
**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)



Contents

	Page
1 Scope	1
2 Normative references	1
3 Definitions	3
3.1 Basic terms from ISO/IEC 9646-1	3
3.2 Terms from ISO/IEC 7498-1	4
3.3 Terms from ISO/IEC 10731	4
3.4 Terms from ISO/IEC 8824-1	4
3.5 Terms from ISO/IEC 8825-1	5
3.6 TTCN specific terms	5
4 Abbreviations	8
4.1 Abbreviations defined in ISO/IEC 9646-1	8
4.2 Abbreviations defined in ISO/IEC 9646-2	8
4.3 Other abbreviations	9
5 The syntax forms of TTCN	9
6 Compliance	10
7 Conventions	10
7.1 Introduction	10
7.2 Syntactic metanotation	10
7.3 TTCN.GR table proformas	11
7.3.1 Introduction	11
7.3.2 Single TTCN object tables	11
7.3.3 Multiple TTCN object tables	12
7.3.4 Alternative compact tables	12
7.3.5 Specification of proformas	13
7.4 Free Text and Bounded Free Text	13
8 Concurrency in TTCN	13
8.1 Test Components	13
8.2 Test Component Configurations	13
9 TTCN test suite structure	15
9.1 Introduction	15
9.2 Test Group References	15
9.3 Test Step Group References	15
9.4 Default Group References	15
9.5 Parts of a TTCN test suite	15
10 Test Suite Overview	16
10.1 Introduction	16
10.2 Test Suite Structure	16
10.3 Test Case Index	18
10.4 Test Step Index	19
10.5 Default Index	20
10.6 Test Suite Exports	21
10.7 The Import Part	22
10.7.1 Introduction	22
10.7.2 Imports	22
11 Declarations Part	24
11.1 Introduction	24
11.2 TTCN types	24
11.2.1 Introduction	24
11.2.2 Predefined TTCN types	24
11.2.3 Test Suite Type Definitions	26

© ISO/IEC 1998

All rights reserved. Unless otherwise specified, 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 Geneve 20 • Switzerland.
Printed in Switzerland.

11.3	TTCN operators and TTCN operations	32
11.3.1	Introduction	32
11.3.2	TTCN operators	32
11.3.3	Predefined operations	33
11.3.4	Test Suite Operation definitions and descriptions	35
11.4	Test Suite Parameter Declarations	40
11.5	Test Case Selection Expression Definitions	41
11.6	Test Suite Constant Declarations	42
11.7	Test Suite Constant Declarations by Reference	43
11.8	TTCN variables	44
11.8.1	Test Suite Variable Declarations	44
11.8.2	Binding of Test Suite Variables	45
11.8.3	Test Case Variable Declarations	45
11.8.4	Binding of Test Case Variables	46
11.9	PCO Type Declaration	46
11.10	PCO Declarations	47
11.11	CP Declarations	49
11.12	Timer Declarations	50
11.13	Test Components and Configuration Declarations	51
11.13.1	Test Components	51
11.13.2	Test Component Configuration Declarations	53
11.14	ASP Type Definitions	55
11.14.1	Introduction	55
11.14.2	ASP Type Definitions using tables	55
11.14.3	Use of Structured Types within ASP Type Definitions	56
11.14.4	ASP Type Definitions using ASN.1	57
11.14.5	ASN.1 ASP Type Definitions by Reference	58
11.15	PDU Type Definitions	59
11.15.1	Introduction	59
11.15.2	PDU Type Definitions using tables	59
11.15.3	Use of Structured Types within PDU definitions	61
11.15.4	PDU Type Definitions using ASN.1	61
11.15.5	ASN.1 PDU Type Definitions by Reference	63
11.16	Test Suite Encoding Information	64
11.16.1	Encoding Definitions	64
11.16.2	Encoding Variations	65
11.16.3	Invalid Field Encoding Definitions	66
11.16.4	Application of encoding rules	68
11.17	CM Type Definitions	69
11.17.1	Introduction	69
11.17.2	CM Type Definitions using tables	69
11.17.3	CM Type Definitions using ASN.1	70
11.18	String length specifications	70
11.19	ASP, PDU and CM Definitions for SEND events	71
11.20	ASP, PDU and CM Definitions for RECEIVE events	72
11.21	Alias Definitions	72
11.21.1	Introduction	72
11.21.2	Expansion of Aliases	72
12	Constraints Part	73
12.1	Introduction	73
12.2	General principles	73
12.3	Parameterization of constraints	74
12.4	Chaining of constraints	74
12.5	Constraints for SEND events	75
12.6	Constraints for RECEIVE events	75
12.6.1	Matching values	75
12.6.2	Matching mechanisms	75
12.6.3	Specific Value	76
12.6.4	Instead of Value	76

12.6.5 Inside Values	79
12.6.6 Attributes of values	80
13 Specification of constraints using tables	81
13.1 Introduction	81
13.2 Structured Type Constraint Declarations	81
13.3 ASP Constraint Declarations	83
13.4 PDU Constraint Declarations	83
13.5 Parameterization of constraints	85
13.6 Base constraints and modified constraints	85
13.7 Formal parameter lists in modified constraints	86
13.8 CM Constraint Declarations	86
14 Specification of constraints using ASN.1	87
14.1 Introduction	87
14.2 ASN.1 Type Constraint Declarations	87
14.3 ASN.1 ASP Constraint Declarations	88
14.4 ASN.1 PDU Constraint Declarations	89
14.5 Parameterized ASN.1 constraints	90
14.6 Modified ASN.1 constraints	90
14.7 Formal parameter lists in modified ASN.1 constraints	91
14.8 ASP Parameter and PDU field names within ASN.1 constraints	91
14.9 ASN.1 CM Constraint Declarations	92
15 The Dynamic Part	92
15.1 Introduction	92
15.2 Test Case dynamic behaviour	92
15.2.1 Specification of the Test Case Dynamic Behaviour table	92
15.2.2 The Test Case Dynamic Behaviour proforma	93
15.2.3 Structure of the Test Case behaviour	94
15.2.4 Concurrent Test Case Behaviour Description	94
15.2.5 Line numbering and continuation	95
15.3 Test Step dynamic behaviour	95
15.3.1 Specification of the Test Step Dynamic Behaviour table	95
15.3.2 The Test Step Dynamic Behaviour proforma	96
15.4 Default dynamic behaviour	97
15.4.1 Default behaviour	97
15.4.2 Specification of the Default Dynamic Behaviour table	97
15.4.3 The Default Dynamic Behaviour proforma	97
15.5 The behaviour description	98
15.6 The tree notation	98
15.7 Tree names and parameter lists	99
15.7.1 Introduction	99
15.7.2 Trees with parameters	99
15.8 TTCN statements	99
15.9 TTCN test events	100
15.9.1 Sending and receiving events	100
15.9.2 Receiving events	100
15.9.3 Sending events	100
15.9.4 Lifetime of events	101
15.9.5 Execution of the behaviour tree	101
15.9.6 The IMPLICIT SEND event	103
15.9.7 The OTHERWISE event	104
15.9.8 OTHERWISE and concurrent TTCN	105
15.9.9 The TIMEOUT event	105
15.9.10 Concurrent TTCN events and constructs	105
15.10 TTCN expressions	106
15.10.1 Introduction	106
15.10.2 References for ASN.1 defined data objects	108
15.10.3 References for data objects defined using tables	110
15.10.4 Assignments	110
15.10.5 Qualifiers	111

15.10.6 Event lines with assignments and qualifiers	111
15.11 Pseudo-events	112
15.12 Timer management	112
15.12.1 Introduction	112
15.12.2 The START operation	113
15.12.3 The CANCEL operation	113
15.12.4 The READTIMER operation	114
15.13 The ATTACH construct	114
15.13.1 Introduction	114
15.13.2 Scope of tree attachment	114
15.13.3 Tree attachment basics	115
15.13.4 The meaning of tree attachment	115
15.13.5 Passing parameterized constraints	117
15.13.6 Recursive tree attachment	117
15.13.7 Tree attachment and Defaults	118
15.14 Labels and the GOTO construct	118
15.15 The REPEAT construct	119
15.16 The Constraints Reference	119
15.16.1 Purpose of the Constraints Reference column	119
15.16.2 Passing parameters in Constraint References	120
15.16.3 Constraints and qualifiers and assignments	120
15.17 Verdicts	120
15.17.1 Introduction	120
15.17.2 Preliminary results	121
15.17.3 Final verdict	121
15.17.4 Verdicts and OTHERWISE	122
15.17.5 Verdict assignment in concurrent TTCN	122
15.18 The meaning of Defaults	122
15.18.1 Introduction	122
15.18.2 Default References	123
15.18.3 The RETURN statement	124
15.18.4 The ACTIVATE statement	124
15.18.5 Defaults and tree attachment	124
15.18.6 Tree Attachment, Defaults, Activate and Return	126
15.18.7 Defaults and CREATE	132
15.18.8 Defaults and CMs	132
16 Page continuation	133
16.1 Page continuation of TTCN tables	133
16.2 Page continuation of dynamic behaviour tables	133
A(normative) Syntax and static semantics of TTCN	134
A.1 Introduction	134
A.2 Conventions for the syntax description	134
A.2.1 Syntactic metanotation	134
A.2.2 TTCN.MP syntax definitions	134
A.3 The TTCN.MP syntax productions in BNF	136
A.3.1 TTCN Specification	136
A.3.2 TTCN Module	136
A.3.2.1 TTCN Module Overview Part	136
A.3.2.2 TTCN Module Import Part	136
A.3.3 Test suite	137
A.3.3.1 The Test Suite Overview	137
A.3.3.2 Test Suite Index	137
A.3.3.3 Test Suite Structure	137
A.3.3.4 Test Case Index	138
A.3.3.5 Test Step Index	138
A.3.3.6 Default Index	138
A.3.3.7 Test Suite Exports	138
A.3.3.8 The Import Part	138
A.3.3.9 The Declarations Part	138

