ASN1_main_Type
53. Tag ::= "(" Class ClassNumber ")"
54. Class ::= UNIVERSAL | APPLICATION | PRIVATE | empty
55. ClassNumber ::= Number | DefinedValue
56. TaggedValue ::= ASN1_Value
57. AnyType ::= ANY | ANY DEFINED_ BY Identifier
58. ASN1_AnyValue ::= ASN1_main_Type Colon ASN1_Value
59. ObjectIdentifierType ::= OBJECT IDENTIFIER
60. ObjectIdentifierValue ::= "{" ObjIdComponentList "}" | "{" DefinedValue ObjIdComponentList "}"
61. ObjIdComponentList ::= ObjComponent | ObjIdComponentList ObjComponent
62. ObjComponent ::= NameForm | NumberForm | NameAndNumberForm
63. NameForm ::= Identifier
64. NumberForm ::= Number | DefinedValue
65. NameAndNumberForm ::= Identifier "{" NumberForm "}"
66. CharacterStringType ::= Identifier
67. CharacterStringValue ::= Cstring
68. UsefulType ::= Identifier
69. SubType ::= ParentType SubtypeSpec | SET SizeConstraint OF ASN1_main_Type | SEQUENCE SizeConstraint OF ASN1_main_Type

70. ParentType ::= ASN1_main_Type
71. SubtypeSpec ::= "(" SubtypeValueSetList ")"
72. SubtypeValueSetList ::= SubtypeValueSet | SubtypeValueSetList '|' SubtypeValueSet
73. SubtypeValueSet ::= ASN1_Value | ContainedSubtype | ASN1_ValueRange | PermittedAlphabet | SizeConstraint | InnerTypeConstraints
74. ContainedSubtype ::= INCLUDES ASN1_main_Type
75. ASN1_ValueRange ::= LowerEndpoint DotDot UpperEndpoint
76. LowerEndpoint ::= LowerEndValue | LowerEndValue "<"
77. LowerEndValue ::= ASN1_Value | MIN
78. UpperEndpoint ::= UpperEndValue | "<" UpperEndValue
79. UpperEndValue ::= ASN1_Value | MAX
80. SizeConstraint ::= SIZE SubtypeSpec
81. PermittedAlphabet ::= FROM SubtypeSpec
82. InnerTypeConstraints ::= WITH COMPONENT SingleTypeConstraint | WITH COMPONENTS MultipleTypeConstraints
83. SingleTypeConstraint ::= SubtypeSpec
84. MultipleTypeConstraints ::= FullSpecification | PartialSpecification
85. FullSpecification ::= "{" TypeConstraints "}"
86. PartialSpecification ::= "{" DotDotDot Comma TypeConstraints "}"
87. TypeConstraints ::= NamedConstraint | NamedConstraint Comma TypeConstraints
88. NamedConstraint ::= Identifier Constraint | Constraint
89. Constraint ::= ValueConstraint PresenceConstraint
90. ValueConstraint ::= SubtypeSpec | empty
91. PresenceConstraint ::= PRESENT | ABSENT | OPTIONAL | empty

TTCN Static Semantics

The following are those static semantics from the TTCN and ASN.1 standards which are checked by the Analyzer tool in the TTCN suite.

Test Suite

SuiteId shall be the same as the SuiteId declared in Test Suite Structure table (Suite Structure).

Test Case Index

Test Cases shall be listed in the order that they exist in the dynamic part.

Test Step Index

TestStepId shall not include a formal parameter list.

Test Steps shall be listed in the order that they exist in the dynamic part.

Default Index

DefaultId shall not include a formal parameter list.

Defaults shall be listed in the order that they exist in the dynamic part.

Test Suite Type Definitions

Wherever types are referenced within Test Suite Type Definitions those references shall not be recursive (neither directly or indirectly)

Simple Type Definitions

The base type[1] shall be a Predefined Type or a Simple Type.

The set of values defined by a restriction must be a true subset of the values of the base type.

In specification of a particular length range, only non-negative INTEGER literals or the keyword INFINITY for the upper bound shall be used.

1. By Base Type we refer to the particular type in the Type production

If Minus is used in SimpleValueList then LiteralValue shall be a number.

The restriction type shall be a list of distinguished values of the base type.

Where a range is used in a type definition either as a value range or as a length range (for strings) it shall be stated with the lower of the two values on the left.

