

INTERNATIONAL STANDARD

ISO/IEC
9646-3

First edition
1992-10-01

9

Information technology — Open Systems Interconnection — Conformance testing methodology and framework —

Part 3:

The Tree and Tabular Combined Notation (TTCN)

*Technologies de l'information — Interconnexion de systèmes ouverts —
Essais de conformité — Méthodologie générale et procédures —*

Partie 3: Notation combinée arborescente et tabulaire (TTCN)



Reference number
ISO/IEC 9646-3:1992(E)



Documents
Document Sharing Hub

Contents

Page

1 Scope	1
2 Normative references	1
3 Definitions	2
3.1 Basic terms from ISO/IEC 9646-1	2
3.2 Terms from ISO 7498	3
3.3 Terms from ISO/TR 8509	4
3.4 Terms from ISO/IEC 8824	4
3.5 Terms from ISO/IEC 8825	4
3.6 TTCN specific terms	4
4 Abbreviations	6
4.1 Abbreviations defined in ISO/IEC 9646-1	6
4.2 Abbreviations defined in ISO/IEC 9646-2	7
4.3 Other abbreviations	7
5 The syntax forms of TTCN	7
6 Compliance	8
7 Conventions	8
7.1 Introduction	8
7.2 Syntactic metanotation	8
7.3 TTCN.GR table proformas	9
7.3.1 Introduction	9
7.3.2 Single TTCN object tables	9
7.3.3 Multiple TTCN object tables	10
7.3.4 Alternative compact tables	10
7.3.5 Specification of proformas	10
7.4 Free Text and Bounded Free Text	10
8 TTCN test suite structure	11
8.1 Introduction	11
8.2 Test Group References	11
8.3 Test Step Group References	11
8.4 Default Group References	11
8.5 Components of a TTCN test suite	12
9 Test Suite Overview	12
9.1 Introduction	12
9.2 Test Suite Structure	12
9.3 Test Case Index	14
9.4 Test Step Index	15
9.5 Default Index	15
10 Declarations Part	17
10.1 Introduction	17
10.2 TTCN types	17
10.2.1 Introduction	17
10.2.2 Predefined TTCN types	17
10.2.3 Test Suite Type Definitions	19
10.2.3.1 Introduction	19
10.2.3.2 Simple Type Definitions using tables	19
10.2.3.3 Structured Type Definitions using tables	20
10.2.3.4 Test suite type definitions using ASN.1	21
10.2.3.5 ASN.1 Type Definitions by Reference	22
10.3 TTCN operators and TTCN operations	24
10.3.1 Introduction	24

© ISO/IEC 1992

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case Postale 56 • CH-1211 Genève 20 • Switzerland. Printed in Switzerland.

10.3.2	TTCN operators	24
10.3.2.1	Introduction	24
10.3.2.2	Predefined arithmetic operators	24
10.3.2.3	Predefined relational operators	24
10.3.2.4	Predefined Boolean operators	25
10.3.3	Predefined operations	25
10.3.3.1	Introduction	25
10.3.3.2	Predefined conversion operations	25
10.3.3.3	Other predefined operations	26
10.3.4	Test Suite Operation Definitions	27
10.4	Test Suite Parameter Declarations	29
10.5	Test Case Selection Expression Definitions	29
10.6	Test Suite Constant Declarations	30
10.7	TTCN variables	31
10.7.1	Test Suite Variable Declarations	31
10.7.2	Binding of Test Suite Variables	32
10.7.3	Test Case Variable Declarations	32
10.7.4	Binding of Test Case Variables	33
10.8	PCO Declarations	33
10.9	Timer Declarations	34
10.10	ASP Type Definitions	35
10.10.1	Introduction	35
10.10.2	ASP Type Definitions using tables	35
10.10.3	Use of Structured Types within ASP Type Definitions	37
10.10.4	ASP Type Definitions using ASN.1	37
10.10.5	ASN.1 ASP Type Definitions by Reference	38
10.11	PDU Type Definitions	39
10.11.1	Introduction	39
10.11.2	PDU Type Definitions using tables	39
10.11.3	Use of Structured Types within PDU definitions	41
10.11.4	PDU Type Definitions using ASN.1	41
10.11.5	ASN.1 PDU Type Definitions by Reference	43
10.12	String length specifications	43
10.13	ASP and PDU Definitions for SEND events	44
10.14	ASP and PDU Definitions for RECEIVE events	44
10.15	Alias Definitions	45
10.15.1	Introduction	45
10.15.2	Expansion of Aliases	45
11	Constraints Part	46
11.1	Introduction	46
11.2	General principles	46
11.3	Parameterization of constraints	46
11.4	Chaining of constraints	47
11.5	Constraints for SEND events	47
11.6	Constraints for RECEIVE events	47
11.6.1	Matching values	47
11.6.2	Matching mechanisms	47
11.6.3	Specific Value	48
11.6.4	Instead of Value	49
11.6.4.1	Complement	49
11.6.4.2	Omit	49
11.6.4.3	AnyValue	49
11.6.4.4	AnyOrOmit	49
11.6.4.5	ValueList	50
11.6.4.6	Range	50
11.6.4.7	SuperSet	50
11.6.4.8	SubSet	51
11.6.5	Inside Values	51
11.6.5.1	AnyOne	51
11.6.5.2	AnyOrNone	51
11.6.5.3	Permutation	52
11.6.6	Attributes of values	52

11.6.6.1	Length	52
11.6.6.2	IfPresent	53
12	Specification of constraints using tables	53
12.1	Introduction	53
12.2	Structured Type Constraint Declarations	53
12.3	ASP Constraint Declarations	54
12.4	PDU Constraint Declarations	55
12.5	Parameterization of constraints	57
12.6	Base constraints and modified constraints	57
12.7	Formal parameter lists in modified constraints	57
13	Specification of constraints using ASN.1	58
13.1	Introduction	58
13.2	ASN.1 Type Constraint Declarations	58
13.3	ASN.1 ASP Constraint Declarations	59
13.4	ASN.1 PDU Constraint Declarations	60
13.5	Parameterized ASN.1 constraints	61
13.6	Modified ASN.1 constraints	61
13.7	Formal parameter lists in modified ASN.1 constraints	61
13.8	ASP Parameter and PDU field names within ASN.1 constraints	61
14	The Dynamic Part	63
14.1	Introduction	63
14.2	Test Case dynamic behaviour	63
14.2.1	Specification of the Test Case Dynamic Behaviour table	63
14.2.2	The Test Case Dynamic Behaviour proforma	64
14.2.3	Structure of the Test Case behaviour	65
14.2.4	Line numbering and continuation	65
14.3	Test Step dynamic behaviour	66
14.3.1	Specification of the Test Step Dynamic Behaviour table	66
14.3.2	The Test Step Dynamic Behaviour proforma	66
14.4	Default dynamic behaviour	67
14.4.1	Default behaviour	67
14.4.2	Specification of the Default Dynamic Behaviour table	68
14.4.3	The Default Dynamic Behaviour proforma	68
14.5	The behaviour description	69
14.6	The tree notation	69
14.7	Tree names and parameter lists	70
14.7.1	Introduction	70
14.7.2	Trees with parameters	70
14.8	TTCN statements	70
14.9	TTCN test events	71
14.9.1	Sending and receiving events	71
14.9.2	Receiving events	71
14.9.3	Sending events	71
14.9.4	Lifetime of events	71
14.9.5	Execution of the behaviour tree	72
14.9.5.1	Introduction	72
14.9.5.2	The concept of snapshot semantics	73
14.9.5.3	Restrictions on using events	73
14.9.6	The IMPLICIT SEND event	74
14.9.7	The OTHERWISE event	75
14.9.8	The TIMEOUT event	75
14.10	TTCN expressions	76
14.10.1	Introduction	76
14.10.2	References for ASN.1 defined data objects	77
14.10.3	References for data objects defined using tables	78
14.10.4	Assignments	78
14.10.4.1	Introduction	78
14.10.4.2	Assignment rules for string types	78
14.10.5	Qualifiers	79
14.10.6	Event lines with assignments and qualifiers	79
14.11	Pseudo-events	80
14.12	Timer management	80

14.12.1	Introduction	80
14.12.2	The START operation	80
14.12.3	The CANCEL operation	81
14.12.4	The READ TIMER operation	81
14.13	The ATTACH construct	82
14.13.1	Introduction	82
14.13.2	Scope of tree attachment	82
14.13.3	Tree attachment basics	82
14.13.4	The meaning of tree attachment	83
14.13.5	Passing parameterized constraints	85
14.13.6	Recursive tree attachment	85
14.13.7	Tree attachment and Defaults	85
14.14	Labels and the GOTO construct	85
14.15	The REPEAT construct	86
14.16	The Constraints Reference	87
14.16.1	Purpose of the Constraints Reference column	87
14.16.2	Passing parameters in Constraint References	87
14.16.3	Constraints and qualifiers and assignments	88
14.17	Verdicts	88
14.17.1	Introduction	88
14.17.2	Preliminary results	88
14.17.3	Final verdict	89
14.17.4	Verdicts and OTHERWISE	89
14.18	The meaning of Defaults	89
14.18.1	Introduction	89
14.18.2	Defaults and tree attachment	90
14.19	Default References	92
15	Page continuation	94
15.1	Page continuation of TTCN tables	94
15.2	Page continuation of dynamic behaviour tables	94

Annexes

A	(normative) Syntax and static semantics of TTCN	96
A.1	Introduction	96
A.2	Conventions for the syntax description	96
A.2.1	Syntactic metanotation	96
A.2.2	TTCN.MP syntax definitions	96
A.3	The TTCN.MP syntax productions in BNF	98
A.3.1	Test suite	98
A.3.2	The Test Suite Overview	98
A.3.2.1	General	98
A.3.2.2	Test Suite Structure	98
A.3.2.3	Test Case Index	98
A.3.2.4	Test Step Index	98
A.3.2.5	Default Index	98
A.3.3	The Declarations Part	98
A.3.3.1	General	98
A.3.3.2	Definitions	99
A.3.3.2.1	General	99
A.3.3.2.2	Test Suite Type Definitions	99
A.3.3.2.3	Simple Type Definitions	99
A.3.3.2.4	Structured Type Definitions	99
A.3.3.2.5	ASN.1 Type Definitions	100
A.3.3.2.6	ASN.1 Type Definitions by Reference	100
A.3.3.2.7	Test Suite Operation Definitions	100
A.3.3.3	Parameterization and Selection	100
A.3.3.3.1	General	100
A.3.3.3.2	Test Suite Parameter Declarations	101
A.3.3.3.3	Test Case Selection Expression Definitions	101
A.3.3.4	Declarations	101

A.3.3.4.1	General	101
A.3.3.4.2	Test Suite Constant Declarations	101
A.3.3.4.3	Test Suite Variable Declarations	101
A.3.3.4.4	Test Case Variable Declarations	101
A.3.3.4.5	PCO Declarations	102
A.3.3.4.6	Timer Declarations	102
A.3.3.5	ASP and PDU Type Definitions	102
A.3.3.5.1	General	102
A.3.3.5.2	ASP Type Definitions	102
A.3.3.5.3	Tabular ASP Type Definitions	102
A.3.3.5.4	ASN.1 ASP Type Definitions	103
A.3.3.5.5	ASN.1 ASP Type Definitions by Reference	103
A.3.3.5.6	PDU Type Definitions	103
A.3.3.5.7	Tabular PDU Type Definitions	103
A.3.3.5.8	ASN.1 PDU Type Definitions	104
A.3.3.5.9	ASN.1 PDU Type Definitions by Reference	104
A.3.3.5.10	Alias Definitions	104
A.3.4	The Constraints Part	104
A.3.4.1	General	104
A.3.4.2	Test Suite Type Constraint Declarations	104
A.3.4.3	Structured Type Constraint Declarations	104
A.3.4.4	ASN.1 Type Constraint Declarations	104
A.3.4.5	ASP Constraint Declarations	105
A.3.4.6	Tabular ASP Constraint Declarations	105
A.3.4.7	ASN.1 ASP Constraint Declarations	105
A.3.4.8	PDU Constraint Declarations	105
A.3.4.9	Tabular PDU Constraint Declarations	105
A.3.4.10	ASN.1 PDU Constraint Declarations	107
A.3.5	The Dynamic Part	107
A.3.5.1	General	107
A.3.5.2	Test Cases	107
A.3.5.3	Test Step Library	108
A.3.5.4	Default Library	108
A.3.5.5	Behaviour descriptions	108
A.3.5.6	Behaviour lines	109
A.3.5.7	TTCN statements	109
A.3.5.8	Expressions	110
A.3.5.9	Timer operations	112
A.3.6	Types	112
A.3.6.1	General	112
A.3.6.2	Predefined types	112
A.3.6.3	Referenced types	112
A.3.7	Values	112
A.3.8	Miscellaneous productions	113
A.4	General static semantics requirements	114
A.4.1	Introduction	114
A.4.2	Uniqueness of identifiers	114
A.5	Differences between TTCN.GR and TTCN.MP	117
A.5.1	Differences in syntax	117
A.5.2	Additional static semantics in the TTCN.MP	117
B (normative)	Operational semantics of TTCN	118
B.1	Introduction	118
B.2	Precedence	118
B.3	Processing of test case errors	118
B.4	Transformation algorithms	119
B.4.1	Introduction	119
B.4.2	Appending default behaviour	119
B.4.3	Removal of REPEAT constructs	120
B.4.4	Expanding ATTACHED trees	120
B.5	TTCN operational semantics	121
B.5.1	Introduction	121
B.5.2	Introduction to the pseudo-code notation	121