A.3.3.10	Definitions	138
A.3.3.11	Parameterization and Selection	143
A.3.3.12	Declarations	144
A.3.3.13	ASP, PDU and CM Type Definitions	149
A.3.3.14	The Constraints Part	154
A.3.3.15	Test Suite Type Constraint Declarations	154
A.3.3.16	Structured Type Constraint Declarations	154
A.3.3.17	ASN.1 Type Constraint Declarations	155
A.3.3.18	ASP Constraint Declarations	155
A.3.3.19	Tabular ASP Constraint Declarations	155
A.3.3.20	ASN.1 ASP Constraint Declarations	156
A.3.3.21	PDU Constraint Declarations	156
A.3.3.22	Tabular PDU Constraint Declarations	156
A.3.3.23	ASN.1 PDU Constraint Declarations	158
A.3.3.24	CM Constraint Declarations	159
A.3.3.25	Tabular CM Constraint Declaration	159
A.3.3.26	ASN.1 CM Constraint Declaration	159
A.3.3.27	The Dynamic Part	159
A.3.3.28	Test Cases	159
A.3.3.29	Test Step Library	160
A.3.3.30	Default Library	160
A.3.3.31	Behaviour descriptions	160
A.3.3.32	Behaviour lines	161
A.3.3.33	TTCN statements	161
A.3.3.34	Expressions	163
A.3.3.35	Timer operations	164
A.3.3.36	Types	165
A.3.3.37	Values	165
A.3.3.38	Miscellaneous productions	167
A.4	General static semantics requirements	168
A.4.1	Introduction	168
A.4.2	Uniqueness of identifiers	168
A.5	Differences between TTCN.GR and TTCN.MP	172
A.5.1	Differences in syntax	172
A.5.2	Additional static semantics in the TTCN.MP	173
A.6	List of BNF production numbers	174
A.6.1	Introduction	174
A.6.2	The production index	174
B(normative)	Operational semantics of TTCN	181
B.1	Introduction	181
B.2	Precedence	181
B.3	Processing of test case errors	181
B.4	Converting a modularized test suite to an equivalent expanded test suite	181
B.5	TTCN operational semantics	183
B.5.1	Introduction	183
B.5.2	The pseudo-code notation	183
B.5.2.1	Introduction	183
B.5.2.2	Procedures and functions	183
B.5.2.3	Processes	183
B.5.2.4	Natural language within pseudo-code	184
B.5.2.5	Levels and alternatives	184
B.5.3	Execution of a Test Suite	184
B.5.3.1	Introduction	184
B.5.4	Execution of a Test Case	185
B.5.4.1	Execution of a Test Case - pseudo-code	185
B.5.4.2	Execution of a Test Case or Test Component - natural language	186
B.5.5	Expanding a set of alternatives	186
B.5.5.1	Introduction	186
B.5.5.2	Saving Defaults	187

B.5.5.3 Expansion of REPEAT constructs	187
B.5.5.4 Appending default behaviour	188
B.5.5.5 Expanding attached trees	188
B.5.6 Evaluation of an Event Line	189
B.5.6.1 Pseudo-code	189
B.5.6.2 Natural language description	189
B.5.7 Functions for TTCN events	189
B.5.7.1 Functions for TTCN events - pseudo-code	189
B.5.7.2 Functions for TTCN events - natural language description	190
B.5.8 Execution of the SEND event	190
B.5.8.1 Execution of the SEND event - pseudo-code	190
B.5.8.2 Execution of the SEND event - natural language description	191
B.5.9 Execution of the RECEIVE event	191
B.5.9.1 Execution of the RECEIVE event - pseudo-code	191
B.5.9.2 Execution of the RECEIVE event - natural language description	192
B.5.10 Execution of the OTHERWISE event	193
B.5.10.1 Execution of the OTHERWISE event - pseudo-code	193
B.5.10.2 Execution of the OTHERWISE event - natural language description	193
B.5.11 Execution of the TIMEOUT event	193
B.5.11.1 Execution of the TIMEOUT event - pseudo-code	193
B.5.11.2 Execution of the TIMEOUT event - natural language description	194
B.5.12 Execution of the DONE event	195
B.5.12.1 Execution of the DONE event - pseudo-code	195
B.5.12.2 Execution of the DONE event - natural language description	195
B.5.13 Execution of the IMPLICIT SEND event	196
B.5.13.1 Execution of the IMPLICIT SEND event - pseudo-code	196
B.5.13.2 Execution of IMPLICIT SEND - natural language description	196
B.5.14 Execution of a pseudo-event	196
B.5.14.1 Execution of a pseudo-event - pseudo-code	196
B.5.14.2 Execution of PSEUDO-EVENTS - natural language description	196
B.5.15 Execution of BOOLEAN expressions	196
B.5.15.1 Execution of BOOLEAN expressions - pseudo-code	196
B.5.15.2 Execution of BOOLEAN expressions - natural language description	197
B.5.16 Execution of assignments	197
B.5.16.1 Execution of assignments - pseudo-code	197
B.5.16.2 Execution of ASSIGNMENTS - natural language description	197
B.5.17 Execution of TIMER operations	197
B.5.17.1 Execution of TIMER operations - pseudo-code	197
B.5.17.2 Execution of START timer - natural language description	198
B.5.17.3 Execution of CANCEL timer - natural language description	198
B.5.17.4 Execution of READTIMER - natural language description	198
B.5.18 Functions for TTCN constructs	198
B.5.18.1 Functions for TTCN constructs - pseudo-code	198
B.5.18.2 Functions for TTCN constructs - natural language description	198
B.5.19 Execution of the ACTIVATE construct	198
B.5.19.1 Execution of the ACTIVATE construct - pseudo-code	198
B.5.19.2 Execution of the ACTIVATE construct - natural language description	199
B.5.20 Execution of the CREATE construct	199
B.5.20.1 Execution of the CREATE event - pseudo-code	199
B.5.20.2 Execution of the CREATE event - natural language description	199
B.5.21 Execution of the GOTO construct	200
B.5.22 Execution of the RETURN construct	200
B.5.23 The verdict	200
B.5.23.1 The verdict - pseudo-code	200
B.5.23.2 The VERDICT - natural language description	201
B.5.24 The Conformance Log	201
B.5.24.1 The LOG - pseudo-code	201
B.5.24.2 The conformance log - natural language description	201
B.5.25 Tree handling functions and procedures	202

B.5.26 Miscellaneous functions used by the pseudo-code	203
C(normative) TTCN Modules	206
C.1 Introduction	206
C.2 TTCN Module Overview Part	206
C.2.1 Introduction	206
C.2.2 TTCN Module Exports	206
C.2.3 TTCN Module Structure	208
C.2.4 Test Case Index	209
C.2.5 Test Step Index	209
C.2.6 Default Index	209
C.3 Import Part	209
C.3.1 Introduction	209
C.3.2 External	209
C.3.3 Import	210
D(normative) Test Suite Index	211
D.1 Introduction	211
D.2 The Test Suite Index	211
D.2.1 Introduction	211
D.2.2 The Test Suite Index	211
E(normative) Compact proformas	212
E.1 Introduction	212
E.2 Compact proformas for constraints	212
E.2.1 Requirements	212
E.2.2 Compact proformas for ASP constraints	212
E.2.3 Compact proformas for PDU constraints	213
E.2.3.2 Parameterized compact constraints	214
E.2.4 Compact proformas for Structured Type constraints	215
E.2.5 Compact proformas for ASN.1 constraints	217
E.3 Compact proforma for Test Cases	218
E.3.1 Requirements	218
E.3.2 Compact proforma for Test Case dynamic behaviours	218
F(informative) Examples	220
F.1 Examples of tabular constraints	220
F.1.1 ASP and PDU definitions	220
F.1.1.1 Flat type definition	220
F.1.1.2 Structured Type definition	220
F.1.1.3 Special type PDU, in order to allow use of (static) chaining of constraints	221
F.1.2 ASP/PDU constraints	221
F.2 Examples of ASN1 constraints	224
F.2.1 ASP and PDU definitions	224
F.2.2 ASN.1 ASP/PDU constraints	225
F.2.3 Further examples of ASN.1 constraints	229
F.3 Base and modified constraints	231
F.4 Type definition using macros	232
F.5 Use of REPEAT	234
F.6 Test suite operations	234
F.7 Example of a Test Suite Overview	235
F.8 Example of a Test Case in TTCN.MP Form	237
F.9 Use of Component Reference for Field Value Assignment in Constraints	239
F.10 Multi-Party Testing	242
F.11 Multiplexing/Demultiplexing	243
F.12 Splitting and Recombining	244
F.13 Multi-Protocol Test Cases	245
F.14 Example of Modular TTCN	246
F.15 Example of CREATE and DONE	246
G(informative) Style guide	253
G.1 Introduction	253
G.2 Test case structure	253
G.3 Use of TTCN with different abstract test methods	254

G.3.1 Introduction.	254
G.3.2 TTCN and the LS test method	254
G.3.3 TTCN and the DS test method	254
G.3.4 TTCN and the CS test method	254
G.3.5 TTCN and the RS test method	255
G.4 Use of Defaults	255
G.5 Limiting the execution time of a Test Case	255
G.6 Structured Types	255
G.7 Abbreviations.	256
G.8 Test descriptions	256
G.9 Assignments on SEND events	256
G.10 Multi-service PCOs	256
H (informative) Index	257
H.1 Introduction	257
H.2 The Index	257

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 the Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open systems interconnection, data management and open distributed processing*.