An integer range shall be used only with a base type of INTEGER or a type derived from INTEGER.

Where a value list is used, the values shall be of the base type and shall be a true subset of the values defined by the base type.

LengthRestriction shall be provided only when the base type is a string type (i.e., BITSTRING, HEXSTRING, OCTETSTRING or CharacterString) or derived from a string type

The LiteralValues shall be of the base type

In RangeTypeLength:
LowerTypeBound shall be a non-negative number
LowerTypeBound shall be less than UpperTypeBound

In IntegerRange:
LowerTypeBound shall be less than UpperTypeBound

Structured Type Definitions

Where elements may be of a type of arbitrarily complex structure; there shall be no recursive references.

The elements of Structured Type definitions are considered to be optional, i.e., in instances of these types whole elements may not be present.

A structure element Type shall be a PredefinedType, TS_TypeIdentifier, PDU_Identifier, or PDU.

If a Structured Type is used as a macro expansion, then the names of the elements within the Structured Type shall be unique within each ASP or PDU where it will be expanded.

The optional element length restriction can be used in order to give the minimum and maximum length of an element of a string type.

The set of values defined by LengthAttribute shall be a subset of the values of the base type.

ASN1 Type Definition

Types referred to from the type definition shall be defined in other ASN.1 type definition tables, be defined by reference in the ASN.1 type reference table or be defined locally in the same table, following the first type definition. Locally defined types shall not be used in other parts of the test suite.

ASN.1 type definitions used within TTCN shall not use external type references as defined in ISO/IEC 8824.

When ASN1 is used in a TTCN test suite, ASN1 identifiers from the following list shall be unique throughout the test suite:
a) identifiers occurring in an ASN1ENUMERATED type as distinguished values
b) identifiers occurring in a NamedNumberList of an ASN1 INTEGER type

Each terminal type reference used within the Type production shall be one of the following: ASN1_LocalType type reference, TS_TypeIdentifier or PDU_Identifier.

Test Suite Operation Definition

Only predefined types and data types as defined in the Test Suite Type definitions, ASP type definitions or PDU type definitions may be used as types for formal parameters. PCO types shall not be used as formal parameter types.

When a Test Suite Operation is invoked

1. the number of the actual parameters shall be the same as the number of the formal parameters; and

2. each actual parameter shall evaluate to an element of its corresponding formal type of parameter's.

Type in ResultType shall be a Predefined Type, TS_TypeIdentifier, PDU_Identifier or ASP_Identifier.

Test Suite Parameters Declarations

The type shall be a Predefined Type, TS_TypeIdentifier, PDU_Identifier or ASP_Identifier.

Test Case Selection Expression Definition

The expression shall use only literal values, Test Suite Parameters, Test Suite Constants and other selection expression identifiers in its terms.

The expression shall evaluate to a BOOLEAN value.

Expression shall not recursively refer (neither directly nor indirectly) to the SelExprIdentifier being defined by that Expression.

Test Suite Constant Declarations

The type shall be a predefined type, an ASN.1 type, a Test Suite Type or a PDU type.

The terms in the value expression shall not contain: Test Suite Variables or Test Case Variables.

The value shall evaluate to an element of the type indicated in the type column.

Test Suite Variable Declarations

Type shall be a Predefined Type, TS_TypeIdentifier, PDU_Identifier or ASP_Identifier.

The terms in the value expression shall not contain: Test Suite Variables or Test Case Variables.

Specifying an initial value is optional.

If an unbound Test Suite Variable is used in the right-hand side of an assignment, then it is a test case error.

The value shall evaluate to an element of the type indicated in the type column.

Test Case Variable Declarations

Type shall be a Predefined Type, TS_TypeIdentifier, PDU_Identifier or ASP_Identifier.

The terms in the value expression shall not contain: Test Suite Variables or Test Case Variables.

Specifying an initial value is optional.

The value shall evaluate to an element of the type indicated in the type column.

PCO Declarations

It is possible to define a PCO to correspond to a *set* of SAPs.

Timer Declarations

The terms in the value expression shall not contain: Test Suite Variables or Test Case Variables.

The timer duration shall evaluate to an unsigned positive INTEGER value.

ASP Type Definition

ASP type definitions may include ASN1 type definitions, if appropriate.

If only a single PCO is defined within a test suite then PCO_TypeIdentifier is optional. The PCO type shall be one of the PCO types used in the PCO Type Declaration proforma.