B.5.3	Execution of a test case	121
B.5.3.1	Execution of a Test Case - pseudo-code	121
B.5.3.2	Execution of a Test Case - natural language description	122
B.5.4	Functions for TTCN events	123
B.5.4.1	Functions for TTCN events - pseudo-code	123
B.5.4.2	Functions for TTCN events - natural language description	123
B.5.5	Execution of the SEND event	123
B.5.5.1	Execution of the SEND event - pseudo-code	123
B.5.5.2	Execution of the SEND event - natural language description	124
B.5.6	Execution of the RECEIVE event	124
B.5.6.1	Execution of the RECEIVE event - pseudo-code	124
B.5.6.2	Execution of the RECEIVE event - natural language description	125
B.5.7	Execution of the OTHERWISE event	126
B.5.7.1	Execution of the OTHERWISE event - pseudo-code	126
B.5.7.2	Execution of the OTHERWISE event - natural language description	126
B.5.8	Execution of the TIMEOUT event	127
B.5.8.1	Execution of the TIMEOUT event - pseudo-code	127
B.5.8.2	Execution of the TIMEOUT event - natural language description	127
B.5.9	Execution of the IMPLICIT SEND event	128
B.5.9.1	Execution of the IMPLICIT SEND event - pseudo-code	128
B.5.9.2	Execution of IMPLICIT SEND - natural language description	128
B.5.10	Execution of the PSEUDO-EVENT	128
B.5.10.1	Execution of PSEUDO-EVENTS - pseudo-code	128
B.5.10.2	Execution of PSEUDO-EVENTS - natural language description	129
B.5.11	Execution of BOOLEAN expressions	129
B.5.11.1	Execution of BOOLEAN expressions - pseudo-code	129
B.5.11.2	Execution of BOOLEAN expressions - natural language description	129
B.5.12	Execution of ASSIGNMENTS	129
B.5.12.1	Execution of EXECUTE_ASSIGNMENT - pseudo-code	129
B.5.12.2	Execution of ASSIGNMENTS - natural language description	130
B.5.13	Execution of TIMER operations	130
B.5.13.1	Execution of TIMER operations - pseudo-code	130
B.5.13.2	Execution of START timer - natural language description	130
B.5.13.3	CANCEL timer - natural language description	131
B.5.13.4	READTIMER - natural language description	131
B.5.14	Functions for TTCN constructs	131
B.5.14.1	Functions for TTCN constructs - pseudo-code	131
B.5.14.2	Functions for TTCN constructs - natural language description	131
B.5.15	Execution of the GOTO construct	132
B.5.15.1	Execution of the GOTO construct - pseudo-code	132
B.5.15.2	Execution of the GOTO construct - natural language description	132
B.5.16	The VERDICT	132
B.5.16.1	The VERDICT - pseudo-code	132
B.5.16.2	The VERDICT - natural language description	132
B.5.17	The Conformance Log	133
B.5.17.1	The LOG - pseudo-code	133
B.5.17.2	The conformance log - natural language description	133
B.5.18	Other miscellaneous functions used by the pseudo-code	133

C (normative) Compact proformas	135
C.1 Introduction	135
C.2 Compact proformas for constraints	135
C.2.1 Requirements	135
C.2.2 Compact proformas for ASP constraints	135
C.2.3 Compact proformas for PDU constraints	136
C.2.3.1 Introduction	136
C.2.3.2 Parameterized compact constraints	137
C.2.4 Compact proformas for Structured Type constraints	138
C.2.5 Compact proformas for ASN.1 constraints	140
C.3 Compact proforma for Test Cases	141
C.3.1 Requirements	141
C.3.2 Compact proforma for Test Case dynamic behaviours	141
D (informative) Examples	143
D.1 Examples of tabular constraints	143
D.1.1 ASP and PDU definitions	143
D.1.2 ASP/PDU constraints	144
D.2 Examples of ASN1 constraints	147
D.2.1 ASP and PDU definitions	147
D.2.2 ASN.1 ASP/PDU constraints	148
D.2.3 Further examples of ASN.1 constraints	152
D.3 Base and modified constraints	154
D.4 Type definition using macros	155
D.5 Use of REPEAT	156
D.6 Test suite operations	157
D.7 Example of a Test Suite Overview	157
D.8 Example of a Test Case in TTCN.MP Form	159
E (informative) Style guide	162
E.1 Introduction	162
E.2 Test case structure	162
E.3 Use of TTCN with different abstract test methods	163
E.3.1 Introduction	163
E.3.2 TTCN and the LS test method	163
E.3.3 TTCN and the DS test method	163
E.3.4 TTCN and the CS test method	163
E.3.5 TTCN and the RS test method	164
E.4 Use of Defaults	164
E.5 Limiting the execution time of a Test Case	164
E.6 Structured Types	164
E.7 Abbreviations	165
E.8 Test descriptions	165
E.9 Assignments on SEND events	165
E.10 Multi-service PCOs	165
F (informative) Summary of differences between Draft International Standard and International Standard versions of TTCN	166
F.1 Summary of differences	166
F.1.1 General	166
F.1.2 Convergence with ASN.1	166
F.1.3 Static semantics	166
F.1.4 Table layout	166
F.1.5 Test suite overview	166
F.1.6 Declarations	166
F.1.7 Constraints	167
F.1.8 Behaviour part	167
F.1.9 Verdicts, defaults and OTHERWISE	167
F.2 Summary of major changes	168
G (informative) List of BNF production numbers	169
G.1 Introduction	169
G.2 The production index	169
H (informative) Index of part 3	173
H.1 Introduction	173
H.2 The Index	173

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 9646-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO/IEC 9646 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — Conformance testing methodology and framework*:

- *Part 1: General concepts*
- *Part 2: Abstract test suite specification*
- *Part 3: The Tree and Tabular Combined Notation (TTCN)*
- *Part 4: Test realization*
- *Part 5: Requirements on test laboratories and clients for the conformance assessment process*
- *Part 6: Protocol profile test specification*
- *Part 7: Implementation conformance statement — Requirements and guidance on ICS and ICS proformas*

Annexes A, B and C form an integral part of this part of ISO/IEC 9646. Annexes D, E, F, G and H are for information only.

Introduction

This part of ISO/IEC 9646, one of a multi-part International Standard defines an informal test notation, called the Tree and Tabular Combined Notation (TTCN), for use in the specification of OSI abstract conformance test suites.

In constructing a standardized abstract test suite, a test notation is used to describe abstract test cases. The test notation can be an informal notation (without formally defined semantics) or a formal description technique (FDT). TTCN is an informal notation with clearly defined, but not formally defined, semantics.

TTCN is designed to meet the following objectives:

- a) to provide a notation in which abstract test cases can be expressed in standardized test suites;
- b) to provide a notation which is independent of test methods, layers and protocols;
- c) to provide a notation which reflects the abstract testing methodology defined in ISO/IEC 9646.

In the abstract testing methodology a test suite is looked upon as a hierarchy ranging from the complete test suite, through test groups, test cases and test steps, down to test events. TTCN provides a naming structure to reflect the position of test cases in this hierarchy. It also provides the means of structuring test cases as a hierarchy of test steps culminating in test events. In TTCN the basic test events are sending and receiving Abstract Service Primitives (ASPs), Protocol Data Units (PDUs) and timer events.

Two forms of the notation are provided: a human-readable tabular form, called TTCN.GR, for use in OSI conformance test suite standards; and a machine-processable form, called TTCN.MP, for use in representing TTCN in a canonical form within computer systems and as the syntax to be used when transferring TTCN test cases between different computer systems. The two forms are semantically equivalent.

This part of ISO/IEC 9646 is also to be published by CCITT as Recommendation X.292 (1992).

Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)

1 Scope

This part of ISO/IEC 9646 defines an informal test notation, called the Tree and Tabular Combined Notation (TTCN), for OSI conformance test suites, which is independent of test methods, layers and protocols, and which reflects the abstract testing methodology defined in ISO/IEC 9646-1 and ISO/IEC 9646-2.

It also specifies requirements and provides guidance for using TTCN in the specification of system-independent conformance test suites for one or more OSI standards. It specifies two forms of the notation: one, a human-readable form, applicable to the production of conformance test suite standards for OSI protocols; and the other, a machine-processable form, applicable to processing within and between computer systems.

This part of ISO/IEC 9646 applies to the specification of conformance test cases which can be expressed abstractly in terms of control and observation of protocol data units and abstract service primitives. Nevertheless, for some protocols, test cases may be needed which cannot be expressed in these terms. The specification of such test cases is outside the scope of this part of ISO/IEC 9646, although those test cases may need to be included in a conformance test suite standard.

NOTE 1 - For example, some static conformance requirements related to an application service may require testing techniques which are specific to that particular application.

This part of ISO/IEC 9646 specifies requirements on what a test suite standard may specify about a conforming realization of the test suite, including the operational semantics of TTCN test suites.

NOTE 2 - ISO/IEC 9646-4 specifies requirements concerning test realization including ETS derivation.

This part of ISO/IEC 9646 applies to the specification of conformance test suites for OSI protocols in OSI layers 2 to 7, specifically including Abstract Syntax Notation One (ASN.1) based protocols. The following are outside the scope of this part of ISO/IEC 9646:

- a) the specification of conformance test suites for multi-peer or Physical layer protocols;
- b) the relationship between TTCN and formal description techniques;
- c) the specification of test cases in which more than one behaviour description is to be run concurrently;

NOTE 3 - Use of parallel trees and synchronization between them is to be covered by a future amendment to this part of ISO/IEC 9646.

- d) the means of realization of executable test suites (ETS) from abstract test suites.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9646. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 9646 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 646 : 1991, *Information technology - ISO 7-bit coded character set for information interchange*.

ISO 7498 : 1984, *Information processing systems - Open Systems Interconnection - Basic Reference Model*.
(See also CCITT Recommendation X.200 : 1988.)

ISO/TR 8509 : 1987, *Information processing systems - Open Systems Interconnection - Service conventions*.
(See also CCITT Recommendation X.210 : 1988.)

ISO/IEC 8824 : 1990, *Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)*.
(See also CCITT Recommendation X.208 : 1988.)

ISO/IEC 8825 : 1990, *Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*.
(See also CCITT Recommendation X.209 : 1988)

ISO/IEC 9646-1 : 1991, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts*.
(See also CCITT Recommendation X.290 : - ¹⁾)

ISO/IEC 9646-2 : 1991, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 2: Abstract test suite specification*.
(See also CCITT Recommendation X.291 : - ¹⁾)

ISO/IEC 9646-4 : 1991, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 4: Test realization*.
(See also CCITT Recommendation X.293 : - ¹⁾)

ISO/IEC 9646-5 : 1991, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 5: Requirements on test laboratories and clients for the conformance assessment process*. (See also CCITT Recommendation X.294 : - ¹⁾)

ISO/IEC 10646-1 : - ¹⁾, *Information technology - Multiple-Octet Coded Character Set - Part 1: Architecture and Basic Multilingual Plane*.

3 Definitions

3.1 Basic terms from ISO/IEC 9646-1

The following terms defined in ISO/IEC 9646-1 apply:

- a) abstract service primitive
- b) abstract testing methodology
- c) abstract test case
- d) abstract test method
- e) abstract test suite
- f) conformance log
- g) conformance test suite
- h) coordinated test method
- i) distributed test method
- j) executable test case
- k) executable test case error
- l) executable test suite
- m) fail verdict
- n) idle testing state
- o) implementation under test
- p) inconclusive verdict
- q) invalid test event
- r) local test method

¹⁾ To be published.

- s) lower tester
- t) means of testing
- u) pass verdict
- v) PICS proforma
- w) PIXIT proforma
- x) protocol implementation conformance statement
- y) protocol implementation extra information for testing
- z) point of control and observation
- aa) remote test method
- ab) stable testing state
- ac) standardized abstract test suite
- ad) static conformance requirements
- ae) syntactically invalid test event
- af) system under test
- ag) test body
- ah) test case
- ai) test case error
- aj) test coordination procedures
- ak) test event
- al) test group
- am) test group objective
- an) test laboratory
- ao) test management protocol
- ap) test outcome
- aq) (test) postamble
- ar) (test) preamble
- as) test purpose
- at) test realization
- au) test realizer
- av) test step
- aw) test suite
- ax) test system
- ay) upper tester
- az) (test) verdict
- ba) testing state

3.2 Terms from ISO 7498

The following terms defined in ISO 7498 apply:

- a) application layer
- b) protocol data unit
- c) service access point
- d) session layer
- e) subnetwork

- f) transfer syntax
- g) transport layer

3.3 Terms from ISO/TR 8509

The following terms defined in ISO/TR 8509 apply:

- a) service-provider

3.4 Terms from ISO/IEC 8824

The following terms defined in ISO/IEC 8824 apply:

- a) bitstring type
- b) characterstring type
- c) enumerated type
- d) external type
- e) object identifier
- f) octetstring type
- g) real type
- h) selection type
- i) sequence type
- j) sequence-of type
- k) set type
- l) set-of type
- m) subtype

NOTE - Where there may be ambiguity with TTCN terms these terms are prefixed with the term ASN.1.

3.5 Terms from ISO/IEC 8825

The following term defined in ISO/IEC 8825 applies:

- encoding

3.6 TTCN specific terms

For the purposes of this part of ISO/IEC 9646 the following definitions apply:

3.6.1 attach construct: A TTCN statement which attaches a Test Step to a calling tree.

3.6.2 base constraint: Specifies a set of default values for each and every field in an ASP or PDU type definition.

3.6.3 base type: The type from which a type defined in a test suite is derived.

3.6.4 behaviour line: An entry in a dynamic behaviour table representing a test event or other TTCN statement together with associated label, verdict, constraints reference and comment information as applicable.

3.6.5 behaviour tree: A specification of a set of sequences of test events, and other TTCN statements.

3.6.6 blank entry: In a modified compact constraint table a blank entry in a constraint parameter or field denotes that a constraint value is to be inherited.

3.6.7 calling tree: The behaviour tree to which a Test Step is attached.

3.6.8 compact constraint table: Declaration of a set of constraints for an ASP, PDU or Structured Type arranged in a single table.

3.6.9 compact test case table: Declaration of a set of Test Cases for a given Test Group arranged in a single table.

3.6.10 constraints part: That part of a TTCN test suite concerned with the specification of the values of ASP parameters and PDU fields being sent to the IUT, and conditions on ASP parameters and PDU fields received from the IUT.

3.6.11 constraints reference: A reference to a constraint, given in a behaviour line.

3.6.12 declarations part: That part of a TTCN test suite concerned with the definition and/or declaration of all non-

predefined components that are used in the test suite.

3.6.13 default behaviour: The events, and other TTCN statements, which may occur at any level of the associated tree, and which are indicated in the Default behaviour proforma.

3.6.14 default group: A named set of default behaviours.

3.6.15 default group reference: A path specifying the logical location of a Default in the Default Library.

3.6.16 default identifier: A unique name for a Default.

3.6.17 default library: The set of the Default behaviours in a test suite.

3.6.18 default reference: A reference to a Default in the Default Library from a Test Case or Test Step table.

3.6.19 derivation path: An identifier, consisting of a base constraint identifier concatenated with one or more modified constraint identifiers, separated by dots and finishing with a dot.