This second edition cancels and replaces the first edition (ISO/IEC 9646-3:1992), which has been technically revised.

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 Statements*

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

Introduction

This part of ISO/IEC 9646, one of a multi-part International Standard defines a 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;
- d) to provide a capability to use concurrency in the specification of abstract test cases, when appropriate, in both multi-party testing and single-party testing.

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 positions 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 page intentionally left blank

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

1 Scope

1.1 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.

1.2 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.

1.3 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.

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

The specification of test cases in which more than one behaviour description is to be run in parallel is dealt with by the concurrency features (particularly involving the definition of Test Components and Test Component Configurations).

1.4 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.

1.5 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 Physical layer protocols;
- b) the relationship between TTCN and formal description techniques;
- c) the means of realization of executable test suites (ETS) from abstract test suites.

1.6 This part of ISO/IEC 9646 defines mechanisms for using concurrency in the specification of abstract test cases. Concurrency in TTCN is applicable to the specification of test cases:

- a) in a multi-party testing context;
- b) which handle multiplexing and demultiplexing in either a single-party or multi-party testing context;
- c) which handle splitting and recombining in either a single-party or multi-party testing context;
- d) in a single-party testing context when the complexity of the protocol or set of protocols handled by the IUT is such that concurrency can simplify the specification of the test case.

1.7 TTCN modules are defined to allow sharing of common TTCN specifications between 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 indicated 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/IEC 7498-1 : 1994, *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*.

(See also ITU-T Recommendation X.200 : 1994.)

ISO/IEC 8824-1: 1995, *Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

(See also ITU-T Recommendation X.680 : 1994.)

ISO/IEC 8824-1: 1995/Amd. 1: 1996, *Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation - Amendment 1: Rules of extensibility.*

(See also ITU-T Recommendation X.680 Amendment 1 : 1995.)

ISO/IEC 8824-2: 1995, *Information technology - Abstract Syntax Notation One (ASN.1): Information object specification.*

(See also ITU-T Recommendation X.681 : 1994.)

ISO/IEC 8824-2: 1995/Amd. 1 : 1996, *Information technology - Abstract Syntax Notation One (ASN.1): Information object specification - Amendment 1: Rules of extensibility.*

(See also ITU-T Recommendation X.681 Amendment 1 : 1995.)

ISO/IEC 8824-3 : 1995, *Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification.*

(See also ITU-T Recommendation X.682 : 1994.)

ISO/IEC 8824-4 : 1995, *Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.*

(See also ITU-T Recommendation X.683 : 1994.)

ISO/IEC 8825-1 : 1995, *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

(See also ITU-T Recommendation X.690 : 1994.)

ISO/IEC 8825-2 : 1996, *Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

(See also ITU-T Recommendation X.690 : 1995.)

ISO/IEC 9646-1 : 1994, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General concepts.*

(See also ITU-T Recommendation X.290 : 1995)

ISO/IEC 9646-2 : 1994, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 2: Abstract test suite specification.*

(See also ITU-T Recommendation X.291 : 1995)

ISO/IEC 9646-4 : 1994, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 4: Test realization.*

(See also ITU-T Recommendation X.293 : 1995)

ISO/IEC 9646-5 : 1994, *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 ITU-T Recommendation X.294 : 1995)

ISO/IEC 9646-6 : 1994, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 6: Protocol profile test specification.*

(See also ITU-T Recommendation X.295 : 1995)

ISO/IEC 9646-7 : 1995, *Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 7: Implementation Conformance Statements.*

(See also ITU-T Recommendation X.296 : 1995)

ISO/IEC 10646-1 : 1993, *Information technology - Multiple-Octet Coded Character Set (UCS): Architecture and basic multilingual plane.*

ISO/IEC 10731: 1994, *Information technology - Open Systems Interconnection - Basic Reference Model - Conventions for the definition of OSI Services.*

(See also ITU-T Recommendation X.210 : 1993.)

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
- 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/IEC 7498-1

The following terms defined in ISO/IEC 7498-1: 1994 apply:

- a) (N)-layer (particularly for application, session and transport layers)
- b) (N)-protocol-data-unit
- c) (N)-service-access-point
- d) subnetwork
- e) transfer syntax

3.3 Terms from ISO/IEC 10731

The following terms defined in ISO/IEC 10731: 1994 apply:

- a) OSI-service-provider

3.4 Terms from ISO/IEC 8824-1

The following terms defined in ISO/IEC 8824-1: 1995 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 8825-1

The following term defined in ISO/IEC 8825-1: 1995 applies:

- a) encoding

3.6 TTCN specific terms

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

- 3.6.1 applicable encoding rules:** The actual encoding rules that are to be used when sending or receiving a PDU, after all relevant encoding defaults and overrides, if any, have been combined.
- 3.6.2 attach construct:** A TTCN statement which attaches a Test Step to a calling tree.
- 3.6.3 base constraint:** Specifies a set of default values for each and every field in an ASP or PDU type definition.
- 3.6.4 base type:** The type from which a type defined in a test suite is derived.
- 3.6.5 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.
- 3.6.6 behaviour tree:** A specification of a set of sequences of test events, and other TTCN statements.
- 3.6.7 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.
- 3.6.8 calling tree:** The behaviour tree to which a Test Step is attached.
- 3.6.9 compact constraint table:** Declaration of a set of constraints for an ASP, PDU or Structured Type arranged in a single table.
- 3.6.10 compact test case table:** Declaration of a set of Test Cases for a given Test Group arranged in a single table.
- 3.6.11 concurrent test case:** A test case which is specified using concurrent TTCN.
- 3.6.12 concurrent TTCN:** TTCN which uses test components and test component configurations in order to express concurrency in the dynamic behaviour of test cases.
- 3.6.13 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.
- 3.6.14 constraints reference:** A reference to a constraint, given in a behaviour line.
- 3.6.15 coordination message [CM]:** An item of structured information which may be transferred from one Test Component to another at a Coordination Point.
- 3.6.16 coordination point [CP]:** A point within a testing environment, assigned to two Test Components in a Test Component Configuration, where CMs may be exchanged asynchronously between these Test Components.
- 3.6.17 declarations part:** That part of a TTCN test suite concerned with the definition and/or declaration of all non-predefined objects that are used in the test suite.
- 3.6.18 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.
- 3.6.19 default group:** A named set of default behaviours.
- 3.6.20 default group reference:** A path specifying the logical location of a Default in the Default Library.
- 3.6.21 default identifier:** A unique name for a Default.
- 3.6.22 default library:** The set of the Default behaviours in a test suite.
- 3.6.23 default reference:** A reference to a Default in the Default Library from a Test Case or Test Step table.
- 3.6.24 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.

- 3.6.25 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.
- 3.6.26 dynamic part:** That part of a TTCN test suite concerned with the specification of Test Case, Test Step and Default dynamic behaviour descriptions.
- 3.6.27 expanded test suite:** A test suite with all imported objects expanded. This will be a result of converting of a modularized test suite according to the algorithm in annex B.
- 3.6.28 explicit external:** A named object in the External table. An object which is explicitly declared as external in a module is to be explicitly defined or exported as an external object.
- 3.6.29 explicitly defined object:** An object for which a definition or declaration exists in the module or test suite.
- 3.6.30 explicitly exported object:** A named object in the Exports tables being available for use. If the object is an imported object, the name of the source object is to be given.
- 3.6.31 explicitly imported object:** A named object in the Import tables being available for explicit references.
- 3.6.32 exported object:** An explicitly defined object or explicitly imported object in a source object, made available for use in any other module or test suite. An exported object is either an explicitly exported object or an implicitly exported object.
- 3.6.33 external object:** An object being referred to by its name in a module, but neither imported nor explicitly defined. An external object is to be declared in the External table. An external object may be either explicitly external or implicitly external.
- 3.6.34 global result variable:** A predefined test case variable maintained by a Main Test Component in the MPyT context or by the test case in the SPyT context to record the accumulated effect of all the preliminary results of the test case in order to determine the test verdict.
- 3.6.35 implicit external:** An externally declared object in an export table which is omitted from a corresponding Import table.
- 3.6.36 implicitly exported object:** An explicitly defined object or explicitly imported object, which is not itself explicitly exported but which is referred to by an explicitly exported object.
- 3.6.37 implicitly imported object:** An object referred to by some explicitly imported object. The use of an implicitly imported object is restricted to the explicitly imported objects (from the same source object) referring to it.
- 3.6.38 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.
- 3.6.39 imported object:** An object copied from some other source object, being available for use. An imported object is either an explicitly imported object or an implicitly imported object.
- 3.6.40 level of indentation:** Indicates the tree structure of a behaviour description. It is reflected in the behaviour description by indentation of text.
- 3.6.41 local result variable:** A predefined variable maintained by a Test Component to record the accumulated effect of its preliminary results.
- 3.6.42 local tree:** A behaviour tree defined in the same proforma as its calling tree.
- 3.6.43 main test component [MTC]:** The single Test Component in a Test Component Configuration responsible for creating and controlling Parallel Test Components and computing and assigning the test verdict.
- 3.6.44 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.
- 3.6.45 modularized test suite:** A test suite containing Import tables.
- 3.6.46 module:** A self-contained collection of TTCN objects. All referenced objects are either explicitly defined in the Module, are imported from other sources or are defined as external objects in the module.
- 3.6.47 non-concurrent test case:** A test case which is specified in TTCN but without using concurrent TTCN.
- 3.6.48 object:** An element of one of the object categories listed in A.4.2.2 (for TTCN objects with a globally unique identifier) and A.4.2.6 (for ASN.1 identifiers which are globally unique throughout the test suite).
- 3.6.49 operational semantics:** Semantics explaining the execution of a TTCN behaviour tree.