The macro symbol shall be used only with Structured Types defined in the Structured Types definitions.

Parameters may be of a type of arbitrarily complex structure, including being specified as a Test Suite Type (either predefined, Simple Type, Structured Type or ASN.1 type).

If a parameter is to be structured as a PDU, then its type may be stated either:

- as a PDU identifier to indicate that in the constraint for the ASP this parameter may be chained to a PDU constraint of a specific PDU type.
- as **PDU** to indicate that in the constraint for the ASP this parameter may be chained to a PDU constraint of any PDU type.

The boundaries shall be specified in terms of (non-negative) INTEGER literals, Test Suite Parameters, Test Suite Constants or the keyword INFINITY.

The parameters of ASP type definitions are considered to be optional, i.e., in instances of these types whole parameters may not be present.

The names of ASP parameters shall be unique within the ASP in which they are declared.

The optional attribute is Length.

In ASPs that are sent from the tester, values for ASP parameters that are defined in the Constraints Part shall correspond to the parameter or field definition. This means:
a) the value shall be of the type specified for that ASP parameter; and
b) In the case of substructured ASPs, either using Structured Types or ASN.1, the above rules apply to the fields of the substructure(s) recursively.

ASN1 ASP Type Definitions

The PCO type shall be one of the PCO types used in the PCO declaration proforma.

If only a single PCO is defined within a test suite, specifying the PCO type in an ASP type definition is optional.

Types referred to from the ASP definition shall be defined in other ASN.1 type definition tables, be defined by reference in the ASN.1 type reference table or be defined locally in the same table, following the first type definition.

Locally defined types shall not be used in other parts of the test suite.

ASN1 ASP Type Definition By Reference

The PCO type shall be one of the PCO types used in the PCO declaration proforma.

If only a single PCO is defined within a test suite, specifying the PCO type in an ASP type definition is optional.

PDU Type Definition

The PCO type shall be one of the PCO types used in the PCO declaration proforma.

The macro symbol shall be used only with Structured Types defined in the Structured Types definitions.

Fields may be of a type of arbitrarily complex structure, including being specified as a Test Suite Type (either predefined, Simple Type, Structured Type or ASN.1 type).

If a field is to be structured as a PDU, then its type may be stated either

- as a PDU identifier to indicate that in the constraint for the PDU this field may be chained to a PDU constraint of a specific PDU type; or
- as PDU to indicate that in the constraint for the PDU this field may be chained to a PDU constraint of any PDU type.

The boundaries shall be specified in terms of non-negative INTEGER literals, Test Suite Parameters, Test Suite Constants or the keyword INFINITY.

The fields of PDU type definitions are considered to be optional, i.e., in instances of these types whole fields may not be present.

The names of PDU fields shall be unique within the PDU in which they are declared

The optional attribute is Length;

In PDUs that are sent from the tester, values for PDU fields that are defined in the Constraints Part shall correspond to the field definition. This means

1. that the value shall be of the type specified for that PDU field; and
2. in the case of substructured ASPs and/or PDUs, either using Structured Types or ASN.1, the above rules apply to the fields of the substructure(s) recursively.

The set of values defined by LengthAttribute shall be a true subset of the values of the base type.

LengthAttribute shall be provided only when the base type is a string type (i.e. BITSTRING, HEXSTRING, OCTETSTRING or CharacterString) or derived from a string type.

ASN1 PDU Type Definition

The PCO type shall be one of the PCO types used in the PCO declaration proforma.

If only a single PCO is defined within a test suite, specifying the PCO type in an PDU type definition is optional.

Types referred to from the ASP definition shall be defined in other ASN.1 type definition tables, be defined by reference in the ASN.1 type reference table or be defined locally in the same table, following the first type definition.

Locally defined types shall not be used in other parts of the test suite.

ASN1 PDU Type Definition By Reference

The PCO type shall be one of the PCO types used in the PCO declaration proforma.

If only a single PCO is defined within a test suite, specifying the PCO type in an PDU type definition is optional.

String Length Specifications

TTCN permits the specification of length restrictions on string types (i.e., BITSTRING, HEXSTRING, OCTETSTRING and all Character-String types) in the following instances:

1. when declaring Test Suite Types as a type restriction;
2. when declaring simple ASP parameters, PDU fields and elements of Structured Types as an attribute of the parameter, field or element type;
   and
3. when defining ASP/PDU or Structured Type constraints as an attribute of the constraint value. In the context of constraints, length restrictions can also be specified on values of type SEQUENCE OF or SET OF, thus limiting the number of their elements.