3.6.20 dynamic chaining: The linking from constraint declarations of an ASP parameter or PDU field to the constraint declaration of another PDU by means of parameterization. Which PDUs are chained is specified in the constraints reference of a behaviour line.

3.6.21 dynamic part: That part of a TTCN test suite concerned with the specification of Test Case, Test Step and Default dynamic behaviour descriptions.

3.6.22 implicit send event: A mechanism used in Remote Test Methods for specifying that the IUT should be made to initiate a particular PDU or ASP.

3.6.23 level of indentation: Indicates the tree structure of a behaviour description. It is reflected in the behaviour description by indentation of text.

3.6.24 local tree: A behaviour tree defined in the same proforma as its calling tree.

3.6.25 modified constraint: A constraint defined for an ASP or a PDU that already has a base constraint, and which makes modifications on that base constraint.

3.6.26 operational semantics: Semantics explaining the execution of a TTCN behaviour tree.

3.6.27 otherwise event: The TTCN mechanism for dealing with unforeseen test events in a controlled way.

3.6.28 overview part: That part of a TTCN test suite concerned with presenting an overview of the structure of the test suite, the structure (if any) of the Test Step Library, the structure (if any) of the Default Library and the association of selection expressions (if any) with Test Cases and/or Test Groups. This part also provides indexes to Test Cases, Test Steps and Defaults.

3.6.29 preliminary result: A result recorded before the end of a test case indicating whether the associated part of the test case passed, failed or was inconclusive.

3.6.30 pseudo-event: A pseudo-event is a TTCN expression or Timer operation appearing on a statement line in the behaviour description without any associated event.

3.6.31 qualified event: An event that has an associated Boolean expression.

3.6.32 receive event: The receipt of an ASP or PDU at a named or implied PCO.

3.6.33 root tree: The main behaviour tree of a Test Case, occurring at the level of entry into the Test Case.

3.6.34 send event: The sending of an ASP or PDU to a named or implied PCO.

3.6.35 set of alternatives: TTCN statements coded at the same level of indentation and belonging to the same predecessor node. They represent the possible events, pseudo-events and constructs which are to be considered at the relevant point in the execution of the Test Case.

3.6.36 single constraint table: Declaration of a constraint for a single ASP or PDU of a given type arranged in a single table.

3.6.37 snapshot semantics: A semantic model to eliminate the effect of timing on the execution of a Test Case, defined in terms of snapshots of the test environment, during which the environment is effectively frozen for a prescribed period.

3.6.38 specific value: A value in TTCN which does not contain any matching mechanism or unbound variable.

3.6.39 static chaining: The linking from constraint declarations of an ASP parameter or PDU field to the constraint declaration of another PDU by explicitly referencing a constraint as its value.

3.6.40 static semantics: Semantic rules that restrict the usage of the TTCN syntax.

3.6.41 structured type: A collection of one or more ASP parameters or PDU fields which may exist in one or more ASP or PDU type definition which is defined in a separate declaration and which may be used to specify a portion of a flat structure or a substructure within the ASP or PDU.

3.6.42 test case identifier: A unique name for a Test Case.

3.6.43 test case variable: One of a set of variables declared globally to the test suite, but whose value is retained only for the execution of a single Test Case.

3.6.44 test group reference: A path specifying the logical location of a Test Case in the ATS structure.

3.6.45 test step group: A named set of test steps.

3.6.46 test step group reference: A path specifying the logical location of a Test Step in the Test Step Library.

3.6.47 test step identifier: A unique name for a Test Step.

3.6.48 test step library: The set of the Test Step dynamic behaviour descriptions in the test suite, that are not local Test Steps.

3.6.49 test step objective: An informal statement of what the Test Step is meant to accomplish.

3.6.50 test suite constant: One of a set of constants, *not* derived from the PICS or PIXIT, which will remain constant throughout the test suite.

3.6.51 test suite parameter: One of a set of constants derived from the PICS or PIXIT which globally parameterize a test suite.

3.6.52 test suite variable: One of a set of variables declared globally to the test suite, and which retain their values between Test Cases.

3.6.53 timeout event: An event which is used within a behaviour tree to check for expiration of a specified timer.

3.6.54 tree attachment: The method of indicating that a behaviour tree specified elsewhere (either at a different point in the current proforma, or as a Test Step in the Test Step Library) is to be included in the current behaviour tree.

3.6.55 tree header: An identifier for a local tree followed by an optional list of formal parameters for the tree.

3.6.56 tree identifier: A name identifying a local tree.

3.6.57 tree leaf: A TTCN statement in a behaviour tree or Test Step which has no specified subsequent behaviour.

3.6.58 tree node: A single TTCN statement.

3.6.59 tree notation: The notation used in TTCN to represent Test Cases as trees.

3.6.60 TTCN statement: An event, a pseudo-event or construct which is specified in a behaviour description.

3.6.61 unforeseen test event: A test event which has not been identified as a test event within a foreseen test outcome in the test suite. It is normally handled using the OTHERWISE event.

3.6.62 unqualified event: An event that does not have an associated Boolean expression.

4 Abbreviations

4.1 Abbreviations defined in ISO/IEC 9646-1.

For the purposes of this part of ISO/IEC 9646, the following abbreviations defined in ISO/IEC 9646-1:1991, clause 4 apply:

ATS : abstract test suite

ASP : abstract service primitive

ETS : executable test suite

IUT : implementation under test

LT : lower tester

MOT : means of testing

PCO : point of control and observation

PICS : protocol implementation conformance statement

PIXIT : protocol implementation extra information for testing

SUT : system under test

TMP : test management protocol
UT : upper tester

4.2 Abbreviations defined in ISO/IEC 9646-2

For the purposes of this part of ISO/IEC 9646, the following abbreviations defined in ISO/IEC 9646-2:1991, clause 4 apply:

DS : distributed single-layer (test method)
LS : local single-layer (test method)
RS : remote single-layer (test method)
TTCN : tree and tabular combined notation

4.3 Other abbreviations

For the purposes of this part of ISO/IEC 9646, the following abbreviations also apply:

ASN.1 : abstract syntax notation one
BNF : the extended Backus-Naur form used in TTCN
FDT : formal description technique
FIFO : first in first out
OSI : open systems interconnection
PDU : protocol data unit
SAP : service access point
TCP : test coordination procedures
TTCN.GR : tree and tabular combined notation, graphical form
TTCN.MP : tree and tabular combined notation, machine processable form

5 The syntax forms of TTCN

TTCN is provided in two forms

- a) a graphical form (TTCN.GR) suitable for human readability;
- b) a machine processable form (TTCN.MP) suitable for transmission of TTCN descriptions between machines and possibly suitable for other automated processing.

TTCN.GR is defined using tabular proformas. TTCN.MP is defined using syntax productions which have special TTCN.MP keywords as terminal symbols instead of the fixed parts of the tabular proformas (e.g., the box lines and headers). The entries within the TTCN.GR tables are defined by syntax productions which do not include any TTCN.MP keywords; these productions are common to both TTCN.GR and TTCN.MP.

The syntax productions of TTCN.MP are specified in annex A. As an aid to clarifying the TTCN.GR description, many of the syntax productions that are common to both TTCN.MP and TTCN.GR are embedded in the text of the body of this part of ISO/IEC 9646; these are marked: SYNTAX DEFINITION. To aid readability some productions will appear in several places in the text.

The syntax productions embedded within the text are intended to be identical copies of the corresponding productions from annex A, but if there is any conflict annex A shall take precedence.

The text description of TTCN.GR is intended to be consistent with the underlying syntax as defined in the TTCN.MP syntax productions, except for the differences identified in clause A.5 and the static semantic restrictions specified in annex A (which are common to both TTCN.MP and TTCN.GR).

If there is any conflict between the TTCN.GR syntax and static semantics as described by the text and as described by annex A, then

- a) except for the differences specified in clause A.5, the TTCN.MP syntax productions shall have precedence over the text and syntax productions in the body of this part of ISO/IEC 9646;
- b) the static semantics restrictions specified in clause A.4 and in the static semantics comments (marked STATIC SEMANTICS) on the syntax productions in clause A.3 specify restrictions on what is valid TTCN, restricting what is allowed according to the syntax productions;
- c) the static semantics restrictions specified in annex A shall have precedence over the text in the body of this part of ISO/IEC 9646.

If an ATS is specified in TTCN.GR in compliance with this part of ISO/IEC 9646, then there is a unique corresponding TTCN.MP representation of that ATS sharing the same underlying syntax. These two representations have identical operational semantics. Two different representations of an ATS are equivalent if and only if they have identical operational semantics.

NOTE - If there is a standardized ATS specified in TTCN.GR and an apparently equivalent TTCN.MP representation, but there is a conflict in interpretation of the operational semantics of the two, then the operational semantics of the TTCN.GR takes precedence, because it is the TTCN.GR version that is the standardized ATS.

6 Compliance

6.1 ATSs that comply with this part of ISO/IEC 9646 shall satisfy the requirements for either TTCN.GR or TTCN.MP.

NOTE - See ISO/IEC 9646-1:1991, clause 10, for an explanation of the use of the term "compliance" in ISO/IEC 9646.

6.2 ATSs that comply with the requirements of TTCN.GR shall satisfy the TTCN.GR syntax requirements stated in clauses 8 through 15 and clause A.4.

6.3 ATSs that comply with the requirements of TTCN.MP shall satisfy the TTCN.MP syntax requirements stated in clause A.3.

6.4 ATSs that comply with this part of ISO/IEC 9646 shall satisfy the static semantic requirements specified in clauses 7 through 15 and have operational semantics in accordance with the definition of the operational semantics in annex B, such that they are semantically valid.

6.5 A standardized ATS that complies with this part of ISO/IEC 9646 shall require that any realization of that test suite that claims to conform to that standardized ATS shall

- a) have operational semantics equivalent to the operational semantics of the test suite as defined by annex B;
- b) comply with ISO/IEC 9646-4.

NOTE - If, during execution of the executable test case that conforms to the TTCN specification of the corresponding abstract test case, a static semantic or operational semantic error is detected, then a test laboratory complying with ISO/IEC 9646-5 will record an abstract or executable test case error, depending on where the error is located.

6.6 Any standardized ATS that reached Draft International Standard status during or before 1991 or that is approved as a CCITT Recommendation during the 1989-1992 Study Period may be stated to comply with ISO/IEC 9646-3 but use some or all of the Draft International Standard (DIS) TTCN features which have changed between DIS and International Standard (IS), as outlined in annex F. Such a test suite shall reference ISO/IEC 9646-3 and contain a description of the differences between the features of TTCN that it uses and those specified in this IS specification of TTCN.

7 Conventions

7.1 Introduction

The following conventions have been used when defining the TTCN.GR table proformas and the TTCN.MP grammar.

7.2 Syntactic metanotation

Table 1 defines the metanotation used to specify the extended BNF grammar for TTCN (henceforth called BNF):

Table 1 - The TTCN.MP Syntactic Metanotation

::=	is defined to be
	alternative
[abc]	0 or 1 instances of abc
{abc}	0 or more instances of abc
{abc}+	1 or more instances of abc
(...)	textual grouping
abc	the non-terminal symbol abc
abc	a terminal symbol abc
"abc"	a terminal symbol abc

EXAMPLE 1 - Use of the BNF metanotation:

FormalParList ::= "(" FormalPar&Type {SemiColon FormalPar&Type} ")"

The following conventions will be used for text used in table proformas:

- a) Bold text (**like this**) shall appear verbatim in each actual table in a TTCN test suite;
- b) Text in italics (*like this*) shall not appear verbatim in a TTCN test suite. This font is used to indicate that actual text shall be substitute for the italicized symbol. Syntax requirements for the actual text can be found in the corresponding TTCN.MP BNF production.

EXAMPLE 2 - *SuiteIdentifier* corresponds to production 3 in annex A

7.3 TTCN.GR table proformas

7.3.1 Introduction

The TTCN.GR is defined using two types of tables:

- a) single TTCN object tables (see 7.3.2),
which are used to define, declare or describe a single TTCN object such as a PDU declaration or a Test Case dynamic behaviour;
- b) multiple TTCN object tables (see 7.3.3);
are used to define a number of TTCN object of the same type in a single table, such as simple type definitions or Test Case Variables.

7.3.2 Single TTCN object tables

The general lay-out of a table for a single TTCN object is shown below:

Title of Table		
Object Name : : : Comments : This entire last header entry is OPTIONAL		
Column Title	 Other Columns	Comments
		This last column is OPTIONAL
Detailed Comments: This footer is OPTIONAL		

Title

Header

Body

Footer

Figure 1 - Generalized layout of a single declaration table

The header of the table contains general information on the object defined in the table. The first item in the header, named *Object Name*, contains an identifier for the object. The last item, named *Comments* contains an informal description of the object. This item may be omitted.

The body of the table consists of one or more columns. Each column has a title. The rightmost column, titled *Comments*, contains informal descriptions of the components of the object specified in the body. It does not exist in all proformas. In proformas containing a comments column this column can be omitted.

The footer of the table contains one item, named *Detailed Comments*. This footer can be used for the same purposes as the comments column in the body of the table. The test suite specifier can use the detailed comments footer in combination with the comments column, instead of a comments column, or not at all, in which case the footer can be omitted.