- 3.6.50 original source object:** The module or test suite where an object is explicitly defined.
- 3.6.51 otherwise event:** The TTCN mechanism for dealing with unforeseen test events in a controlled way.
- 3.6.52 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.
- 3.6.53 parallel test component [PTC]:** A test component created by the main test component.
- 3.6.54 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.
- 3.6.55 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.
- 3.6.56 qualified event:** An event that has an associated Boolean expression.
- 3.6.57 receive event:** The receipt of an ASP or PDU at a named or implied PCO.
- 3.6.58 result variable:** A predefined test case variable for storing preliminary results. In non-concurrent TTCN there is one result variable called R. In concurrent TTCN, there is one global result variable called R, each PTC has a local result variable called R, and the MTC has a local result variable called MTC_R.
- 3.6.59 root tree:** The main behaviour tree of a Test Case, occurring at the level of entry into the Test Case.
- 3.6.60 send event:** The sending of an ASP or PDU to a named or implied PCO.
- 3.6.61 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.
- 3.6.62 single constraint table:** A declaration of a constraint for a single ASP or PDU of a given type arranged in a single table.
- 3.6.63 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.
- 3.6.64 source object:** A module or test suite which is imported and has a corresponding Import table.
- 3.6.65 specific value:** A value in TTCN which does not contain any matching mechanism or unbound variable.
- 3.6.66 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.
- 3.6.67 static semantics:** Semantic rules that restrict the usage of the TTCN syntax.
- 3.6.68 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.
- 3.6.69 submodule:** A module which is included in another module.
- 3.6.70 test case identifier:** A unique name for a Test Case.
- 3.6.71 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.
- 3.6.72 test component:** A named subdivision of a concurrent test case capable of being executed in parallel with other test components, and declared as having a fixed number of PCOs and a fixed or maximal number of CPs.
- 3.6.73 test component configuration:** A fixed arrangement of Test Components, PCOs and CPs that is declared for use in concurrent test cases.
- 3.6.74 test group reference:** A path specifying the logical location of a Test Case in the ATS structure.
- 3.6.75 test step group:** A named set of test steps.
- 3.6.76 test step group reference:** A path specifying the logical location of a Test Step in the Test Step Library.
- 3.6.77 test step identifier:** A unique name for a Test Step.
- 3.6.78 test step library:** The set of the Test Step dynamic behaviour descriptions in the test suite, that are not local Test Steps.

- 3.6.79 test step objective:** An informal statement of what the Test Step is meant to accomplish.
- 3.6.80 test suite constant:** One of a set of constants, *not* derived from the PICS or PIXIT, which will remain constant throughout the test suite.
- 3.6.81 test suite parameter:** One of a set of constants derived from the PICS or PIXIT which globally parameterize a test suite.
- 3.6.82 test suite variable:** One of a set of variables declared globally to a test suite and retains its value between Test Cases.
- 3.6.83 timeout event:** An event which is used within a behaviour tree to check for expiration of a specified timer.
- 3.6.84 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.
- 3.6.85 tree header:** An identifier for a local tree followed by an optional list of formal parameters for the tree.
- 3.6.86 tree identifier:** A name identifying a local tree.
- 3.6.87 tree leaf:** A TTCN statement in a behaviour tree or Test Step which has no specified subsequent behaviour.
- 3.6.88 tree node:** A single TTCN statement.
- 3.6.89 tree notation:** The notation used in TTCN to represent Test Cases as trees.
- 3.6.90 TTCN statement:** An event, a pseudo-event or construct which is specified in a behaviour description.
- 3.6.91 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.
- 3.6.92 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 9646-1:1994, 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
UTCF :	upper tester control function
LTCF :	lower tester control function

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 9646-2:1994, clause 4 apply:

CS:	coordinated single-layer (test method)
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
CM :	coordination message
CP :	coordination point
FDT :	formal description technique
FIFO :	first in first out
MTC :	main test component
OSI :	open systems interconnection
PDU :	protocol data unit
PTC :	parallel test component
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 graphical form (TTCN.GR) suitable for human readability;
- 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 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 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, on the one hand, and the static and operational semantics, on the other, as described by the text and as described by Annex A, then

- a) except for the differences specified in 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 A.4 and in the static semantics comments (marked STATIC SEMANTICS) on the syntax productions in A.3 specify restrictions on what is valid TTCN, restricting what is allowed according to the syntax productions;
- c) similarly, the operational semantics restrictions specified in the operational semantics comments (marked OPERATIONAL SEMANTICS) on the syntax productions in A.3 specify restrictions on what is valid TTCN at run-time, restricting what is allowed according to the syntax productions;
- d) the static and operational 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 9646-1:1994, 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 9 through 16 and A.4.

6.3 ATSS that comply with the requirements of TTCN.MP shall satisfy the TTCN.MP syntax requirements stated in 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 16 and annex A and have operational semantics in accordance with the definition of the operational semantics in annex B together with the operational semantics restrictions specified in A.3, 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) meet the additional operational semantics requirements specified in A.3;
- c) 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.

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
abc xyz	abc followed by xyz
	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 substituted 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 table:

- 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			Title
Object Name : Group : (Optional way of grouping together related objects) : : Comments : This entire comment line is optional.			Header
Object Name	... Other Columns ...	Comment	Body
		This column is optional	
Detailed Comments: This footer is optional.			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. A second item, named *Group*, may be used to provide an identifier to group together related objects in the same category. This item is optional. The last item, named *Comments* contains an informal description of the object. This item is optional.

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:

Title of Table		
Group: (Optional way of grouping together related sets of objects)		
Collective Comment: <i>A comment valid for the below defined/declared objects. This comment has a scope reaching to next Collective Comment or until the end of this table.</i>		
Object Name	... Other Columns ...	Comments
Collective Comment: <i>A comment valid for the below defined/declared objects. This comment has a scope reaching to next Collective Comment or until the end of this table.</i>		
Object Name	... Other Columns ...	Comments
Detailed Comments:		

Figure 2 - Generalized layout of a multiple declaration table

The optional *Collective Comments* may be used preceding a group of related objects declared in a multiple object table, both to indicate the grouping and to give a comment that applies to each member of the group or the group as a whole.

This type of table has only a minimal optional header section, which may contain a *Group* identifier and a *Collective Comment*. 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 E.

7.3.5 Specification of proformas

This part of ISO/IEC 9646 specifies numerous 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 Concurrency in TTCN

8.1 Test Components

TTCN allows the specification of test components which may be executed concurrently. This clause gives an overview of the additional proformas and mechanisms available in concurrent TTCN. These proformas and mechanisms shall not be used in ATSS that do not use concurrency (*i.e.* the use of concurrency is optional).

A tester consists of a Main Test Component (MTC) and zero or more Parallel Test Components (PTCs). In non-concurrent TTCN it is not necessary to declare the Main Test Component since there is only one test component and the default is that it is the Main Test Component.

Test components are declared in the Test Component Declarations table. A test component may communicate with the IUT via one or more Points of Control and Observation (PCOs). Test components may communicate with each other by exchanging Coordination Messages (CMs) through Coordination Points (CPs). PTCs may also communicate with the MTC implicitly, by means of assignments to the global result variable and by the MTC being able to check whether or not one or more PTCs have terminated execution. The Test Component Configuration Declarations tables are used to specify (abstract) configurations of test components. These declarations (one for each configuration) show which PCOs and CPs are used, if any, by the test components. CMs are specified in a manner very similar to the method used to specify ASPs. ASN.1 may be used for CM specification. CM constraints are also very similar to ASP constraints. Special proformas are provided for the definition of CM Types and the declaration of CM constraints. CMs are sent and received using the normal TTCN SEND and RECEIVE statements.

In summary, if concurrent TTCN is used the following proformas shall be used:

- a) Test Component Declarations;
- b) Test Component Configuration Declarations.

In addition, if concurrent TTCN is used the following proformas may be used:

- c) CP Declarations;
- d) CM Type Definitions and/or ASN.1 CM Type Definitions, provided that CP declarations are used;
- e) CM Constraints Declarations, provided that CM Type Definitions are used;
- f) ASN.1 CM Constraint Declarations, provided that ASN.1 CM Type Definitions are used.

8.2 Test Component Configurations

Some possible configurations of test components are shown in Figure 3 and Figure 4. In a realization of these abstract configurations, test components may reside in a single machine or be distributed over several machines.

It is possible to use different PTC configurations in different test cases of an Abstract Test Suite. Each Abstract Test Case which uses concurrency shall use one of the declared Test Component Configurations.