Alias Definitions

An Alias shall be used only to replace an ASP identifier or a PDU identifier within a single TTCN statement in a behaviour tree. It shall be used only in a behaviour description column.

Structured Type Constraint Declarations

If an ASP or PDU definition refers to a Structured Type as a substructure of a parameter or field (i.e., with a parameter name or a field name specified for it) then the corresponding constraint shall have the same parameter or field name in the corresponding position in the parameter name or field name column of the constraint and the value shall be a reference to a constraint for that parameter or field (i.e., for that substructure in accordance with the definition of the Structured Type).

ASP Constraint Declarations

If the ASP definition refers to a Structured Type by macro expansion (i.e., with <- in place of the ASP field name) then in a corresponding constraint either

- the individual elements from the Structured Type shall be included directly within the constraints.

- the macro symbol (<-) shall be placed in the corresponding position in the ASP field name column of the constraint and the value shall be a reference to a constraint for the Structured Type referenced from the ASP definition.

PDU Constraint Declarations

If the PDU definition refers to a Structured Type by macro expansion (i.e., with <- in place of the PDU field name) then in a corresponding constraint either:

- the individual elements from the Structured Type shall be included directly within the constraints;
  or
- the macro symbol (<-) shall be placed in the corresponding position in the PDU field name column of the constraint and the value shall be a reference to a constraint for the Structured Type referenced from the PDU definition.

Constraints Part

If an ASP and/or PDU is substructured, then the constraints for ASPs and/or PDUs of that type shall have the same tabular structure or a compatible ASN.1 structure (i.e., possibly with some groupings).

Structured Types expanded into an ASP or PDU definition by use of the macro symbol (<-) are not considered to be substructures. Constraints for such ASPs or PDUs shall either have a completely flat structure (i.e., the elements of an expanded structure are explicitly listed in the ASP or PDU constraint) or shall reference a corresponding structure constraint for macro expansion.

Whichever way the values are obtained, they shall correspond to the parameter or field entries in the ASP or PDU type definitions. This means

1. that the value shall be of the type specified for that parameter or field.
2. that the length shall satisfy any restriction associated with the type. (This will not be implemented.)

An expression in a constraint shall contain only literal values, Test Suite Parameters, Test Suite Constants, formal parameters and Test Suite Operations.

Neither Test Suite Variables nor Test Case Variables shall be used in constraints, unless passed as actual parameters.

Literal values, Test Suite Parameters, Test Suite Constants, Test Suite Variables, Test Case Variables and PDU or Test Suite Type constraints may be passed as actual parameters to a constraint in a constraints reference made from a behaviour description. The parameters shall not be of PCO type or ASP type.

In ASN.1 constraints, only ASP parameters and PDU fields declared as OPTIONAL may be omitted. These may be omitted either by using the Omit symbol or by simply leaving out the relevant ASP parameter or PDU field.

The constraint specification of an ASP and/or PDU shall have the same structure as that of the type definition of that ASP or PDU.

In tabular constraints, all ASP parameters and PDU fields are optional and therefore may be omitted using the Omit symbol, to indicate that the ASP parameter or PDU field is to be absent from the event sent.

An expression in a constraint shall contain only Values (including, for instance, ConstraintValue&Attributes), Test Suite Parameters, Test Suite Constraints, formal parameters, Component References and Test Suite Operations.

Matching Mechanisms

Complement: Each constraint value in the list shall be of the type declared for the ASP parameter or PDU field in which the complement mechanism is used.

ValueList: Each value in the ValueList shall be of the type declared for the ASP parameter or PDU field in which the ValueList mechanism is used.

Range: Ranges shall be used only on values of INTEGER type.

Range: A boundary value shall be either:

- INFINITY or -INFINITY
- a constraint expression that evaluates to a specific INTEGER value.

Range: The lower boundary shall be less than the upper boundary.

SuperSet: SuperSet is an operation for matching that shall be used on values of SET OF type. SuperSet shall be used only in ASN1 constraints.

SuperSet: The argument of SuperSet shall be of the type declared for the ASP parameter or PDU field in which the SuperSet mechanism is used.

SubSet: SubSet is an operation for matching that shall be used on values of SET OF type. SubSet shall be used only in ASN1 constraints.