7.3.3 Multiple TTCN object tables

The general lay-out of a table for multiple TTCN objects is shown below:

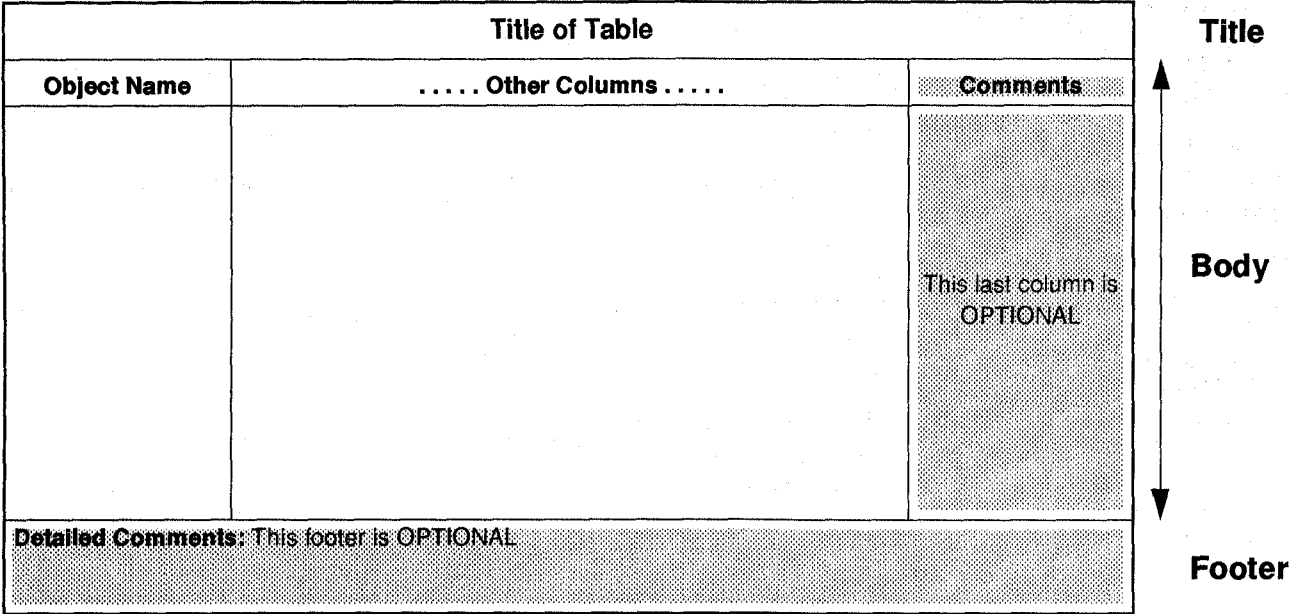


Figure 2 - Generalized layout of a multiple declaration table

This type of table has no header section. The body of the table consists of one or more columns. Each column has a title. The leftmost column, titled *Object Name*, contains identifiers of the objects defined or declared in the table. The rightmost column, titled *Comments*, contains informal descriptions of the objects defined or declared in the table. It does not exist in all proformas. When it exists its use is optional for the test suite specifier. The footer of the table is identical to the footer of the single table type.

7.3.4 Alternative compact tables

In some cases it is allowed to display a number of single TTCN object tables in an alternative space-saving compact format. That is, a number of single TTCN object tables may be displayed in a single compact table. The only tables that may be presented in this format are

- ASP constraints (tabular and ASN.1);
- PDU constraints (tabular and ASN.1);
- Structured Type constraints;
- ASN.1 Type constraints
- Test Case dynamic behaviours.

The formats of these alternative compact proformas are defined in annex C.

7.3.5 Specification of proformas

This part of ISO/IEC 9646 specifies 32 types of TTCN.GR tables and provides a graphic view of the corresponding proformas. These proformas conform to the generalised layout of 7.3.2 and 7.3.3. When a column is shaded in a proforma, this is a reminder that the column is optional.

7.4 Free Text and Bounded Free Text

Some table entries allow the use of free text, *i.e.*, characters from any of the character sets defined in ISO 10646. The following restrictions apply:

- a) Free Text shall not contain the combination of characters “*/”, unless preceded by backslash (\), as this is used in the TTCN.MP to indicate the end of a Free Text string. This means that double backslash (\\) means backslash.
- b) The combinations of characters “/*” and “*/” which open and close BoundedFreeText strings in the TTCN.MP shall not appear in the TTCN.GR, *i.e.*, wherever a Bounded FreeText string appears in a table section, as in a Full Identifier, these combinations of characters shall not be printed.

8 TTCN test suite structure

8.1 Introduction

TTCN allows a test suite to be hierarchically structured in accordance with ISO/IEC 9646-1:1991, 8.1. The components of this structure are

- a) Test Groups;
- b) Test Cases;
- c) Test Steps;

A TTCN test suite may be completely flat (*i.e.*, have no structure) in which case there are no Test Groups.

TTCN allows the use of Test Step Groups and Default Groups, similar to the concept of Test Groups, in order to structure Test Steps and Defaults hierarchically. This hierarchical structure is optional.

8.2 Test Group References

TTCN supports a naming structure that shows a conceptual grouping of Test Cases. Test Groups can be nested. Test Cases can also be stand alone (see ISO/IEC 9646-1:1991, clause 8, figure 9). The Test Group References define the structure of the test suite. Test Group References shall have the following syntax:

SYNTAX DEFINITION:

235 TestGroupReference ::= [SuiteIdentifier “/”] {TestGroupIdentifier “/”}

EXAMPLE 3 - A Transport group reference: TRANSPORT/CLASS0/CONN_ESTAB/

8.3 Test Step Group References

8.3.1 Test steps may be explicitly identified in TTCN and used to structure Test Cases and other Test Steps. Alternatively Test Steps may be implicit within the behaviour description of a Test Case. Explicit Test Steps may be specified either

- locally within a Test Case or Test Step behaviour description; or
- globally within a Test Step Library, which may be hierarchically structured into Test Step Groups.

NOTE - For example, a preamble may consist of just a few statement lines within a behaviour description of the Test Case, in which case it is implicit. Alternatively, a preamble may be explicitly specified with its own behaviour description. If such an explicit preamble is only of use within one Test Case, then it may be specified locally within that Test Case, but if it is of use in several Test Cases then it should be specified in the Test Step Library.

8.3.2 Local Test Steps are identified simply by a tree identifier. Global Test Steps are identified by a Test Step identifier. Global Test Steps also have a Test Step Group Reference, which shows the position of a Test Step in the Test Step Library. The structure of the Test Step Library is independent of the structure of the test suite. Test Step Group References shall have the following syntax:

SYNTAX DEFINITION:

248 TestStepGroupReference ::= [SuiteIdentifier “/”] {TestStepGroupIdentifier “/”}

EXAMPLE 4 - Transport Test Step Group Reference: TRANSPORT/STEP_LIBRARY/CLASS0/CONN_ESTAB/

8.4 Default Group References

Default behaviours (if any) are located in a Default Library.

A Default Group Reference specifies the location of the Default in the Default Library, which may be hierarchically structured. The Default Library has no influence on the test suite structure itself. Default Group References shall have the following syntax:

SYNTAX DEFINITION:

258 DefaultGroupReference ::= [SuiteIdentifier "/"] {DefaultGroupIdentifier "/" }

EXAMPLE 5 - Transport Default Group Reference: TRANSPORT/DEFAULT_LIBRARY/CLASS0/

8.5 Components of a TTCN test suite

An ATS written in TTCN shall have the following four sections in the order indicated:

- a) Suite Overview (see clause 9),
which contains the information needed for the general presentation and understanding of the test suite, such as test references and a description of its overall purpose;
- b) Declarations Part (see clause 10),
which contains the definitions or declarations of all the components that comprise the test suite (*e.g.*, PCOs, Timers, ASPs, PDUs, and their parameters or fields);
- c) Constraints Part (see clause 11, 12, 13),
which contains the declarations of values for the ASPs, PDUs, and their parameters used in the Dynamic Part. The constraints shall be specified using
 - 1) TTCN tables; or
 - 2) the ASN.1 value notation; or
 - 3) both TTCN tables and the ASN.1 value notation.
- d) Dynamic Part (see clause 14),
which comprises three sections that contain tables specifying test behaviour expressed mainly in terms of the occurrence of ASPs or PDUs at PCOs. These sections are
 - 1) the Test Case dynamic behaviour descriptions;
 - 2) a library containing Test Step dynamic behaviour descriptions (if any);
 - 3) a library containing Default dynamic behaviour descriptions (if any).

9 Test Suite Overview

9.1 Introduction

The purpose of the Test Suite Overview part of the ATS is to provide information needed for general presentation and understanding of the test suite. This includes:

- a) Test Suite Structure (see 9.2);
- b) Test Case Index (see 9.3);
- c) Test Step Index (see 9.4);
- d) Default Index (see 9.5).

9.2 Test Suite Structure

The Test Suite Structure contains identification of the pertinent reference documents, specification of the structure of the test suite, a brief description of its overall purpose, and references to the Test Group selection criteria.

The Test Suite Structure shall include at least the following information:

- a) the name of the test suite;
- b) references to the relevant base standards;
- c) a reference to the PICS proforma;
- d) a reference to the partial PIXIT proforma (see ISO/IEC 9646-2:1991, clause 15);
- e) an indication of the test method or methods to which the test suite applies, plus for the Coordinated Test Methods a reference to where the TMP is specified;
- f) other information which may aid understanding of the test suite, such as how it has been derived; this should be included as a comment;

g) a list of Test Groups in the test suite (if any),

where the following information shall be supplied for each group:

1) the Test Group Reference,

where the first identifier may be the suite name, and each successive identifier represents further conceptual ordering of the test suite. Test Groups shall be listed in the order that their corresponding Test Cases appear in the ATS. Furthermore, they shall be ordered such that every group within a single group immediately follows that group. All Test Groups in the test suite shall be listed;

2) an optional selection expression identifier,

which references an entry in the Test Case Selection Expression Definitions table used to determine if the Test Cases in the group apply to specific IUTs. This column may contain the identifier of a selection expression applicable to the Test Group. If a selection expression identifier is provided for a group, and the referenced selection expression evaluates to FALSE, then no Test Case in that group shall be selected for execution. If the selection expression evaluates to TRUE then Test Cases in that group shall be selected for execution depending on the evaluation of the selection expressions relevant to subgroups of that group and/or individual Test Cases. Omission of a selection expression identifier is equivalent to the Boolean value TRUE;

3) the Test Group Objective,

which is an informal statement of the objective of the Test Group;

4) a page number,

providing the location of the first Test Case of the group in the ATS. The page number listed with each Test Group Reference in the Test Suite Structure table shall be the page number of the first Test Case behaviour description in the group.

This information shall be provided in the format shown in the following proforma:

Test Suite Structure			
Suite Name : <i>SuiteIdentifier</i> Standards Ref : <i>FreeText</i> PICS Ref : <i>FreeText</i> PIXIT Ref : <i>FreeText</i> Test Method(s) : <i>FreeText</i> Comments : <i>[FreeText]</i>			
Test Group Reference	Selection Ref	Test Group Objective	Page Nr
. . . . <i>TestGroupReference</i>	 <i>[SelectExpr- Identifier]</i>	 <i>FreeText</i>	 <i>Number</i>
Detailed Comments: <i>[FreeText]</i>			

Proforma 1 - Test Suite Structure

SYNTAX DEFINITION:

- 3 SuiteIdentifier ::= Identifier
 235 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/" }
 83 SelectExprIdentifier ::= Identifier

9.3 Test Case Index

The Test Case Index contains a complete list of all Test Cases in the ATS. The following information shall be provided for each Test Case:

- a) an optional Test Group Reference (if the ATS is structured into Test Groups),
which defines where in the test suite group structure the Test Case resides. If the group reference for a Test Case is missing, then the Test Case is assumed to reside in the same Test Group as the previous Test Case in the index. Test Groups shall be listed in the order in which they exist in the ATS. An explicit Test Group Reference shall be provided for the first Test Case of each group. An explicit Test Group Reference shall also be provided for each Test Case that immediately follows the last Test Case of the Test Group; this is necessary if a Test Group contains both Test Groups and Test Cases;
- b) the Test Case name,
which shall be the identifier provided in the Test Case dynamic behaviour table. Test Cases shall be listed in the order in which they exist in the ATS;
- c) an optional selection expression identifier,
which references an entry in the Test Case Selection Expression Definitions table used to determine if the Test Case should be selected for execution. This column may contain the identifier of a selection expression applicable to the Test Case. If a selection expression identifier is provided, and the referenced selection expression evaluates to FALSE, then the Test Case shall not be selected for execution. If the selection expression evaluates to TRUE then the Test Case shall be selected for execution depending on the evaluation of the selection expressions for the Test Groups containing the Test Case. A Test Case is selected if the selection expression for the Test Case, and all groups containing the Test Case, evaluate to TRUE. Omission of a selection expression identifier is equivalent to the Boolean value TRUE;
- d) a description of the Test Case,
which is possibly a shortened form of the test purpose;
- e) a page number,
providing the location of the Test Case in the ATS. The page number listed with each Test Case Identifier in the Test Case Index table shall be the page number of the corresponding Test Case behaviour description.

This information shall be provided in the format shown in the following proforma:

Test Case Index				
Test Group Reference	Test Case Id	Selection Ref	Description	Page Nr
. [TestGroupReference]	 TestCase Identifier	 [SelectExpr- Identifier]	 FreeText	 Number
.				
Detailed Comments: FreeText				

Proforma 2 - Test Case Index

SYNTAX DEFINITION:

- 235 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}
- 233 TestCaseIdentifier ::= Identifier
- 83 SelectExprIdentifier ::= Identifier

9.4 Test Step Index

The Test Step Index contains a complete list of all Test Steps in the ATS. The following information shall be provided for each Test Step:

a) an optional Test Step Group Reference, (if the ATS is structured into Test Step Groups),

which defines where in the Test Step Library structure the Test Step resides. If the group reference for a Test Step is missing, then the Test Step is assumed to reside in the same group as the previous Test Step in the index. Test Step Groups shall be listed in the order in which they exist in the ATS. An explicit Test Step Group Reference shall be provided for the first Test Step of each group. An explicit Test Step Group Reference shall also be provided for each Test Step that immediately follows the last Test Step of the group; this is necessary if a Test Step Group contains both Test Step Groups and Test Steps;

b) the Test Step name,

which shall be the identifier provided in the Test Step dynamic behaviour table. Test Steps shall be listed in the order in which they exist in the ATS;

c) a description of the Test Step,

which is possibly a shortened form of the Test Step Objective;

d) a page number,

providing the location of the Test Step in the ATS. The page number listed with each Test Step Identifier in the Test Step Index table shall be the page number of the corresponding Test Step behaviour description;

This information shall be provided in the format shown in the following proforma:

Test Step Index			
Test Step Group Reference	Test StepId	Description	Page Nr
. . . [TestStepGroupReference]	. . . TestStep Identifier	. . . FreeText	. . . Number
Detailed Comments: [FreeText]			

Proforma 3 - Test Step Index

SYNTAX DEFINITION:

248 TestStepGroupReference ::= [SuiteIdentifier "/"] (TestStepGroupIdentifier "/")

246 TestStepIdentifier ::= Identifier

9.5 Default Index

The Default Index contains a complete list of all Defaults in the ATS. The following information shall be provided for each Default:

a) an optional Default Group Reference, (if the ATS is structured into Default Groups),

which defines where in the Default Library structure the Default resides. If the group reference for a Default is missing, then the Default is assumed to reside in the same group as the previous Default in the index. Defaults shall be listed in the order in which they exist in the ATS. An explicit Default Group Reference shall be provided for the first Default of each group. An explicit Default Group Reference shall also be provided for each Default that immediately follows the last Default of the group;

b) the Default name,

which shall be the identifier provided in the Default dynamic behaviour table. Defaults shall be listed in the order in which they exist in the ATS;

c) a description of the Default,
which is possibly a shortened form of the Default Objective;

d) a page number,
providing the location of the Default in the ATS. The page number listed with each Default Identifier in the Default Index table shall be the page number of the corresponding Default behaviour description.