Note the following valid but unusual cases:

- a) a PTC need not have any PCOs;
- b) a PTC need not have a CP to an MTC. In such cases the only interaction between the PTC and the MTC will be the creation of the PTC and the implicit result reports from the PTC, i.e., the MTC has no explicit control over the PTC after creation;
- c) two PTCs may be connected by more than one CP;
- d) a test case whose test component configuration refers to a PTC need not contain any CREATE statement to start this PTC;
- e) a test case whose test component configuration refers to a CP need not contain any SEND or RECEIVE statements using this CP.

Items a), b) and c) are illustrated in Figure 3 and Figure 4.

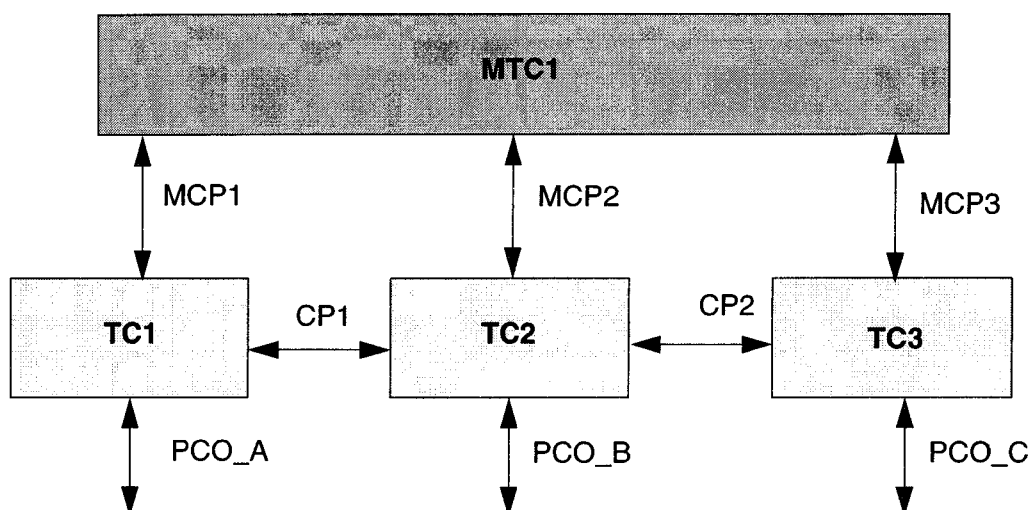


Figure 3 - Example Test Component Configuration CONFIG1

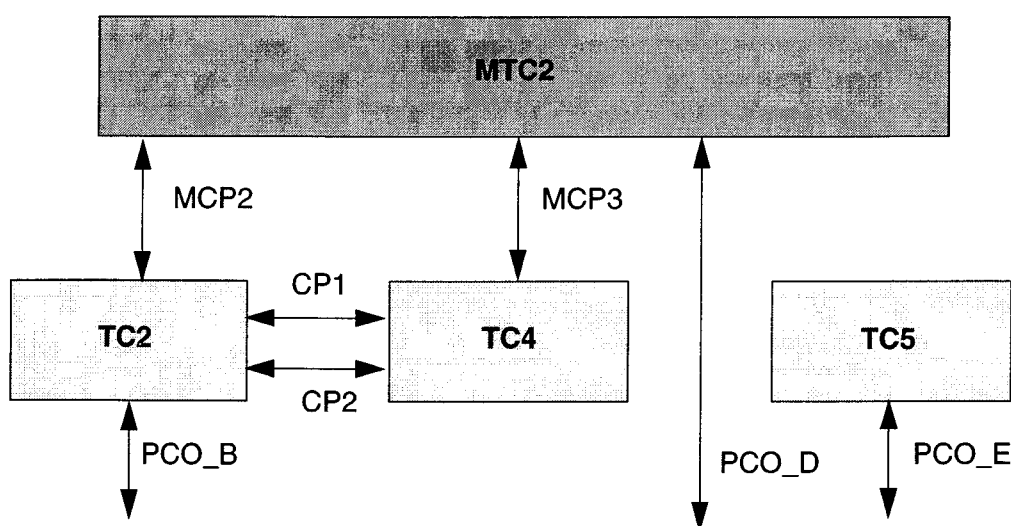


Figure 4 - Example Test Component Configuration CONFIG2

9 TTCN test suite structure

9.1 Introduction

TTCN allows a test suite to be hierarchically structured in accordance with ISO/IEC 9646-1:1994, 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.

9.2 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 9646-1:1994, 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:

626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

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

9.3 Test Step Group References

9.3.1 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.

9.3.2 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:

641 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}

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

9.4 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:

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

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

9.5 Parts 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 10),

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) Import Part (see 10.7),

which contains the declarations of the objects used in the test suite or module that are imported from a source object.

c) Declarations Part (see clause 11),

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);

d) Constraints Part (see clause 12, 13, 14),

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.

e) Dynamic Part (see clause 15),

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).

10 Test Suite Overview

10.1 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 10.2);
- b) Test Case Index (see 10.3);
- c) Test Step Index (see 10.4);
- d) Default Index (see 10.5);
- e) Test Suite Exports (see 10.6).

10.2 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 9646-2:1994, clause 14.1 and ISO/IEC 9646-7:1995, annex E);
- 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 its version number or how it has been derived; this information 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;

imported test cases may be included under any group, independently under which group they are defined in the original source object. A new group may be listed that does not occur in the Dynamic Part. This group shall only contain imported test cases;

the groups of the Dynamic Part shall occur in the same order as they appear there, but the list may be preceded, interrupted or followed by new groups of imported test cases. For these new groups the page number shall not be supplied;

the Selection Ref column may contain the identifier of a selection expression applicable to the new test groups. The new selection expression shall override the specified selection expression in the original test group (if there is any). The absence of the selection expression identifier in this column indicates that the specified selection expression in the original test group is omitted (if there is any);

the Test Group Objective column may contain a new informal statement of the objective of the new test group. This new objective shall override the objective in the imported test group (if any). The absence of the test group objective in this column indicates that the specified test group objective is omitted;

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 sub-groups 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>Free Text</i>			
PICS Ref : <i>Free Text</i>			
PIXIT Ref : <i>Free Text</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:

```
41 SuiteIdentifier ::= Identifier
626 TestGroupReference ::= [SuiteIdentifier "/" ] {TestGroupIdentifier "/" }
202 SelectExprIdentifier ::= Identifier
741 Number ::= (NonZeroNum {Num}) | 0
```

10.3 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,
- c) 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;
- d) 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;
- e) a description of the Test Case,
which is possibly a shortened form of the test purpose;
- f) 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 folloing 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

Collective comments may be used in this table according to Figure 2.

SYNTAX DEFINITION:

626 TestGroupReference ::= [SuiteIdentifier "/"] {TestGroupIdentifier "/"}

624 TestCaseIdentifier ::= Identifier

202 SelectExprIdentifier ::= Identifier

The complete list of test cases shall include the imported test cases. Explicitly defined Test Cases shall be listed in the order in which they exist in the ATS. Page numbers shall not be supplied for imported test cases.

The Selection Ref column has similar semantic as the one given in the previous section (10.2).

The Description column may contain a new shortened form of the Test Purpose. This new description shall override the description in the imported test case (if any). The absence of the description in this column indicates that the specified description is omitted.

10.4 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 Step Id	Description	Page Nr
. . TestStepGroupReference . .	. . TestStep- Identifier . .	. . FreeText . .	. . Number . .
Detailed Comments: [FreeText]			

Proforma 3 - Test Step Index

Collective comments may be used in this table according to Figure 2.

SYNTAX DEFINITION:

641 TestStepGroupReference ::= [SuiteIdentifier "/"] {TestStepGroupIdentifier "/"}

639 TestStepIdentifier ::= Identifier

The complete list of test steps shall include the imported test steps. Explicitly defined Test Steps shall be listed in the order in which they exist in the ATS. Page numbers shall not be supplied for imported test steps.

The Description column may contain a new shortened form of the Test Step Objective. This new description shall override the description in the imported test step (if any). The absence of the description in this column indicates that the specified description is omitted.

10.5 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	Default Id	Description	Page Nr
.	.	.	.
.	.	.	.
<i>DefaultGroupReference</i>	<i>Default- Identifier</i>	<i>FreeText</i>	<i>Number</i>
.	.	.	.
.	.	.	.
Detailed Comments: [FreeText]			

Proforma 4 - Default Index

Collective comments may be used in this table according to Figure 2.

SYNTAX DEFINITION:

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

650 DefaultIdentifier ::= Identifier

The complete list of defaults shall include the imported defaults. Explicitly defined Defaults shall be listed in the order in which they exist in the ATS. Page numbers shall not be supplied for imported defaults.

The Description column may contain a new shortened form of the Default Objective. This new description shall override the description in the imported default (if any). The absence of the description in this column indicates that the specified description is omitted.

10.6 Test Suite Exports

The Test Suite Exports table may be used to specify explicitly which objects in the test suite are designed to be re-usable and hence may be imported into other test suites or TTCN modules.

The Test Suite Exports proforma is used to identify the objects which may be exported.

If a PCO type is given as an exported object in the Export table, it shall be defined in the optional PCO Type table.

The name of the original source object shall be given if the object is itself imported.

If the object is declared as an external object (explicit external) or is an object which is omitted in the imported source object (implicit external), the keyword EXTERNAL is given instead of the source object name.

Exporting an object of type Enumeration or Named Number requires that the corresponding type is given. The other objects which are defined in the corresponding type are not exported as well. They are however implicitly exported and can be referred in other exported objects. The type name is given as a suffix to the object name embedded in brackets.