SubSet: The argument of SubSet shall be of the type declared for the ASP parameter or PDU field in which the SubSet mechanism is used.

Permutation: Permutation an operation for matching that can be used only on values inside a value of SEQUENCE OF type. Permutation shall be used only in ASN1 constraints.

Permutation: Each element listed in Permutation shall be of the type declared inside the SEQUENCE OF type of the ASP parameter or PDU field.

Base Constraints and Modified Constraints

The name of the modified constraint shall be a unique identifier.

The name of the base constraint which is to be modified shall be indicated in the derivation path entry in the constraint header. This entry shall be left blank for a base constraint.

A modified constraint can itself be modified. In such a case the Derivation Path indicates the concatenation of the names of the base and previously modified constraints, separated by dots (.) A dot shall follow the last modified constraint name.

If a base constraint is defined to have a formal parameter list, the following rules apply to all modified constraints derived from that base

constraint, whether or not they are derived in one or several modification steps:

1. The modified constraint shall have the same parameter list as the base constraint. In particular, there shall be no parameters omitted from or added to this list
2. The formal parameter list shall follow the constraint name for every modified constraint
3. Parameterized ASP parameters or PDU in a base constraint fields shall not be modified or explicitly omitted in a modified constrain.

In tabular constraints Omit shall be denoted by dash (-). In ASN.1 constraints Omit is denoted by OMIT.

If the ASP or PDU definition refers to a parameter or field specified as being of metatype PDU then in a corresponding constraint the value for that parameter or field shall be specified as the name of a PDU constraint, or formal parameter.

The Behaviour Description

Statements in the first level of alternatives having no predecessor in the root or local tree belong to, shall have the indentation value of zero.

Statements having a predecessor shall have the indentation value of the predecessor plus one as their indentation value.

The parameters may provide PCOs, constraints, variables, or other such items for use within the tree.

Test Case root trees shall not be parameterized.

Formal parameters may be of PCO type, ASP type, PDU type, structure type or one of the other predefined or Test Suite Types.

TTCN Test Events

In the simplest form, an ASP identifier or PDU identifier follows the SEND symbol (!) for events to be initiated by the LT or UT, or a RECEIVE symbol (?) for events which it is possible for the LT or UT to accept.

If both a qualifier and an assignment are associated with the same event, then the qualifier shall appear first.

The tree header identifier used for local trees shall be unique within the dynamic behaviour description in which they appear, and shall not be the same as any identifier having a unique meaning throughout the test suite.

TTCN Expressions

The index notation is used to refer to elements (bits) of the ASN.1 BIT-STRING type. BITSTRING is assumed to be defined as SEQUENCE OF {BOOLEAN}. If certain bits of a BITSTRING are associated with an identifier (named bit) then either the dot notation or this identifier shall be used to refer to the bit.

Where a parameter, field or element is defined to be a true substructure of a type defined in a Structured Type table, a reference to the elements in the substructure shall consist of the reference to the parameter, field or element identifier followed by a dot and the identifier of the item within that substructure.

Where a structure is used as a macro expansion, the elements in the structure shall be referred to as if it was expanded into the structure referring to it.

If a parameter, field or element is defined to be of metatype PDU no reference shall be made to fields of that substructure.

The ATTACH Construct

Tree reference may be Test Step Identifiers or tree identifiers, where

1. A Test Step Identifier denotes the attachment of a Test Step that resides in the Test Step Library; the Test Step is referenced by its unique identifier
2. A tree identifier shall be the name of one of the trees in the current behaviour description; this is attachment of a local tree.

Constraints may be passed as parameters to Test Steps. If the constraint has a formal parameter list then the constraint shall be passed together with an actual parameter list.

When a parameterized tree is attached:

1. The number of the actual parameters shall be the same as the number of formal parameters

2. Each actual parameter shall evaluate to an element of its corresponding formal parameter type

Labels and the GOTO Construct

A GOTO to a label may be specified within a behaviour tree provided that the label is associated with the first of a set of alternatives, one of which is an ancestor node of the point from which the GOTO is to be made.

A GOTO shall be used only for jumps within one tree, i.e., within a Test Case root tree, a Test Step tree a Default tree or a local tree. As a consequence, each label used in a GOTO construct shall be found within the same tree in which the GOTO is used.

Labels used within a tree shall be unique within a tree.