This information shall be provided in the format shown in the following proforma:

Default Index			
Default Group Reference	DefaultId	Description	Page Nr
. . . [DefaultGroupReference]	. . . Default Identifier	. . . FreeText	. . . Number
Detailed Comments: [FreeText]			

Proforma 4 - Default Index

SYNTAX DEFINITION:

- 258 DefaultGroupReference ::= [SuitIdentifier "/"] {DefaultGroupIdentifier "/" }
- 257 DefaultIdentifier ::= Identifier

10 Declarations Part

10.1 Introduction

The purpose of the declarations part of the ATS is to define and declare all the components used in the test suite. The following components of an ATS referenced from the overview part, the constraints part and the dynamic part shall have been declared in the declarations part. These components are

a) definitions:

- 1) Test Suite Types (see 10.2.3);
- 2) Test suite operations (see 10.3.4);

b) parameterization and selection of Test Cases:

- 1) Test Suite Parameters (see 10.4);
- 2) Test Case Selection Expressions (see 10.5);

c) declarations/definitions:

- 1) Test Suite Constants (see 10.6);
- 2) Test Suite Variables (see 10.7.1);
- 3) Test Case Variables (see 10.7.3);
- 4) PCO s (see 10.8);
- 5) Timers (see 10.9);
- 6) ASP types (see 10.10);
- 7) PDU types (see 10.11);
- 8) Aliases (see 10.15).

10.2 TTCN types

10.2.1 Introduction

TTCN supports a number of predefined types and mechanisms that allow the definition of specific Test Suite Types. These types may be used throughout the test suite and may be referenced when Test Suite Parameters, Test Suite Constants, Test Suite Variables, ASP parameters, PDU fields *etc.* are declared.

10.2.2 Predefined TTCN types

A number of commonly used types are predefined for use in TTCN. All types defined in ASN.1 and in this clause may be referenced even though they do not appear in a type definition in a test suite. All other types used in a test suite shall be declared in the Test Suite Type definitions, ASP definitions or PDU definitions and referenced by name.

The following TTCN predefined types are considered to be the same as their counterparts in ASN.1:

a) **INTEGER predefined type:** a type with distinguished values which are the positive and negative whole numbers, including zero.

Values of type INTEGER shall be denoted by one or more digits; the first digit shall not be zero unless the value is 0; the value zero shall be represented by a single zero;

b) **BOOLEAN predefined type:** a type consisting of two distinguished values.

Values of the BOOLEAN type are TRUE and FALSE;

c) **BITSTRING predefined type:** a type whose distinguished values are the ordered sequences of zero, one, or more bits.

Values of type BITSTRING shall be denoted by an arbitrary number (possibly zero) of zeros and ones, preceded by a single ' and followed by the pair of characters 'B';

EXAMPLE 6 - '01101'B

d) **HEXSTRING predefined type:** a type whose distinguished values are the ordered sequences of zero, one, or more HEX digits, each corresponding to an ordered sequence of four bits.

Values of type HEXSTRING shall be denoted by an arbitrary number (possibly zero) of the HEX digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

preceded by a single ' and followed by the pair of characters 'H'; each HEX digit is used to denote the value of a semi-octet using a hexadecimal representation;

EXAMPLE 7 - 'AB01D'H

e) **OCTETSTRING predefined type**: a type whose distinguished values are the ordered sequences of zero or a positive even number of HEX digits (every pair of digits corresponding to an ordered sequence of eight bits).

Values of type OCTETSTRING shall be denoted by an arbitrary, but even, number (possibly zero) of the HEX digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

preceded by a single ' and followed by the pair of characters 'O'; each HEX digit is used to denote the value of a semi-octet using a hexadecimal representation;

EXAMPLE 8 - 'FF96'O

f) **CharacterString predefined types**: types whose distinguished values are zero, one, or more characters from some character set; the CharacterString types listed in table 2 may be used; they are defined in clause 31 of ISO/IEC 8824:1990.

Table 2 - Predefined CharacterString Types

<i>NumericString</i>
<i>PrintableString</i>
<i>TeletexString</i> (i.e., T61 String)
<i>VideotexString</i>
<i>VisibleString</i> (i.e., ISO646 String)
<i>IA5String</i>
<i>GraphicString</i>
<i>GeneralString</i>

Values of CharacterString types shall be denoted by an arbitrary number (possibly zero) of characters from the character set referenced by the CharacterString type, preceded and followed by double quote ("); if the CharacterString type includes the character double quote, this character shall be represented by a pair of double quote in the denotation of any value.

SYNTAX DEFINITION:

- 332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING | CharacterString
- 333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString | VisibleString | IA5String | GraphicString | GeneralString
- 338 Number ::= (NonZeroNum {Num}) | 0
- 339 NonZeroNum ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- 340 Num ::= 0 | NonZeroNum
- 341 BooleanValue ::= TRUE | FALSE
- 342 Bstring ::= "" {Bin | Wildcard} "" B
- 343 Bin ::= 0 | 1
- 344 Hstring ::= "" {Hex | Wildcard} "" H
- 345 Hex ::= Num | A | B | C | D | E | F
- 346 Ostring ::= "" {Oct | Wildcard} "" O
- 347 Oct ::= Hex Hex
- 348 Cstring ::= "" {Char | Wildcard | "\"} ""
- 349 Char ::= /* REFERENCE - A character defined by the relevant character string type */
- 350 Wildcard ::= AnyOne | AnyOrNone
- 351 AnyOne ::= "?"
- 352 AnyOrNone ::= ""

10.2.3 Test Suite Type Definitions

10.2.3.1 Introduction

Type definitions to be used as types for data objects and as subtypes for structured ASPs, PDUs *etc.* can be introduced using a tabular format and/or ASN.1. Wherever types are referenced within Test Suite Type definitions those references shall not be recursive (neither directly or indirectly).

10.2.3.2 Simple Type Definitions using tables

To define a new Simple Type, the following information shall be provided:

- a) a name for the type;
- b) the base type,

where the base type shall be a predefined type or a Simple Type. The base type is followed by the type restriction that shall take one of the following forms:

- 1) a list of distinguished values of the base type; these values comprise the new type;
- 2) a specification of a range of values of type INTEGER; the new type comprises the values including the lower boundary and the upper boundary specified in the range. In order to specify an infinite range, the keyword INFINITY may be used instead of a value indicating that there is no upper boundary or lower boundary;
- 3) a specification of a particular length or length range of a predefined or test suite string type; the length value(s) shall be interpreted according to table 4; only non-negative INTEGER literals or the keyword INFINITY for the upper bound shall be used.

This information shall be provided in the format shown in the following proforma:

Simple Type Definitions		
Type Name	Type Definition	Comments
SimpleTypeIdentifier	Type&Restriction	[FreeText]
Detailed Comments: [FreeText]		

Proforma 5 - Simple Type Definitions

SYNTAX DEFINITION:

- ```

27 SimpleTypeIdentifier ::= Identifier
29 Type&Restriction ::= Type [Restriction]
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING | CharacterString
333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString | VisibleString | IA5String |
GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
30 Restriction ::= LengthRestriction | IntegerRange | SimpleValueList
31 LengthRestriction ::= SingleTypeLength | RangeTypeLength
32 SingleTypeLength ::= "["Number "]"
33 RangeTypeLength ::= "[" LowerTypeBound To UpperTypebound "]"
34 IntegerRange ::= "(" LowerTypeBound To UpperTypebound ")"
35 LowerTypeBound ::= [Minus] Number | Minus INFINITY
36 UpperTypebound ::= [Minus] Number | INFINITY
37 To ::= TO | ".."
38 SimpleValueList ::= "(" [Minus] LiteralValue {Comma [Minus] LiteralValue} ")"

```

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. In the latter case, integer range shall be a subrange of the set of values defined by the base type.

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. Where a length restriction is used, the set of values for a type defined by this restriction shall be a true subset of the values defined by the base type.

EXAMPLE 9 - Simple Test Suite Type definitions

| Simple Type Definitions |                        |                                                                 |
|-------------------------|------------------------|-----------------------------------------------------------------|
| Type Name               | Type Definition        | Comments                                                        |
| Transport_classes       | INTEGER(0, 1, 2, 3, 4) | classes that may be used for transport layer connection         |
| String5                 | IA5String[5]           | string of length 5                                              |
| SeqNumbers              | INTEGER(0..127)        | all numbers from 0 to 127                                       |
| PositiveNumbers         | INTEGER(1..INFINITY)   | all positive INTEGER numbers                                    |
| String10to20            | IA5String [10 .. 20]   | string, min. length 10 characters and max. length 20 characters |

10.2.3.3 Structured Type Definitions using tables

Structured Types can be defined in the tabular form to be used for declaring structured objects as subtypes within ASP and PDU definitions and other Structured Types *etc.*

The following information shall be supplied for each Structured Type:

- a) its name,  
where appropriate the full name, as given in the relevant protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;
  - b) a list of the elements associated with the Structured Type,  
where the following information shall be supplied for each element:
    - 1) its name,  
where the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;
    - 2) its type and an optional attribute,  
where elements may be of a type of arbitrarily complex structure; there shall be no recursive references (neither directly nor indirectly);  
the optional element length restriction can be used in order to give the minimum and maximum length of an element of a string type (see 10.12).
- The elements of Structured Type definitions are considered to be optional, *i.e.*, in instances of these types whole elements may not be present.

This information shall be provided in the format shown in the following proforma:

| Structured Type Definition                                                           |                            |                   |
|--------------------------------------------------------------------------------------|----------------------------|-------------------|
| <b>Type Name :</b> <i>StructId&amp;FullId</i><br><b>Comments :</b> <i>[FreeText]</i> |                            |                   |
| Element Name                                                                         | Type Definition            | Comments          |
| <i>ElemId&amp;FullId</i>                                                             | <i>Type&amp;Attributes</i> | <i>[FreeText]</i> |
| <b>Detailed Comments:</b> <i>[FreeText]</i>                                          |                            |                   |

Proforma 6 - Structured Type Definition

#### SYNTAX DEFINITION:

```

42 StructId&FullId ::= StructIdentifier [FullIdentifier]
44 StructIdentifier ::= Identifier
48 ElemId&FullId ::= ElemIdentifier [FullIdentifier]
49 ElemIdentifier ::= Identifier
43 FullIdentifier ::= "(" BoundedFreeText ")"
154 Type&Attributes ::= (Type [LengthAttribute]) | PDU
331 Type ::= PredefinedType | ReferenceType
332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING | CharacterString
333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString | VisibleString | IA5String |
 GraphicString | GeneralString
334 ReferenceType ::= TS_TypeIdentifier | ASP_Identifier | PDU_Identifier
335 TS_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1_TypeIdentifier
155 LengthAttribute ::= SingleLength | RangeLength
156 SingleLength ::= "[" Bound "]"
157 Bound ::= Number | TS_ParIdentifier | TS_ConstIdentifier
158 RangeLength ::= "[" LowerBound To UpperBound "]"
159 LowerBound ::= Bound
37 To ::= TO | "."
160 UpperBound ::= Bound | INFINITY

```

#### 10.2.3.4 Test suite type definitions using ASN.1

Test Suite Types can be specified using ASN.1 This shall be achieved by an ASN.1 definition using the ASN.1 syntax as defined in ISO/IEC 8824. The following information shall be supplied for each ASN.1 type:

a) its name,

where appropriate the full name, as given in the relevant protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;

b) the ASN.1 type definition,

which shall follow the syntax defined in ISO/IEC 8824. For identifiers within that definition the dash symbol ( - ) shall not be used. The underscore symbol ( \_ ) may be used instead. The type identifier in the table header is the name of the first type defined in the table body.

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 ASN.1 comments can be used within the table body. The comments column shall not be present in this table.

This information shall be provided in the following proforma:

|                                                         |
|---------------------------------------------------------|
| ASN.1 Type Definition                                   |
| Type Name : ASN1_TypeId&FullId<br>Comments : [FreeText] |
| Type Definition                                         |
| ASN1_Type&LocalTypes                                    |
| Detailed Comments: [FreeText]                           |

Proforma 7 - ASN.1 Type Definition

SYNTAX DEFINITION:

- 54 ASN1\_TypeId&FullId ::= ASN1\_TypeIdentifier [FullIdentifier]
- 55 ASN1\_TypeIdentifier ::= Identifier
- 43 FullIdentifier ::= "(" BoundedFreeText ")"
- 57 ASN1\_Type&LocalTypes ::= ASN1\_Type {ASN1\_LocalType}
- 58 ASN1\_Type ::= Type  
/\* REFERENCE -Where Type is a non-terminal defined in ISO/IEC 8824 \*/
- 59 ASN1\_LocalType ::= Typeassignment  
/\* REFERENCE -Where Typeassignment is a non-terminal defined in ISO/IEC 8824 \*/

EXAMPLE 10 - An ASN.1 Test Suite Type definition:

|                                                                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ASN.1 Type Definition                                                                                                                                                                                                              |
| Type Name : DATE_type<br>Comments : to illustrate the structure of ASN.1 type definitions                                                                                                                                          |
| Type Definition                                                                                                                                                                                                                    |
| SEQUENCE {<br>day DAY_type,<br>month MONTH_type,<br>year YEAR_type<br>}<br><br>-- local DAY_type<br>DAY_type::= INTEGER {first(1), last(31)}<br><br>-- MONTH_type and YEAR_type are defined in other ASN.1 Type Definitions tables |

10.2.3.5 ASN.1 Type Definitions by Reference

Types can be specified by a precise reference to an ASN.1 type defined in an OSI standard or by referencing an ASN.1 type defined in an ASN.1 module attached to the test suite. The following information shall be supplied for each type:

- a) its name,  
where this name may be used throughout the entire test suite. This name shall be specified without a FullIdentifier;
- b) the type reference,  
which shall follow the identifier rules stated in ISO/IEC 8824;

c) the module identifier,

which consists of a module reference that shall follow the identifier rules stated in ISO/IEC 8824, and an optional ObjectIdentifier; the module shall be unique within the domain of interest.