The following information shall be supplied in the Test Suite Exports table for each of the exported objects:

a) the name of the object

If the object is of type NamedNumber or Enumeration the corresponding type shall be given as a suffix to the object name embedded in brackets.

b) the object type

c) the name of the original source object if the object is imported, or the object directive EXTERNAL

d) a page number

providing the location of the object in the test suite (no page number shall be given for imported objects)

e) an optional comment

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

Test Suite Exports				
Object Name	Object Type	Source Name	Page Nr	Comments
. . <i>ObjectIdentifier</i> . .	. . <i>TTCN_ObjectType</i> .	. . <i>[SourceIdentifier ObjectDirective]</i> .	. . <i>[Number]</i> .	<i>[FreeText]</i>
Detailed Comments: <i>[FreeText]</i>				

Proforma 5 - Test Suite Exports

SYNTAX DEFINITION:

- 12 ObjectIdentifier ::= Identifier | ObjectTypeReference
- 15 TTCN_ObjectType ::= SimpleType_Object | StructType_Object | ASN1_Type_Object | TS_Op_Object | TS_Proc_Object | TS_Par_Object | SelectExpr_Object | TS_Const_Object | TS_Var_Object | TC_Var_Object | PCO_Type_Object | PCO_Object | CP_Object | Timer_Object | TComp_Object | TCompConfig_Object | TTCN_ASP_Type_Object | ASN1_ASP_Type_Object | TTCN_PDU_Type_Object | ASN1_PDU_Type_Object | TTCN_CM_Type_Object | ASN1_CM_Type_Object | EncodingRule_Object | EncodingVariation_Object | InvalidFieldEncoding_Object | Alias_Object | StructTypeConstraint_Object | ASN1_TypeConstraint_Object | TTCN_ASP_Constraint_Object | ASN1_ASP_Constraint_Object | TTCN_PDU_Constraint_Object | ASN1_PDU_Constraint_Object | TTCN_CM_Constraint_Object | ASN1_CM_Constraint_Object | TestCase_Object | TestStep_Object | Default_Object | NamedNumber_Object | Enumeration_Object
- 17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier
- 18 ObjectDirective ::= Omit | EXTERNAL

EXAMPLE 6 - Test Suite Exports

Test Suite Exports				
Object Name	Object Type	Source Name	Page Nr	Comments
String5	SimpleTypeDef	Module_B	3	
wait	TimerDcl			
INTC	TTCN_PDU_Type	TestSuite_1	13	
DEF1	Default			
TC_2	TestCase	TestSuite_2	33	
TC_3	TestCase			
Preamble	TestStep	EXTERNAL		
Detailed Comments:				

10.7 The Import Part**10.7.1 Introduction**

The purpose of the Import Part is to declare the objects used in the test suite that are imported from a source object. The effect of the imports is equivalent to having a copy of the imported objects within the test suite.

An object may be imported only if it is exported by a source object. A test suite without an export table exports all objects which have a global name. A module and a test suite with at least one export table export the objects contained in the export tables. An object which is not itself explicitly imported is implicitly imported if it is referenced by an imported object.

10.7.2 Imports

The Imports table identifies the source object and provides information on the overall objective of the source object. The following information shall be supplied in the Imports table:

- a) the name of the source object;
- b) a description of the objective of the source object;
- c) a full reference to the source object; which should contain a document identifier and other information, such as version and date;
- d) other information which may aid understanding of the source object, this should be included as a comment;
- e) a list of the objects from the imported source object; for each object the following information shall be provided:
 - 1) the name of the object as used in the source object;
 - 2) the type of the object; which shall be the same as the type given in the source object;
 - 3) the name of the original source object if the object is imported from another source object, the object directive OMIT or “-” if the object is to be omitted from the set of objects imported from the source object, or the object directive EXTERNAL if the object is declared as external in the source object.

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

Imports			
Source Name : <i>SourceIdentifier</i>			
Group : <i>[ImportsGroupReference]</i>			
Standards Ref: <i>[FreeText]</i>			
Comments : <i>[FreeText]</i>			
Object Name	Object Type	Source Name	Comments
<i>.</i> <i>ObjectIdentifier</i> <i>.</i> <i>.</i>	<i>.</i> <i>TTCN_ObjectType</i> <i>.</i>	<i>.</i> <i>[SourceIdentifier /</i> <i>ObjectDirective]</i> <i>.</i>	<i>[FreeText]</i>
Detailed Comments: <i>[FreeText]</i>			

Proforma 6 - Imports

SYNTAX DEFINITION:

- 17 SourceIdentifier ::= SuiteIdentifier | TTCN_ModuleIdentifier
- 34 ImportsGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ImportsGroupIdentifier "/" }
- 12 ObjectIdentifier ::= Identifier | ObjectTypeReference
- 15 TTCN_ObjectType ::= SimpleType_Object | StructType_Object | ASN1_Type_Object | TS_Op_Object | TS_Proc_Object | TS_Par_Object | SelectExpr_Object | TS_Const_Object | TS_Var_Object | TC_Var_Object | PCO_Type_Object | PCO_Object | CP_Object | Timer_Object | TComp_Object | TCompConfig_Object | TTCN_ASP_Type_Object | ASN1_ASP_Type_Object | TTCN_PDU_Type_Object | ASN1_PDU_Type_Object | TTCN_CM_Type_Object | ASN1_CM_Type_Object | EncodingRule_Object | EncodingVariation_Object | InvalidFieldEncoding_Object | Alias_Object | StructTypeConstraint_Object | ASN1_TypeConstraint_Object | TTCN_ASP_Constraint_Object | ASN1_ASP_Constraint_Object | TTCN_PDU_Constraint_Object | ASN1_PDU_Constraint_Object | TTCN_CM_Constraint_Object | ASN1_CM_Constraint_Object | TestCase_Object | TestStep_Object | Default_Object | NamedNumber_Object | Enumeration_Object
- 18 ObjectDirective ::= Omit | EXTERNAL

EXAMPLE 7 - An Imports table

Imports			
Test Case Name : Module A			
Source Ref : {ISO standard 1234}			
Standards Ref : ISO 300 313			
Comments : Layer 2 Test Suite			
Object Name	Object Type	Source Name	Comments
String5	SimpleTypeDef	ModuleB EXTERNAL	1)
Wait	TimerDcl		2)
R1_POSTAMBLE	TestStep		3)
TSAP	PCO_TypeDcl	OMIT	4)
blue[ColorEnum]	Enumeration		
a[NN_type1]	NamedNumber		
Detailed Comments:			
1) The original source of this timer is ModuleB			
2) This test step is declared as external in ModuleA and must be explicitly defined or imported where this module is used.			
3) TSAP must be defined in the PCO Type Dcl table.			
4) This Named Number is omitted from the imports and hence should be redefined explicitly in the test suite.			

11 Declarations Part

11.1 Introduction

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

- a) definitions:
 - 1) Test Suite Types (see 11.2.3);
 - 2) Test Suite operations (see 11.3.4);
- b) parameterization and selection of Test Cases:
 - 1) Test Suite Parameters (see 11.4);
 - 2) Test Case Selection Expressions (see 11.5);
- c) declarations/definitions:
 - 1) Test Suite Constants (see 11.6 and 11.7);
 - 2) Test Suite Variables (see 11.8.1);
 - 3) Test Case Variables (see 11.8.3);
 - 4) PCO types (see 11.9);
 - 5) PCO s (see 11.10);
 - 6) CPs (see 11.11);
 - 7) Timers (see 11.12);
 - 8) Test Components (see 11.13.1);
 - 9) Test Component Configurations (see 11.13.2);
 - 10) ASP types (see 11.14);
 - 11) PDU types (see 11.15);
 - 12) Encoding Rules (see 219);
 - 13) Encoding Variations (see 11.16.2);
 - 14) Invalid Field Encodings (see 11.16.3);
 - 15) CM types (see 11.17);
 - 16) Aliases (see 11.21).

11.2 TTCN types

11.2.1 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.

TTCN is a weakly typed language, in that values of any two types which have the same base type are considered to be type compatible (e.g. for the purposes of performing assignments or parameter passing).

11.2.2 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 8 - '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 9 - '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 10 - 'FF96'O

f) **OBJECTIDENTIFIER predefined type:** a type whose distinguished values are the set of all object identifiers allocated in accordance with the rules of ISO/IEC 8824-1: 1994.

g) **R_TYPE predefined type:** a type consisting of the following distinguished values:

pass, fail, inconc and none

These values are predefined identifiers and as such, are case sensitive. This predefined type is for use with verdicts, see 15.17.

h) **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 8824-1: 1994.

Table 2 - Predefined CharacterString Type

<i>NumericString</i>
<i>PrintableString</i>
<i>TeletexString</i>
<i>T61String</i>
<i>VideotexString</i>
<i>VisibleString</i>
<i>ISO646String</i>
<i>IA5String</i>
<i>GraphicString</i>
<i>GeneralString</i>
<i>BMPString</i>
<i>UniversalString</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:

```

735 PredefinedType ::= INTEGER | BOOLEAN | BITSTRING | HEXSTRING | OCTETSTRING | OBJECTIDENTIFIER |
    R_Type | CharacterString
736 CharacterString ::= NumericString | PrintableString | TeletexString | VideotexString | VisibleString | IA5String |
    GraphicString | GeneralString | T61String | ISO646String
741 Number ::= (NonZeroNum {Num}) | 0
742 NonZeroNum ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
743 Num ::= 0 | NonZeroNum
744 BooleanValue ::= TRUE | FALSE
745 Bstring ::= "" {Bin | Wildcard} "" B
746 Bin ::= 0 | 1
747 Hstring ::= "" {Hex | Wildcard} "" H
748 Hex ::= Num | A | B | C | D | E | F
749 Ostring ::= "" {Oct | Wildcard} "" O
750 Oct ::= Hex Hex
751 Cstring ::= "" {Char | Wildcard | "\"} ""
752 Char ::= /* REFERENCE - A character defined by the relevant CharacterString type. */
753 Wildcard ::= AnyOne | AnyOrNone
754 AnyOne ::= "?"
755 AnyOrNone ::= "*"

```

11.2.3 Test Suite Type Definitions

11.2.3.1 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).