The Constraints Reference

The actual parameter list shall fulfil the following:

1. the number of actual parameters shall be the same as the number of formal parameters;
   and

2. each actual parameter shall evaluate to an element of its corresponding formal type.

Verdicts

A predefined variable called R is available in each Test Case to store any Intermediate results. R can take the values *pass*, *fail*, *inconc* and *none*. These values are predefined identifiers and as such are case sensitive.

R shall not be used on the left-hand side of an assignment statement.

PASS or P, FAIL or F and INCONC or I are keywords that are used in the verdicts column only. The predefined identifiers pass, fail, inconc and none are values that represent the possible contents of the pre-

defined variable R. These predefined identifiers are to be used for testing the variable R in behaviour lines only.

A Verdict shall not occur in corresponding to entries in the behaviour tree which are any of the following: empty, an ATTACH construct, a GOTO construct, an IMPLICIT SEND or a RETURN.

Default References

Test Cases or Test Steps shall not be referred to as Defaults.

The actual parameter list shall fulfill the following:

1. The number of actual parameters shall be the same as the number of formal parameters.
2. Each actual parameter shall evaluate to an element of its corresponding formal type.
3. All variables appearing in the parameter list shall be bound when the constraint is invoked. (This won't be implemented.)

Formal Parameters

The formal parameter names which may optionally appear as part of the following shall be unique within that formal parameter list, and shall not be the same as any identifier having a unique meaning throughout the test suite.

A formal parameter name contained in the formal parameter list of a local tree shall header shall take precedence over a formal parameter name contained in the formal parameter list of the Test Step in which it is defined, within the scope of that local formal parameter list.

DataObjectReferences

A reference to a component of one of the following types: SEQUENCE, SET and CHOICE is constructed using a dot notation; i.e., appending a dot and the name (component identifier) of the desired component to the data object identifier; the component identifier shall be used if specified.

ASN.1 Static Semantics

IntegerType: Each identifier appearing in the NamedNumberList shall be different.

BitStringType: The DefinedValue shall be a reference to a value of type integer, or of a type derived from integer by tagging.

BitStringType: Each identifier appearing in the NamedNumberList shall be different.

BitStringValue: Each identifier in BitStringValue shall be the same as an identifier in the BitStringType sequence with which the value is associated.

SequenceType: The Type in the fourth alternative of the ElementType (COMPONENTS OF) shall be sequence type.

SequenceType: If OPTIONAL or DEFAULT are present, the corresponding value may be omitted from a value of the new type.

SequenceType: The identifiers in all NamedType sequence of the ElementTypeList shall be distinct.

SequenceType: The {} notation shall only be used if:
a) all ElementType sequences in the SequenceType are marked DEFAULT or OPTIONAL and all values are omitted
b) the type notation was SEQUENCE {}

SequenceType: There shall be one NamedValue for each NamedType in the SequenceType which is not marked OPTIONAL or DEFAULT, and the values shall be in the same order as the corresponding NamedType sequences.

SequenceOfType: Each Value sequence in the ValueList shall be the notation for a value of the Type specified in the SequenceOfType.

SequenceOfType: The {} notation is used when there are no component values in the sequence-of value.

SetType: The Type in the fourth alternative of the ElementType (COMPONENTS OF) shall be set type.

SetType: If OPTIONAL or DEFAULT are present, the corresponding value may be omitted from a value of the new type.

SetType: The identifiers in all NamedType sequence of the ElementTypeList shall be distinct.

SetType: The {} notation shall only be used if:
a) all ElementType sequences in the SetType are marked DEFAULT or OPTIONAL and all values are omitted
b) the type notation was SET {}

SetOfType: Each Value sequence in the ValueList shall be the notation for a value of the Type specified in the SetOfType.

SetOfType: The {} notation is used when there are no component values in the set-of value.

ChoiceType: The identifiers in all NamedType sequences of the AlternativeTypeList shall be distinct.

ChoiceType: If the NamedValue contains an identifier (in our case it contains always), it shall be a notation for a value of that type in the AlternativeTypeList that is named by the same identifier.

SelectionType: Type is a notation referencing the ChoiceType, and identifier is the identifier in the NamedType.

Subtype: When the SubtypeSpec notation follows the SelectionType notation, the parent type is the SelectionType, not the Type in the SelectionType notation.

Subtype: When the SubtypeSpec notation follows a set-of or sequence-of type notation, it applies to the Type in the set-of or sequence-of notation, not to the set-of or sequence-of type.