This information shall be provided in the following proforma:

| ASN.1 Type Definitions By Reference |                |                   |            |
|-------------------------------------|----------------|-------------------|------------|
| Type Name                           | Type Reference | Module Identifier | Comments   |
| ASN1_TypeId&FullId                  | TypeReference  | ModuleIdentifier  | [FreeText] |
| Detailed Comments: [FreeText]       |                |                   |            |

### Proforma 8 - ASN.1 Type Definitions By Reference

#### SYNTAX DEFINITION:

```

54 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
55 ASN1_TypeIdentifier ::= Identifier
43 FullIdentifier ::= "(" BoundedFreeText ")"
63 TypeReference ::= typereference
/* REFERENCE -Where typereference is defined in ISO/IEC 8824:1990, subclause 8.2 */
65 ModuleIdentifier ::= ModuleIdentifier
/* REFERENCE -Where ModuleIdentifier is a non-terminal defined in ISO/IEC 8824 */

```

Since the ASN.1 types imported from ASN.1 modules can contain identifiers, type references and value references that follow the identifier rules in ISO/IEC 8824, they can contain hyphens. To be able to use the imported definitions in TTCN it is necessary to change the hyphens in imported identifiers to underscore. This is done in the import process.

**EXAMPLE 11** - The following type definition in an ASN.1 module:

module-1 DEFINITIONS BEGIN

```

Type-1 ::= SEQUENCE {
 field1 Sub-Type-1,
 field2 BIT STRING {first-bit(0), second-bit(1)} }

```

END

can be imported to TTCN with:

| ASN.1 Type Definitions By Reference |                      |                      |          |
|-------------------------------------|----------------------|----------------------|----------|
| Type Name                           | Type Reference       | Module Identifier    | Comments |
| Type_1<br>Sub_Type_1                | Type-1<br>Sub-Type-1 | module-1<br>module-1 |          |

The above reference definition of Type-1 is equivalent to the following definition:

| ASN.1 Type Definition                                                              |  |
|------------------------------------------------------------------------------------|--|
| Type Name : Type_1                                                                 |  |
| Comments :                                                                         |  |
| Type Definition                                                                    |  |
| SEQUENCE { field1 Sub_Type_1,<br>field2 BIT STRING {first_bit(0), second_bit(1)} } |  |

10.3 TTCN operators and TTCN operations

10.3.1 Introduction

TTCN supports a number of predefined operators, operations and mechanisms that allow the definition of Test Suite Operations. These operators and operations may be used throughout any dynamic behaviour descriptions and constraints.

10.3.2 TTCN operators

10.3.2.1 Introduction

The predefined operators fall into three categories:

- a) arithmetic;
- b) relational;
- c) Boolean.

The precedence of these operators is shown in table 3. Parentheses may be used to group operands in expressions, a parenthesized expression has the highest precedence for evaluation.

Table 3 - Precedence of Operators

|         |        |                |
|---------|--------|----------------|
| highest |        | ( )            |
| ↓       | Unary  | + - NOT        |
|         |        | * / MOD AND    |
| ↓       | Binary | + - OR         |
| lowest  |        | = < > <> >= <= |

Within any row in table 3, the listed operators have equal precedence. If more than one operator of equal precedence appear in an expression, the operations are evaluated left to right.

SYNTAX DEFINITION:

```
321 AddOp ::= "+" | "-" | OR
322 MultiplyOp ::= "*" | "/" | MOD | AND
323 UnaryOp ::= "+" | "-" | NOT
324 RelOp ::= "=" | "<" | ">" | "<>" | ">=" | "<="
```

10.3.2.2 Predefined arithmetic operators

The predefined arithmetic operators are:

**"+", "-", "\*", "/", MOD**

They represent the operations of addition, subtraction, multiplication, division and modulo. Operands of these operators shall be of type INTEGER (i.e., TTCN or ASN.1 predefined) or derivations of INTEGER (i.e., subrange). ASN.1 Named Values shall not be used within arithmetic expressions as operands of operations.

The result type of arithmetic operations is INTEGER.

In the case where plus ( + ) or minus ( - ) is used as the unary operator the rules for operands apply as well. The result of using the minus operator is the negative value of the operand if it was positive and vice versa.

The result of performing the division operation ( / ) on two INTEGER values gives the whole INTEGER value resulting from dividing the first INTEGER by the second (i.e., fractions are discarded).

The result of performing the MOD operation on two INTEGER values gives the remainder of dividing the first INTEGER by the second.

10.3.2.3 Predefined relational operators

The predefined relational operators are:

**"=" | "<" | ">" | "<>" | ">=" | "<="**

They represent the relations of equality, less than, greater than, not equal to, greater than or equal to and less than or equal to. Operands of equality ( = ) and not equal to ( <> ) may be of an arbitrary type. The two operands shall be

compatible. All other relational operators shall have operands only of type INTEGER or derivatives of INTEGER. The result type of these operations is BOOLEAN.

In string comparisons BITSTRING, HEXSTRING, OCTETSTRING and all kinds of CharacterStrings may contain the wildcard characters AnyOrNone ( \* ) and AnyOne ( ? ). In this case the comparison is performed according to the pattern matching rules defined in 11.6.5.

#### 10.3.2.4 Predefined Boolean operators

The predefined Boolean operators are

#### NOT AND OR

They represent the operations of negation, logical AND and logical OR. Their operands shall be of type BOOLEAN (TTCN or ASN.1 or predefined). The result type of the Boolean operators is BOOLEAN.

The logical AND returns the value TRUE if both its operands are TRUE; otherwise it returns the value FALSE. The logical OR returns the value TRUE if at least one of its operands is TRUE; it returns the value FALSE only if both operands are FALSE. The logical NOT is the unary operator that returns the value TRUE if its operand was of value FALSE and returns the value FALSE if the operand was of value TRUE.

### 10.3.3 Predefined operations

#### 10.3.3.1 Introduction

The predefined operations fall into two categories:

- a) conversion;
- b) others

Predefined operations may be used in every test suite. They do not require an explicit definition using a Test Suite Operation Definition table. When a predefined operation is invoked

- a) the number of the actual parameters shall be the same as the number of the formal parameters; and
- b) each actual parameter shall evaluate to an element of its corresponding formal parameter's type; and
- c) all variables appearing in the parameter list shall be bound.

Each of the predefined operations is presented in the following format:

OPERATION\_NAME (FORMAL\_PARAMETER\_LIST)  $\Rightarrow$  RESULT\_TYPE

#### 10.3.3.2 Predefined conversion operations

10.3.3.2.1 TTCN supports the following predefined operations for type conversions:

- a) HEX\_TO\_INT converts HEXSTRING to INTEGER;
- b) BIT\_TO\_INT converts BITSTRING to INTEGER;
- c) INT\_TO\_HEX converts INTEGER to HEXSTRING;
- d) INT\_TO\_BIT converts INTEGER to BITSTRING.

These operations provide encoding rules within the context of the operations only. It is invalid to assume these encoding rules apply outside the domain of the operations in TTCN.

##### 10.3.3.2.2 HEX\_TO\_INT(hexvalue:HEXSTRING) $\Rightarrow$ INTEGER

This operation converts a single HEXSTRING value to a single INTEGER value.

For the purposes of this conversion, a HEXSTRING shall be interpreted as a positive base 16 INTEGER value. The rightmost HEX digit is least significant, the leftmost HEX digit is the most significant. The HEX digits 0 ... F represent the decimal values 0 ... 15 respectively.

##### 10.3.3.2.3 BIT\_TO\_INT(bitvalue:BITSTRING) $\Rightarrow$ INTEGER

This operation converts a single BITSTRING value to a single INTEGER value.

For the purposes of this conversion, a BITSTRING shall be interpreted as a positive base 2 INTEGER value. The rightmost BIT is least significant, the leftmost BIT is the most significant. The bits 0 and 1 represent the decimal values 0 and 1 respectively.

##### 10.3.3.2.4 INT\_TO\_HEX(intvalue, length:INTEGER) $\Rightarrow$ HEXSTRING

This operation converts a single INTEGER value to a single HEXSTRING value. The resulting string is *length* HEX digits long.

For the purposes of this conversion, a HEXSTRING shall be interpreted as a positive base 16 INTEGER value. The rightmost HEX digit is least significant, the leftmost HEX digit is the most significant. The HEX digits 0 ... F represent the decimal values 0 ... 15 respectively.

If the conversion yields a value with fewer HEX digits than specified in the second parameter, then the HEXSTRING shall be padded on the left with zeros.

A test case error shall occur if the *intvalue* is negative or if the resulting HEXSTRING contains more HEX digits than specified in the second parameter.

#### 10.3.3.2.5 INT\_TO\_BIT(intvalue, length:INTEGER) ⇒ BITSTRING

This operation converts a single INTEGER value to a single BITSTRING value. The resulting string is *length* bits long.

For the purposes of this conversion, a BITSTRING shall be interpreted as a positive base 2 INTEGER value. The rightmost BIT is least significant, the leftmost BIT is the most significant. The bits 0 and 1 represent the decimal values 0 and 1 respectively.

If the conversion yields a value with fewer bits than specified in the second parameter, then the BITSTRING shall be padded on the left with zeros.

A test case error shall occur if the *intvalue* is negative or if the resulting BITSTRING contains more bits than specified in the second parameter.

#### 10.3.3.3 Other predefined operations

TTCN also defines the following predefined operations:

- a) IS\_PRESENT;
- b) NUMBER\_OF\_ELEMENTS;
- c) IS\_CHOSEN;
- d) LENGTH\_OF;

##### 10.3.3.3.1 IS\_PRESENT(DataObjectReference) ⇒ BOOLEAN

As an argument the operation shall take a reference to a field within a data object only if it is defined as being OPTIONAL or if it has a DEFAULT value. The field may be of any type. The result of applying the operation is the BOOLEAN value TRUE if and only if the value of the field is present in the actual instance of the data object. Otherwise the result is FALSE.

The argument of the operation shall have the format as defined in 10.3.4.

##### EXAMPLE 12 - Use of IS\_PRESENT:

```
if received_PDU is of ASN.1 type
 SEQUENCE {
 field_1 INTEGER OPTIONAL,
 field_2 SEQUENCE OF INTEGER }

then, the operation call
 IS_PRESENT(received_PDU.field_1)
evaluates to TRUE if field_1 in the actual instance of received_PDU is present.
```

##### 10.3.3.3.2 NUMBER\_OF\_ELEMENTS(DataObjectReference) ⇒ INTEGER

The operation returns the actual number of elements of a data object that is of type ASN.1 SEQUENCE OF or SET OF. The operation shall not be applied to data objects or fields of data objects other than of ASN.1 type SEQUENCE OF or SET OF. The argument of the operation shall have the format as defined in 10.3.4.

##### EXAMPLE 13 - Use of NUMBER\_OF\_ELEMENTS:

```
if received_PDU is of ASN.1 type
 SEQUENCE {
 field_1 INTEGER OPTIONAL,
 field_2 SEQUENCE OF INTEGER }

then, the operation call
 NUMBER_OF_ELEMENTS(received_PDU.field_2)
returns the number of elements of the SEQUENCE OF INTEGER within the actual data object received_PDU.
```

**10.3.3.3.3 IS\_CHOSEN(DataObjectReference) ⇒ BOOLEAN**

The operation returns the BOOLEAN value TRUE if and only if the data object reference specifies the variant of the CHOICE type that is actually selected for a given data object. Otherwise the result is FALSE. The operation shall not be applied to data objects or fields of data objects other than the of ASN.1 type CHOICE. The argument of the operation shall have the format as defined in 10.3.4.

**EXAMPLE 14 - Use of IS\_CHOSEN:**

if received\_PDU is of ASN.1 type

```
CHOICE { p1 PDU_type1,
 p2 PDU_type2,
 p3 PDU_type }
```

then, the operation call

IS\_CHOSEN(received\_PDU.p2)

returns TRUE if the actual instance of received\_PDU carries a PDU of the type PDU\_type2.

**10.3.3.3.4 LENGTH\_OF(DataObjectReference) ⇒ INTEGER**

The operation returns the actual length of a data object that is of type BITSTRING, HEXSTRING, OCTETSTRING or CharacterString. The units of length for each string type are defined in table 4. The argument of the operation shall have the format as defined in 10.3.4.

The operation shall not be applied to data objects or fields of data objects other than of BITSTRING, HEXSTRING, OCTETSTRING or CharacterString.

**EXAMPLE 15 - Use of LENGTH\_OF**

If S='010'B then LENGTH\_OF(S) returns 3

If S='F3'H then LENGTH\_OF(S) returns 2

If S='F2'O then LENGTH\_OF(S) returns 1

If S="EXAMPLE" then LENGTH\_OF(S) returns 7

**10.3.4 Test Suite Operation Definitions**

Operations specific to a test suite may be introduced by the ATS specifier. To define a new operation, the following information shall be provided:

a) a name for the operation;

b) a list of the input parameters and their types,

this is a list of the formal parameter names and types. Each parameter name shall be followed by a colon and then the name of the parameter's type.

When more than one parameter of the same type is used, the parameters may be specified as a parameter sub-list. When a parameter sub-list is used, the parameter names shall be separated from each other by a comma. The final parameter in the list shall be followed by a colon and then the name of the type of the parameter.

When more than one parameter and type pair (or parameter list and type pair) is used, the pairs shall be separated from each other by semicolons.

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.

**EXAMPLE 16 - Parameter lists**

The following are equivalent methods of specifying a parameter list using two INTEGER parameters and one BOOLEAN parameter:

(A:INTEGER; B:INTEGER; C:BOOLEAN)

(A, B:INTEGER; C:BOOLEAN)

c) the type of the result,

which shall follow the rules for the parameter types in b);

d) a description of the operation,

which shall consist of an explanation of the operation, plus at least one example showing an invocation and corresponding result; the explanation shall begin by stating the operation name, followed by a parenthesized list containing the parameter names of the operation; this provides a "pattern" invocation for the operation.