11.2.3.2 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 5 in 11.18; only non-negative INTEGER literals or the keyword INFINITY for the upper bound shall be used;
- c) optionally, a specific encoding identifier followed by any necessary actual parameter list, in order to specify an explicit encoding for the simple type, which overrides the encoding rules and encoding variations applicable to any PDU in which that simple type is used; the encoding identifier, if any, shall identify either one of the Encoding Variations or an Invalid Field Encoding Definition defined in the test suite (e.g., LD(10)); see 11.16.4.

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

Simple Type Definitions			
Group : <i>[SimpleTypeGroupReference]</i>			
Type Name	Type Definition	Type Encoding	Comments
<i>SimpleTypeIdentifier</i>	<i>Type&Restriction</i>	<i>[PDU_FieldEncodingCall]</i>	<i>[FreeText]</i>
Detailed Comments: <i>[FreeText]</i>			

Proforma 7 - Simple Type Definitions

Collective comments may be used in this table according to Figure 2.

SYNTAX DEFINITION:

- 75 SimpleTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {SimpleTypeGroupIdentifier "/" }
- 79 SimpleTypeIdentifier ::= Identifier
- 81 Type&Restriction ::= Type [Restriction]
- 515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall

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.

Values of any two simple types which have the same base type are considered to be type compatible (e.g. for the purposes of performing assignments or parameter passing).

EXAMPLE 11 - Simple Test Suite Type definition

Simple Type Definition		
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

11.2.3.3 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:

- 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 Encoding Variations to be used for structures of this type within a PDU;

In order to specify explicit Encoding Variations for entire structured types, which override the Encoding Variations applicable to any PDU in which this structured type is used, this optional entry shall reference an entry in the relevant Encoding Variations table (e.g., to change from SD to LD(3)). If this entry is not used, then the applicable Encoding Variations are those applicable to each PDU within which this structured type is used. See 11.16.4.

c) 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 11.18);

3) optionally, a specific encoding identifier followed by any necessary actual parameter list, in order to specify an explicit encoding for the structured type, which overrides the encoding rules and encoding variations applicable to any PDU in which that structured type is used; the encoding identifier, if any, shall identify either one of the Encoding Variations or an Invalid Field Encoding Definition defined in the test suite (e.g., LD(10)); see 11.16.4.

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			
Type Name : <i>StructId&FullId</i>			
Group : <i>[StructTypeGroupReference]</i>			
Encoding Variation : <i>[EncVariationCall]</i>			
Comments : <i>[FreeText]</i>			
Element Name	Type Definition	Field Encoding	Comments
<i>ElemId&FullId</i>	<i>Type&Attributes</i>	<i>[PDU_FieldEncodingCall]</i>	<i>[FreeText]</i>
Detailed Comments: <i>[FreeText]</i>			

Proforma 8 - Structured Type Definition

SYNTAX DEFINITION:

- 97
- StructId&FullId ::= StructIdentifier [FullIdentifier]
- 101
- StructTypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {StructTypeGroupIdentifier "/" }
- 511
- EncVariationCall ::= EncVariationIdentifier [ActualParList]

106 ElemId&FullId ::= ElemIdentifier [FullIdentifier]
 396 Type&Attributes ::= (Type [LengthAttribute]) | **PDU**
 515 PDU_FieldEncodingCall ::= EncVariationCall | InvalidFieldEncodingCall

11.2.3.4 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 8824-1: 1994. 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 Encoding Variations to be used for structures of this type within a PDU;

In order to specify explicit Encoding Variations for entire ASN1_Types, which override the Encoding Variations applicable to any PDU in which this ASN1_Type is used, this optional entry shall reference an entry in the relevant Encoding Variations table (e.g., to change from SD to LD(3)). If this entry is not used, then the applicable Encoding Variations are those applicable to each PDU within which this ASN1_Type is used. See 11.16.4.

c) the ASN.1 type definition,

which shall follow the syntax defined in ISO/IEC 8824-1: 1994, except that there is the additional option of specifying an Encoding Variation or Invalid Field Encoding associated with either the whole ASN1_Type or any ASN.1 Type within the ASN1_Type. This is done by giving a specific encoding identifier followed by any necessary actual parameter list, in order to specify explicit encodings for individual fields or other subtypes of a PDU, which override the encoding rules and encoding variations applicable to the PDU as a whole; the encoding identifier, if any, shall identify either one of the Encoding Variations or an Invalid Field Encoding Definition defined in the test suite (e.g., LD(10)); see 11.16.4.

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 8824-1: 1994. ASN.1 comments can be used within the table body. The comments column shall not be present in this table.

Comments in ASN.1 start with "--" and end with either the next occurrence of "--" or with "end of line", whichever comes first. This prevents a single ASN.1 comment from spanning several lines. "End of line" is not, however, a defined symbol in TTCN.MP. ATS specifiers are recommended to facilitate the exchange of ATSs in TTCN.MP by always closing ASN.1 comments with "--".

This information shall be provided in the following proforma:

ASN.1 Type Definition	
Type Name	: <i>ASN1_TypeId&FullId</i>
Group	: <i>[ASN1_TypeGroupReference]</i>
Encoding Variation	: <i>[EncVariationCall]</i>
Comments	: <i>[FreeText]</i>
Type Definition	
<i>ASN1_Type&LocalTypes</i>	
Detailed Comments: <i>[FreeText]</i>	

Proforma 9 - ASN.1 Type Definition

SYNTAX DEFINITION:

115 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
118 ASN1_TypeGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {ASN1_TypeGroupIdentifier "/" }
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]
121 ASN1_Type&LocalTypes ::= ASN1_Type {ASN1_LocalType}

EXAMPLE 12 - An ASN.1 Test Suite Type definition

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

11.2.3.5 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 8824-1: 1994;
 - c) the module identifier,
which consists of a module reference that shall follow the identifier rules stated in ISO/IEC 8824-1: 1994, and an optional ObjectIdentifier; the module shall be unique within the domain of interest;
 - d) the Encoding Variations to be used for such ASN1_Types within a PDU;
- In order to specify explicit Encoding Variations for entire ASN1_Types, which override the Encoding Variations applicable to any PDU in which this ASN1_Type is used, this optional entry shall reference an entry in the relevant Encoding Variations table (e.g., to change from SD to LD(3)). If this entry is not used, then the applicable Encoding Variations are those applicable to each PDU within which this ASN1_Type is used. See 11.16.4.

This information shall be provided in the following proforma:

ASN.1 Type Definitions By Reference				
Group : [ASN1_TypeGroupReference]				
Type Name	Type Reference	Module Identifier	Encoding Variation	Comments
ASN1_TypeId- &FullId	TypeReference	ASN1_ModuleIdentifier	[EncVariationCall]	[FreeText]
Detailed Comments: [FreeText]				

Proforma 10 - ASN.1 Type Definitions By Reference

Collective comments may be used in this table according to Figure 2.

SYNTAX DEFINITION:

```

118 ASN1_TypeGroupReference ::= [(SuitIdentifier | TTCN_ModuleIdentifier) "/" ] {ASN1_TypeGroupIdentifier "/" }
115 ASN1_TypeId&FullId ::= ASN1_TypeIdentifier [FullIdentifier]
131 TypeReference ::= typereference
133 ASN1_ModuleIdentifier ::= ModuleIdentifier
511 EncVariationCall ::= EncVariationIdentifier [ActualParList]

```

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 8824-1: 1994, 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 13 - 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	Type-1	module-1	
Sub_Type_1	Sub-Type-1	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,	
		field2 BIT STRING {first_bit(0), second_bit(1) } }	

11.3 TTCN operators and TTCN operations

11.3.1 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.

11.3.2 TTCN operators

11.3.2.1 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.

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.

Table 3 - Precedence of Operators

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

SYNTAX DEFINITION:

```

723 AddOp ::= "+" | "-" | OR
724 MultiplyOp ::= "*" | "/" | MOD | AND
725 UnaryOp ::= "+" | "-" | NOT
726 RelOp ::= "=" | "<" | ">" | "<>" | ">=" | "<="

```

11.3.2.2 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.

11.3.2.3 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 12.6.2.

11.3.2.4 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.

11.3.3 Predefined operations

11.3.3.1 Introduction

11.3.3.1.1 The predefined operations fall into two categories:

- a) conversion;
- b) others

11.3.3.1.2 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) ⇒ RESULT_TYPE

11.3.3.2 Predefined conversion operations

11.3.3.2.1 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.

11.3.3.2.2 HEX_TO_INT(hexvalue:HEXSTRING) ⇒ 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.

11.3.3.2.3 BIT_TO_INT(bitvalue:BITSTRING) ⇒ 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.

11.3.3.2.4 INT_TO_HEX(intvalue, length:INTEGER) ⇒ 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.

11.3.3.2.5 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.