This information shall be provided in the format shown in the following proforma:

| Test Suite Operation Definition             |                              |
|---------------------------------------------|------------------------------|
| <b>Operation Name</b>                       | : <i>TS_OpId&amp;ParList</i> |
| <b>Result Type</b>                          | : <i>Type</i>                |
| <b>Comments</b>                             | : <i>[FreeText]</i>          |
| Description                                 |                              |
| <i>FreeText</i>                             |                              |
| <b>Detailed Comments:</b> <i>[FreeText]</i> |                              |

Proforma 9 - Test Suite Operation Definition

#### SYNTAX DEFINITION:

- 69 TS\_OpId&ParList ::= TS\_OpIdentifier [FormalParList]  
 331 Type ::= PredefinedType | ReferenceType  
 332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString  
 333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**  
 334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier  
 335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier

An operation may be compared to a function in an ordinary programming language. However, the parameters to the operation shall not be altered as a result of any call of the operation and there shall be no side effects (*e.g.*, no changes to any Test Suite or Test Case Variable).

When a Test Suite Operation is invoked

- a) the number of the actual parameters shall be the same as the number of the formal parameters; and
- b) each actual parameter shall evaluate to an element of its corresponding formal parameter's type; and
- c) all variables appearing in the parameter list shall be bound; and

#### EXAMPLE 17 - Definition of the operation SUBSTR:

| Test Suite Operation Definition                                                                                                                                                                                                                              |                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| <b>Operation Name</b>                                                                                                                                                                                                                                        | : SUBSTR (source:IA5String; start_index, length:INTEGER) |
| <b>Result Type</b>                                                                                                                                                                                                                                           | : IA5String                                              |
| <b>Comments</b>                                                                                                                                                                                                                                              | :                                                        |
| Description                                                                                                                                                                                                                                                  |                                                          |
| <p><i>SUBSTR(source, start_index, length)</i> is the string of length <i>length</i> starting from index <i>start_index</i> of the source string <i>source</i>.</p> <p>For example: SUBSTR("abcde",3,2) = "cd"<br/>           SUBSTR("abcde",1,3) = "abc"</p> |                                                          |

## 10.4 Test Suite Parameter Declarations

The purpose of this part of the ATS is to declare constants derived from the PICS and/or PIXIT which are used to globally parameterize the test suite. These constants are referred to as Test Suite Parameters, and are used as a basis for Test Case selection and parameterization of Test Cases.

The following information relating to each Test Suite Parameter shall be provided:

a) its name;

b) its type,

where the type shall be a predefined type, an ASN.1 type, a Test Suite Type or a PDU type;

c) PICS/PIXIT entry reference,

which is a reference to an individual PICS/PIXIT proforma entry that will clearly identify where the value to be used for this Test Suite Parameter will be found.

This information shall be provided in the format shown in the following proforma:

| Test Suite Parameter Declarations |      |                |            |
|-----------------------------------|------|----------------|------------|
| Parameter Name                    | Type | PICS/PIXIT Ref | Comments   |
| TS_ParIdentifier                  | Type | FreeText       | [FreeText] |
| Detailed Comments: [FreeText]     |      |                |            |

Proforma 10 - Test Suite Parameter Declarations

### SYNTAX DEFINITION:

77 TS\_ParIdentifier ::= Identifier

331 Type ::= PredefinedType | ReferenceType

332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString

333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**

334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier

335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier

### EXAMPLE 18 - Declaration of Test Suite Parameters:

| Test Suite Parameter Declarations |         |                   |          |
|-----------------------------------|---------|-------------------|----------|
| Parameter Name                    | Type    | PICS/PIXIT Ref    | Comments |
| PAR1                              | INTEGER | PICS question xx  |          |
| PAR2                              | INTEGER | PICS question yy  |          |
| PAR3                              | INTEGER | PIXIT question zz |          |

## 10.5 Test Case Selection Expression Definitions

The purpose of this part of the ATS is to define selection expressions to be used in the Test Case selection process. This part of the ATS shall meet the requirements of ISO/IEC 9646-2.

A selection expression is associated with one or more Test Groups and/or Test Cases by placing its identifier in the Test Case Selection Reference column of the Test Suite Structure and/or Test Case Index. An expression may be referenced by more than one Test Group and/or Test Case.

Use of a selection expression shall be taken to mean that the Test Case is to be run if the selection expression evaluates to TRUE.

The following information relating to each Test Case Selection Expression shall be provided:

- a) its name;
- b) a selection expression,  
which shall evaluate to a BOOLEAN value, and which shall use only literal values, Test Suite Parameters, Test Suite Constants and other selection expression identifiers in its terms;

This information shall be provided in the format shown in the following proforma:

| Test Case Selection Expression Definitions |                      |            |
|--------------------------------------------|----------------------|------------|
| Expression Name                            | Selection Expression | Comments   |
| SelectExprIdentifier                       | SelectionExpression  | [FreeText] |
| Detailed Comments: [FreeText]              |                      |            |

Proforma 11 - Test Case Selection Expression Definitions

SYNTAX DEFINITION:

- 83 SelectExprIdentifier ::= Identifier
- 85 SelectionExpression ::= Expression

10.6 Test Suite Constant Declarations

The purpose of this part of the ATS is to declare a set of names for values *not* derived from the PICS or PIXIT that will be constant throughout the test suite.

The following information relating to each Test Suite Constant shall be provided:

- a) its name;
- b) its type,  
where the type shall be a predefined type, an ASN.1 type, a Test Suite Type or a PDU type;
- c) its value,  
where 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.

This information shall be provided in the format shown in the following proforma:

| Test Suite Constant Declarations |      |                  |            |
|----------------------------------|------|------------------|------------|
| Constant Name                    | Type | Value            | Comments   |
| TS_ConstIdentifier               | Type | DeclarationValue | [FreeText] |
| Detailed Comments: [FreeText]    |      |                  |            |

Proforma 12 - Test Suite Constant Declarations

SYNTAX DEFINITION:

- 90 TS\_ConstIdentifier ::= Identifier
- 331 Type ::= PredefinedType | ReferenceType

332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString  
 333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**  
 334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier  
 335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier  
 93 DeclarationValue ::= Expression

**EXAMPLE 19 - Declaration of Test Suite Constants**

| Test Suite Constant Declarations |           |            |          |
|----------------------------------|-----------|------------|----------|
| Constant Name                    | Type      | Value      | Comments |
| TS_CONST1                        | BOOLEAN   | TRUE       |          |
| TS_CONST2                        | IA5String | "A string" |          |

**10.7 TTCN variables****10.7.1 Test Suite Variable Declarations**

A test suite may make use of a set of variables which are defined globally for the test suite, and retain their values throughout the test suite. These variables are referred to as Test Suite Variables.

A Test Suite Variable is used whenever it is necessary to pass information from one Test Case to another.

The following information shall be provided for each variable declaration:

a) its name;

b) its type,

where the type shall be a predefined type, an ASN.1 type, a Test Suite Type or a PDU type;

c) its initial value (if any),

where the initial value column is used when it is desired to assign an initial value to a Test Suite Variable at its point of declaration; 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. Specifying an initial value is optional.

This information shall be provided in the format shown in the following proforma:

| Test Suite Variable Declarations |      |                    |            |
|----------------------------------|------|--------------------|------------|
| Variable Name                    | Type | Value              | Comments   |
| TS_VarIdentifier                 | Type | [DeclarationValue] | [FreeText] |
| Detailed Comments: [FreeText]    |      |                    |            |

**Proforma 13 - Test Suite Variable Declarations****SYNTAX DEFINITION:**

97 TS\_VarIdentifier ::= Identifier  
 331 Type ::= PredefinedType | ReferenceType  
 332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString  
 333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**  
 334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier  
 335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier  
 93 DeclarationValue ::= Expression

Since it is possible that any particular Test Case may be run independently of the others in the test suite, it is necessary that the use made of Test Suite Variables does not make assumptions about the ordering of the Test Case execution.

EXAMPLE 20 - Declaration of Test Suite Variables:

| Test Suite Variable Declarations |           |        |                                                                                                                              |
|----------------------------------|-----------|--------|------------------------------------------------------------------------------------------------------------------------------|
| Variable Name                    | Type      | Value  | Comments                                                                                                                     |
| state                            | IA5String | "idle" | Used to indicate the final stable state of the previous Test Case, if any, in order to help determine which preamble to use. |

10.7.2 Binding of Test Suite Variables

Initially Test Suite Variables are unbound. They may become bound (or be re-bound) in the following contexts:

- a) at the point of declaration if an initial value is specified;
- b) when the Test Suite Variable appears on the left-hand side of an assignment statement (see 14.10.4);

Once a Test Suite Variable has been bound to a value, the Test Suite Variable will retain that value until either it is bound to a different value, or execution of the test suite terminates - whichever occurs first.

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

10.7.3 Test Case Variable Declarations

A test suite may make use of a set of variables which are declared globally to the test suite but whose scope is defined to be local to the Test Case. These variables are referred to as Test Case Variables.

The following information shall be provided for each variable declaration:

- a) its name;
  - b) its type,
- where the type shall be a predefined type, an ASN.1 type, a Test Suite Type or a PDU type;
- c) its initial value (if any),

where the initial value column is used when it is desired to assign an initial value to a Test Case Variable at its point of declaration; 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. Specifying an initial value is optional.

This information shall be provided in the format shown in the following proforma:

| Test Case Variable Declarations      |             |                           |                   |
|--------------------------------------|-------------|---------------------------|-------------------|
| Variable Name                        | Type        | Value                     | Comments          |
| <i>TC_VarIdentifier</i>              | <i>Type</i> | <i>[DeclarationValue]</i> | <i>[FreeText]</i> |
| Detailed Comments: <i>[FreeText]</i> |             |                           |                   |

Proforma 14 - Test Case Variable Declarations

NOTE - Caution must be exercised when using Test Case Variables as local variables within a Test Step, in order to avoid usage conflicts with other Test Steps or Test Case Variables. A test suite specifier may avoid such problems by adopting a naming convention which will result in all such variables being uniquely named within a test suite.

**SYNTAX DEFINITION:**

103 TC\_VarIdentifier ::= Identifier  
 331 Type ::= PredefinedType | ReferenceType  
 332 PredefinedType ::= **INTEGER** | **BOOLEAN** | **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString  
 333 CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**  
 334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier  
 335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier  
 93 DeclarationValue ::= Expression

**10.7.4 Binding of Test Case Variables**

Initially Test Case Variables are unbound. They may become bound (or be re-bound) in the following contexts:

- a) at the point of declaration if an initial value is specified;
- b) when the Test Case Variable appears on the left-hand side of an assignment statement (see 14.10.4);

Once a Test Case Variable has been bound to a value, the Test Case Variable will retain that value until either it is bound to a different value, or execution of the Test Case terminates - whichever occurs first. At termination of the Test Case, the Test Case Variable becomes re-bound to its initial value, if one is specified, otherwise it becomes unbound.

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

**10.8 PCO Declarations**

This part of the ATS lists the set of points of control and observation (PCOs) to be used in the test suite and explains where in the testing environment these PCOs exist.

The number of PCOs shall be as defined in ISO/IEC 9646-1: 1991, 7.5, and ISO/IEC 9646-2: 1991, 12.6, for the test method(s) identified in the Test Suite Structure table.

TTCN behaviour statements specified for execution at the UT PCO shall not place requirements beyond those specified by ISO/IEC 9646-2.

In TTCN the PCO model is based on two First In First Out (FIFO) queues:

- one output queue for sending ASPs and/or PDUs
- one input queue for receiving ASPs and/or PDUs

The output queue is assumed to be located within the underlying service-provider or in the case of the UT, within the IUT.

A SEND event is successful by being passed from the LT to the service-provider, or by being passed from the UT to the IUT.

For the purpose of receiving events the tester has an input queue. All incoming events are queued and processed by the tester in the same order they were received, and without loss of any events.

NOTE - The queue model is only an abstract model and is not intended to imply a specific implementation.

The following information shall be provided for each PCO used in the test suite:

- a) its name,  
which is used in the behaviour descriptions to specify where particular events occur;
- b) its type,  
which is used to identify the service boundary where the PCO is located;
- c) its role,  
which is an explanation of which type of tester is placed at the PCO. The predefined identifier **UT** indicates that the PCO is an upper tester PCO and **LT** specifies a lower tester PCO.

This information shall be provided in the format shown in the following proforma:

| PCO Declarations              |                    |          |            |
|-------------------------------|--------------------|----------|------------|
| PCO Name                      | PCO Type           | Role     | Comments   |
| PCO_Identifier                | PCO_TypeIdentifier | PCO_Role | [FreeText] |
| Detailed Comments: [FreeText] |                    |          |            |

Proforma 15 - PCO Declarations

SYNTAX DEFINITION:

- 109 PCO\_Identifier ::= Identifier
- 111 PCO\_TypeIdentifier ::= Identifier
- 113 PCO\_Role ::= UT | LT

EXAMPLE 21 - Declaration of PCOs

| PCO Declarations |          |      |                                                     |
|------------------|----------|------|-----------------------------------------------------|
| PCO Name         | PCO Type | Role | Comments                                            |
| L                | TSAP     | LT   | Transport service access point at the lower tester. |
| U                | SSAP     | UT   | Session service access point at the upper tester.   |

Points of control and observation are usually just SAPs, but in general can be any appropriate points at which the test events can be controlled and observed. However, it is possible to define a PCO to correspond to a set of SAPs, provided all the SAPs (Service Access Point) comprising that PCO are

- at the same location (i.e., in the LT or in the UT);
- SAPs of the same service.

When a PCO corresponds to several SAPs the appropriate address is used to identify the individual SAP. PCOs are normally associated with one service access point of the (N-1) service-provider or the IUT.

NOTE - A PCO may not be related to a SAP at all. This could be the case when a layer is composed of sublayers (e.g., in the Application layer, or in the lower layers, where a subnetwork point of attachment is not a SAP).

10.9 Timer Declarations

A test suite may make use of timers. The following information shall be provided for each timer:

- a) the timer name,
- b) the optional timer duration,

where the default duration of the timer shall be an expression which may be omitted if the value cannot be established prior to execution of the test suite; 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;

- c) the time unit,

where the time unit shall be one of the following:

- 1) ps (i.e., picosecond);
- 2) ns (i.e., nanosecond);
- 3) us (i.e., microsecond);

- 4) **ms** (*i.e.*, millisecond);
- 5) **s** (*i.e.*, second);
- 6) **min** (*i.e.*, minute).