11.3.3.3 Other predefined operations

TTCN also defines the following predefined operations:

- a) IS_PRESENT;
- b) IS_CHOSEN;
- c) NUMBER_OF_ELEMENTS;
- d) LENGTH_OF;
- e) SIZE_OF.

11.3.3.3.1 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 15.10.2.

EXAMPLE 14 - 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.

11.3.3.3.2 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 those of ASN.1 type CHOICE. The argument of the operation shall have the format as defined in 15.10.2.

EXAMPLE 15 - 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.

11.3.3.3.3 NUMBER_OF_ELEMENTS(Value) ⇒ INTEGER

The operation returns the actual number of elements of a value that is of type ASN.1 SEQUENCE OF or SET OF. Its result is fully compatible with that of the equivalent ASN.1 SIZE constraint applied to objects of these types. The operation shall not be applied to values other than of ASN.1 type SEQUENCE OF or SET OF. The argument of the operation shall have the format as defined in 15.10.2.

EXAMPLE 16 - 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.

Also, NUMBER_OF_ELEMENTS ({3, 0, 5}) returns 3.

11.3.3.3.4 LENGTH_OF(Value) ⇒ INTEGER

The operation returns the actual length of a value that is of type BITSTRING, HEXSTRING, OCTETSTRING, or CharacterString or of ASN.1 type BIT STRING or OCTET STRING. The units of length for each string type are defined in Table 5 in 11.18.2.

NOTE - These units of length are compatible with those used in ASN.1 SIZE constraints for objects of ASN.1 types, but not for literal values which in this context in TTCN are considered to be of the corresponding TTCN type. Thus, an hstring such as 'F3'H which could in ASN.1 be of type BIT STRING or OCTET STRING, will be interpreted as the TTCN type HEXSTRING.

The argument of the operation shall have the format as defined in 15.10.2.

The operation shall not be applied to values other than of type BITSTRING, HEXSTRING, OCTETSTRING, or CharacterString, or of ASN.1 type BIT STRING or OCTET STRING.

EXAMPLE 17 - Use of LENGTH_OF

If S is of type BITSTRING or ASN.1 type BIT STRING and = '010'B then LENGTH_OF(S) returns 3

If S is of type HEXSTRING and = 'F3'H then LENGTH_OF(S) returns 2

If S is of type OCTETSTRING and = 'F2'O then LENGTH_OF(S) returns 1

If S is of a CharacterString type and = "EXAMPLE" then LENGTH_OF(S) returns 7

If S is of ASN.1 type BIT STRING and = 'F3'H then LENGTH_OF(S) returns 8

If S is of ASN.1 type OCTET STRING and = 'F3'H then LENGTH_OF(S) returns 1

If S is of ASN.1 type OCTET STRING and = '01010011'B then LENGTH_OF(S) returns 1

Also, LENGTH_OF (INT_TO_HEX (26, 4)) returns 4

LENGTH_OF ('F3'H) returns 2

and, LENGTH_OF ("Length_of Example") returns 17

11.3.4 Test Suite Operation definitions and descriptions**11.3.4.1 Introduction**

Operations specific to a test suite may be defined by the ATS specifier. To define a new operation, the following 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. All parameters shall be passed by value, meaning that in evaluating a call of a test suite operation, the actual parameters are assigned to the corresponding formal parameters, as if in an assignment statement.

EXAMPLE 18 - 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 definition of the operation,
which shall consist of one of the following:
- 1) a procedural definition, which when evaluated results in the evaluation of a RETURNVALUE statement to provide the result of the operation, including explanatory comments embedded within the procedural definition at appropriate places as text delimited by “/*” and “*/”, or
 - 2) a description of the operation in text, possibly including a reference to a publicly available specification of the algorithm to be applied when the operation is invoked, plus at least one example showing an invocation and corresponding result; the explanation should 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;
- e) optionally, further comment describing the operation, provided either in the Comments part of the table header or in the Detailed Comments area of the table.

The use of procedural definitions is recommended in order to provide precision in the definition of the operations, but a textual explanation is allowed as an alternative for backwards compatibility.

In the case of a procedural definition, this information shall be provided in the format shown in the following proforma:

Test Suite Operation Procedural Definition	
Operation Name :	TS_ProcId&ParList
Group :	[TS_ProcGroupReference]
Result Type :	Type
Comments :	[FreeText]
Definition	
TS_OpProcDef	
Detailed Comments:	[FreeText]

Proforma 11 - Test Suite Operation Procedural Definition

SYNTAX DEFINITION:

155 TS_ProcId&ParList ::= TS_ProcIdentifier [FormalParList]
 158 TS_ProcGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_ProcGroupIdentifier "/" }
 734 Type ::= PredefinedType | ReferenceType
 162 TS_OpProcDef ::= [VarBlock] ProcStatement

In the case of a textual description, this information shall be provided in the following proforma:

Test Suite Operation Description	
Operation Name :	<i>TS_OpId&ParList</i>
Group :	<i>[TS_OpGroupReference]</i>
Result Type :	<i>Type</i>
Comments :	<i>[FreeText]</i>
Description	
<i>FreeText</i>	
Detailed Comments: <i>[FreeText]</i>	

Proforma 12 - Test Suite Operation Description**SYNTAX DEFINITION:**

141 TS_OpId&ParList ::= TS_OpIdentifier [FormalParList]
 144 TS_OpGroupReference ::= [(SuiteIdentifier | TTCN_ModuleIdentifier) "/"] {TS_OpGroupIdentifier "/" }
 734 Type ::= PredefinedType | ReferenceType

11.3.4.2 Parameters

A test suite operation may be compared to a function in an ordinary programming language. Values shall only be passed into the operation by formal parameters. Each formal parameter shall be declared to be a Predefined Type, a Test Suite Type Identifier, ASP Type Identifier, PDU Type Identifier, CM Type Identifier or the meta-type **PDU**. Test suite variables, test case variables, test suite constants, test suite parameters and constraints shall not directly be used within the procedural definition of a test suite operation, but if required in the test suite operation shall be passed as actual parameters.

There shall be no side-effects, that is, the parameters to the operation shall not be altered as a result of any call of the operation. Predefined operations and other test suite operations may be used within the procedural definition of a test suite operation, without having to be passed as actual parameters.

When a Test Suite Operation is invoked

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

11.3.4.3 Variables and Identifiers

If a procedural definition is used, it may include the declaration of local variables, placed at the head of the procedural definition, between the keywords **VAR** and **ENDVAR**. These variables may be of any type allowed in TTCN. The scope of these local variables is the procedural definition itself. These declarations declare lists of variable identifiers, each of a given type and each list may either be declared to be **STATIC** or not. Variables, both **STATIC** and those not declared as **STATIC**, may be given an optional initial value.

NOTE - It is recommended always to provide **STATIC** variables with an initial value.

The variables which are not declared to be **STATIC** are initialized every time the operation is invoked, with the specified initial value, if any, and thus they shall not convey a value from one evaluation of the test suite operation to another. Those which are declared to be **STATIC** are initialized with the specified initial value, if any, the first time the operation is invoked within a given test component, or within a given test case if test components are not used, and thereafter they retain their values from one invocation to the next within that test component or test case.

Variables which are not assigned an initial value are considered to be unbound and shall be explicitly bound to a value by an assignment in the operation body before being used in an expression. If an unbound variable is used in an expression then it is a test case error.

Each identifier used in the procedural definition of a test suite operation shall be one of the following:

- a) locally declared variable name;
- b) a type name, used in a variable declaration;
- c) a formal parameter name declared in a formal parameter list of the operation;
- d) a test suite operation name.

The scope of formal parameter names and locally declared variable names is the procedural definition of the test suite operation.. Thus, the values of all other types of identifier are not directly accessible within the procedural definition of a test suite operation. To access such values they shall be passed as actual parameters to the test suite operation.

11.3.4.4 Procedure Statements

In a procedural definition, following the declaration of local variables, if any, there shall be a procedure statement of one of the following kinds:

- a) a Return statement;
- b) an Assignment statement;
- c) an If statement;
- d) a While loop;
- e) a Case statement;
- f) a block containing a sequence of procedure statements separated by semicolons and all enclosed by the keywords **BEGIN** and **END**.

Comments may be embedded as text within procedural statements, delimited by “/*” and “*/”. Comments shall not be embedded within other comments.

11.3.4.5 ReturnValue statements

Each evaluation of a test suite operation shall end with the evaluation of a ReturnValue statement, consisting of the keyword **RETURNVALUE** followed by an expression. This statement shall return the value of the given expression as the result of the test suite operation. The type of this result shall match the Result Type specified in the header of the test suite operation definition table.

11.3.4.6 Assignment statements

The form of Assignment is the same as in the TTCN behaviour descriptions (see 15.10.4), except that it is not enclosed in parentheses. The DataObjectReference on the left hand side shall begin with a local variable. If the type of the local variable is a structured type then the DataObjectReference may access a component of that structure (using a record reference, array reference or bit reference, as appropriate, see 15.10.2 and 15.10.3).

11.3.4.7 If statements

There are two forms of If statement:

- IF expression **THEN** procedure-statement **ELSE** procedure-statement **ENDIF**
- IF expression **THEN** procedure-statement **ENDIF**

The expression following the keyword **IF** shall be evaluated first and shall evaluate to a Boolean value. If this evaluates to **TRUE** then the procedure statement following the keyword **THEN** shall be evaluated. If the expression evaluates to **FALSE**