Time units are determined by the test suite designer and are fixed at the time of specification. Different timers may use different units within the same test suite. If a PICS or PIXIT entry exists, the timer declaration shall specify the same units included in the PICS/PIXIT entry.

This information shall be provided in the format shown in the following proforma:

| Timer Declarations                          |                           |                 |                   |
|---------------------------------------------|---------------------------|-----------------|-------------------|
| Timer Name                                  | Duration                  | Unit            | Comments          |
| <i>TimerIdentifier</i>                      | <i>[DeclarationValue]</i> | <i>TimeUnit</i> | <i>[FreeText]</i> |
| <b>Detailed Comments:</b> <i>[FreeText]</i> |                           |                 |                   |

Proforma 16 - Timer Declarations

#### SYNTAX DEFINITION:

- 117 TimerIdentifier ::= Identifier
- 93 DeclarationValue ::= Expression
- 120 TimeUnit ::= **ps** | **ns** | **us** | **ms** | **s** | **min**

#### EXAMPLE 22 - Declaration of timers

| Timer Declarations |          |      |                                                                                                                                         |
|--------------------|----------|------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Timer Name         | Duration | Unit | Comments                                                                                                                                |
| wait               | 15       | s    | General purpose wait.                                                                                                                   |
| no_response        | A        | min  | Used to wait for IUT to connect or react to connection establishment, longer duration than general purpose wait. Gets value from PIXIT. |
| delay_time         |          | ms   | Duration to be established during execution of the test suite.                                                                          |

## 10.10 ASP Type Definitions

### 10.10.1 Introduction

The purpose of this part of the abstract TTCN test suite is to declare the types of ASPs that may be sent or received at the declared PCOs. ASP type definitions may include ASN.1 type definitions, if appropriate.

### 10.10.2 ASP Type Definitions using tables

The following information shall be supplied for each ASP:

- a) its name,  
where the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;
- b) the PCO type associated with the ASP,  
where 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.

c) a list of the parameters associated with the ASP,  
where the following information shall be supplied for each parameter:

- 1) its name,  
where either:
- the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses; or
  - the macro symbol ( < - ) indicating that the entry in the type column identifies a set of parameters that is to be inserted directly in the list of ASP parameters; the macro symbol shall be used only with Structured Types defined in the Structured Types definitions;

- 2) its type and an optional attribute,  
where 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; or
  - as **PDU** to indicate that in the constraint for the ASP this parameter may be chained to a PDU constraint of any PDU type;

and where the optional attribute is Length;  
in which case the specification may restrict the parameter to a particular length or a range according to 10.12. The length values shall be interpreted according to table 4. The boundaries shall be specified in terms of non-negative INTEGER literals, Test Suite Parameters, Test Suite Constants or the keyword INFINITY.  
The length specifications defined for the ASP parameter type in the Test Suite Type definitions shall not conflict with the length specifications in the ASP type definition, *i.e.*, the set of strings defined by a length restriction in an ASP definition shall be a true subset of the set of strings defined by the Test Suite Type definition.  
The keyword INFINITY can be used as a value for the upper boundary in order to indicate that there is no upper limit of length.

NOTE - It is usually unnecessary to restrict the length of ASP parameters, but in some cases this may be necessary in order to effectively restrict the length of a corresponding PDU field in an underlying protocol.

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

This information shall be provided in the format shown in the following proforma:

| ASP Type Definition                                                                                           |                            |                   |
|---------------------------------------------------------------------------------------------------------------|----------------------------|-------------------|
| ASP Name : <i>ASP_Id&amp;FullId</i><br>PCO Type : <i>[PCO_TypeIdentifier]</i><br>Comments : <i>[FreeText]</i> |                            |                   |
| Parameter Name                                                                                                | Parameter Type             | Comments          |
| <i>ASP_ParIdOrMacro</i>                                                                                       | <i>Type&amp;Attributes</i> | <i>[FreeText]</i> |
| Detailed Comments: <i>[FreeText]</i>                                                                          |                            |                   |

Proforma 17 - ASP Type Definition

SYNTAX DEFINITION:

127 ASP\_Id&FullId ::= ASP\_Identifier [FullIdentifier]

128 ASP\_Identifier ::= Identifier  
 43 FullIdentifier ::= "(" BoundedFreeText ")"  
 111 PCO\_TypeIdentifier ::= Identifier  
 132 ASP\_ParIdOrMacro ::= ASP\_ParId&FullId | MacroSymbol  
 133 ASP\_ParId&FullId ::= ASP\_ParIdentifier [FullIdentifier]  
 134 ASP\_ParIdentifier ::= Identifier  
 150 MacroSymbol ::= "<"  
 154 Type&Attributes ::= (Type [LengthAttribute]) | PDU  
 331 Type ::= PredefinedType | ReferenceType  
 332 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING | CharacterString  
 333 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString | VisibleString | IA5String | GraphicString | GeneralString  
 334 ReferenceType ::= TS\_TypeIdentifier | ASP\_Identifier | PDU\_Identifier  
 335 TS\_TypeIdentifier ::= SimpleTypeIdentifier | StructIdentifier | ASN1\_TypeIdentifier  
 155 LengthAttribute ::= SingleLength | RangeLength  
 156 SingleLength ::= "[" Bound "]"  
 157 Bound ::= Number | TS\_ParIdentifier | TS\_ConstIdentifier  
 158 RangeLength ::= "[" LowerBound To UpperBound "]"  
 159 LowerBound ::= Bound  
 37 To ::= TO | "."  
 160 UpperBound ::= Bound | INFINITY

#### EXAMPLE 23 - T\_CONNECTrequest Abstract Service Primitive

The figure below shows an example from the Transport Service [ISO 8072]. This could be part of the set of ASPs used to describe the behaviour of an abstract UT in a DS test suite for the Class 0 Transport. CDA, CGA and QOS are Test Suite Types [ISO 8073].

| ASP Type Definition                                                                        |                |                               |
|--------------------------------------------------------------------------------------------|----------------|-------------------------------|
| <b>ASP Name :</b> CONreq (T_CONNECTrequest)<br><b>PCO Type :</b> TSAP<br><b>Comments :</b> |                |                               |
| Parameter Name                                                                             | Parameter Type | Comments                      |
| Cda (Called Address)                                                                       | CDA            | ... of upper tester           |
| Cga (Calling Address)                                                                      | CGA            | ... of lower tester           |
| QoS (Quality of Service)                                                                   | QOS            | should ensure class 0 is used |
| <b>Detailed Comments:</b> ASP to be sent at Transport service access point                 |                |                               |

#### 10.10.3 Use of Structured Types within ASP Type Definitions

There are two possible relationships between a Structured Type and ASP definitions which refer to it, as follows:

- a) if a parameter name is given in the definition, then the Structured Type referenced is a substructure. This allows definition of ASPs containing a multi-level substructure of parameters;
- b) if the macro symbol ( <- ) is used instead of a parameter name then this is equivalent to a macro expansion; the entry in the ASP type definition expands directly to a list of parameters without introducing an additional level of substructure.

The macro symbol shall not be used on the same line as references to types defined in ASN.1 or Simple Types, *i.e.*, only Structured Types defined in tabular form can be expanded into other Structured Types as macro expansions.

#### 10.10.4 ASP Type Definitions using ASN.1

Where more appropriate, ASPs can be specified in ASN.1. This shall be achieved by an ASN.1 definition using the ASN.1 syntax as defined in ISO/IEC 8824. The following information shall be supplied for each ASN.1 ASP:

- a) its name,

where the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;

b) the PCO type associated with the ASP,

where 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;

c) the ASN.1 ASP type definition,

which shall follow the syntax defined in ISO/IEC 8824. For identifiers within that definition the hyphen symbol ( - ) shall not be used. The underscore symbol ( \_ ) may be used instead. The ASP identifier in the table header is the name of the first type defined in the table body.

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.

ASN.1 comments can be used within the table body. The comments column shall not be present in this table.

This information shall be provided in the following proforma:

| ASN.1 ASP Type Definition                                                                                     |
|---------------------------------------------------------------------------------------------------------------|
| ASP Name : <i>ASP_Id&amp;FullId</i><br>PCO Type : <i>[PCO_TypeIdentifier]</i><br>Comments : <i>[FreeText]</i> |
| Type Definition                                                                                               |
| <i>ASN1_Type&amp;LocalTypes</i>                                                                               |
| Detailed Comments: <i>[FreeText]</i>                                                                          |

Proforma 18 - ASN.1 ASP Type Definition

SYNTAX DEFINITION:

- 127 ASP\_Id&FullId ::= ASP\_Identifier [FullIdentifier]
- 128 ASP\_Identifier ::= Identifier
- 43 FullIdentifier ::= "(" BoundedFreeText ")"
- 111 PCO\_TypeIdentifier ::= Identifier
- 57 ASN1\_Type&LocalTypes ::= ASN1\_Type {ASN1\_LocalType}
- 58 ASN1\_Type ::= Type  
/\* REFERENCE -Where Type is a non-terminal defined in ISO/IEC 8824 \*/
- 59 ASN1\_LocalType ::= Typeassignment  
/\* REFERENCE -Where Typeassignment is a non-terminal defined in ISO/IEC 8824 \*/

10.10.5 ASN.1 ASP Type Definitions by Reference

ASPs can be specified by a precise reference to an ASN.1 ASP defined in an OSI standard or by referencing an ASN.1 type defined in an ASN.1 module attached to the test suite. The following information shall be supplied for each ASP:

a) its name,

where this name may be used throughout the entire test suite;

b) the PCO type associated with the ASP;

where 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;

- c) the type reference,  
which shall follow the identifier rules stated in ISO/IEC 8824;
- d) the module identifier,  
which consists of a module reference that shall follow the identifier rules stated in ISO/IEC 8824 and an optional ObjectIdentifier.

This information shall be provided in the following proforma:

| ASN.1 ASP Type Definitions By Reference |                      |                |                   |            |
|-----------------------------------------|----------------------|----------------|-------------------|------------|
| ASP Name                                | PCO Type             | Type Reference | Module Identifier | Comments   |
| ASP_Id&FullId                           | [PCO_TypeIdentifier] | TypeReference  | ModuleIdentifier  | [FreeText] |
| Detailed Comments: [FreeText]           |                      |                |                   |            |

Proforma 19 - ASN.1 ASP Type Definitions By Reference

#### SYNTAX DEFINITION:

- 127 ASP\_Id&FullId ::= ASP\_Identifier [FullIdentifier]  
 128 ASP\_Identifier ::= Identifier  
 43 FullIdentifier ::= "(" BoundedFreeText ")"  
 111 PCO\_TypeIdentifier ::= Identifier  
 63 TypeReference ::= typereference  
 /\* REFERENCE -Where typereference is defined in ISO/IEC 8824:1990, subclause 8.2 \*/  
 65 ModuleIdentifier ::= ModuleIdentifier  
 /\* REFERENCE -Where ModuleIdentifier is a non-terminal defined in ISO/IEC 8824 \*/

ASN.1 identifiers type references and value references may contain hyphens. In order to be able to use imported definitions in TTCN it is necessary to change the hyphens to underscore (see A.4.2.1).

## 10.11 PDU Type Definitions

### 10.11.1 Introduction

The purpose of this part of the abstract TTCN test suite is to declare the types of the PDUs that may be sent or received either directly or embedded in ASPs at the declared PCOs. PDU type definitions may include ASN.1 type definitions, if appropriate. PDU definitions define the set of PDUs exchanged with the IUT which are syntactically valid with respect to the ATS but not necessarily valid with respect to the protocol specification.

It is required to declare all fields of the PDUs that are defined in the relevant protocol standard, either explicitly or implicitly by referring to encoding rules (ASN.1 encoding rules, if applicable).

The encoding of PDU fields shall follow that as defined in the relevant protocol specification.

### 10.11.2 PDU Type Definitions using tables

The definition of PDUs is similar to that of ASPs. The following information shall be supplied for each PDU:

- a) its name,  
where the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses;
- b) the PCO type associated with the PDU,  
where the PCO type shall be one of the PCO types used in the PCO declarations; if a PDU is sent or received only embedded in ASPs within the whole test suite, specifying the PCO type is optional; if only a single PCO is defined within a test suite, specifying the PCO type in a PDU type definition is optional;

c) a list of the fields associated with the PDU,  
where the following information shall be supplied for each field:

1) its name,  
where either:

- the full name, as given in the appropriate protocol standard, shall be used; if an abbreviation is used, then the full name shall follow in parentheses; or
- the macro symbol ( <- ) indicating that the entry in the type column identifies a set of fields that is to be inserted directly in the list of PDU fields; the macro symbol shall be used only with Structured Types defined in the Structured Type definitions;

2) its type and an optional attribute;

where 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;

and where the optional attribute is Length;

in which case the specification may restrict the field to a particular length or a range according to 10.12. The length values shall be interpreted according to table 4. The boundaries shall be specified in terms of non-negative INTEGER literals, Test Suite Parameters, Test Suite Constants or the keyword INFINITY.

The length specifications defined for the PDU field type in the Test Suite Type definitions shall not conflict with the length specifications in the PDU type definition, *i.e.*, the set of strings defined by a length restriction in a PDU definition shall be a true subset of the set of strings defined by the Test Suite Type definition.

The keyword INFINITY can be used as a value for the upper boundary in order to indicate that there is no upper limit of length.

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

This information shall be provided in the format shown in the following proforma:

| PDU Type Definition                                                                                                                |                            |                   |
|------------------------------------------------------------------------------------------------------------------------------------|----------------------------|-------------------|
| <b>PDU Name :</b> <i>PDU_Id&amp;FullId</i><br><b>PCO Type :</b> <i>[PCO_TypeIdentifier]</i><br><b>Comments :</b> <i>[FreeText]</i> |                            |                   |
| Field Name                                                                                                                         | Field Type                 | Comments          |
| <i>PDU_FieldIdOrMacro</i>                                                                                                          | <i>Type&amp;Attributes</i> | <i>[FreeText]</i> |
| <b>Detailed Comments:</b> <i>[FreeText]</i>                                                                                        |                            |                   |

Proforma 20 - PDU Type Definition

SYNTAX DEFINITION:

- 144 PDU\_Id&FullId ::= PDU\_Identifier [FullIdentifier]
- 145 PDU\_Identifier ::= Identifier
- 43 FullIdentifier ::= "(" BoundedFreeText ")"