INTERNATIONAL STANDARD

ISO/IEC 8882-1

First edition
1993-11-15

Information technology — Telecommunications and information exchange between systems — X.25 DTE conformance testing —

Part 1: General principles

Technologies de l'information — Télécommunications et échange d'information entre systèmes — Test de conformité X.25 DTE —

Partie 1: Principes généraux

Reference number
ISO/IEC 8882-1:1993(E)

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 8882-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Sub-Committee SC 6, *Telecommunications and information exchange between systems.*

ISO/IEC 8882 consists of the following parts, under the general title *Information technology — Telecommunications and information exchange between systems* — X.25 *DTE conformance testing*:

— *Part 1: General principles*

— *Part 2: Data link layer conformance test suite*

— *Part 3: Packet level conformance test suite*

Annex A forms an integral part of ISO/IEC 8882.

Introduction

ISO/IEC 8882 specifies a set of tests to evaluate Data Terminal Equipment (DTE) conformance to International Standards ISO 7776 or ISO/IEC 8208, or both. ISO 7776 and ISO/IEC 8208 allow for a DTE to interface with a Data Circuit-Terminating Equipment (DCE) conforming to CCITT Recommendation X.25 (1980,1984) or to another DTE conforming to ISO 7776 or ISO/IEC 8208 or both. The implementations of ISO 7776 and ISO/IEC 8208 are tested independently.

CCITT Recommendations X.25(1980) and X.25(1984) are written from the perspective of a DCE and therefore do not explicitly specify the DTE operation. However, recommended operation of DTEs is included by implication because of the need to communicate with X.25 DCEs. Tests within ISO/IEC 8882–2 and ISO/IEC 8882–3 pertaining to X.25 (1980, 1984) are based on the DTE operational characteristics implied by CCITT X.25.

This part of ISO/IEC 8882 specifies the framework in which the other parts of ISO/IEC 8882 may be understood and the principles to be applied. The notation used in ISO/IEC 8882–2 and ISO/IEC 8882–3 is TTCN as defined in ISO/IEC 9646–3.

ISO/IEC 8882–2 presents the Data Link Layer aspects for evaluating conformance to ISO 7776 while ISO/IEC 8882–3 presents the Packet Layer aspects for evaluating conformance to ISO/IEC 8208

The conformance tests are designed for use by

— test evaluators (responsible for analysing results and determining whether conformance has been achieved);

— test suite designers or implementors (for determining what tests are required and what results can and should be anticipated by the test device); and

— users implementing ISO 7776 or ISO/IEC 8208 or DTEs interfacing to DCEs that implement CCITT X.25 (1980 or 1984) (for determining the functionality required of their implementations to be considered in conformance).

Information technology — Telecommunications and information exchange between systems — X.25 DTE conformance testing —

Part 1: General principles

1 Scope

ISO/IEC 8882 defines the testing of a DTE operating at the Data Link Layer and at the Packet Layer when accessing, by means of a dedicated path connection, switched or permanent, a public or private packet-switched network conforming to CCITT Recommendation X.25 or another DTE conforming to ISO 7776 and ISO/IEC 8208.

The tests will test the conformance of an implementation by observing its external behaviour. The conformance tests will not test the DTE performance characteristics, the diagnostic and maintenance functions, the correctness of the protocol itself, or DTE internal implementation, or the full capabilities as stated in the PICS.

This part of ISO/IEC 8882

— provides a general introduction;

— refers to those applicable International Standards;

— defines terms applicable to X.25-DTE conformance testing;

— states the test case derivation and description; and

— states the test methodology.

ISO/IEC 8882-1 contains no statement of conformance. Specific statements of conformance are given in ISO/IEC 8882-2 and ISO/IEC 8882-3.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 8882. 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 8882 are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 7498 : 1984, *Information processing systems — Open Systems Interconnection — Basic Reference Model.*

ISO 7776 : 1986, *Information processing systems — Data communications — High-level data link control procedures — Description of the X.25 LAPB-compatible DTE data link procedures.*

ISO/IEC 8208 : 1990, *Information technology — Data communications — X.25 Packet Layer Protocol for Data Terminal Equipment.*

ISO/IEC 8882-2 : 1992, *Information technology — Telecommunications and information exchange between systems — X.25 DTE conformance testing — Part 2: Data link layer conformance test suite.*

ISO/IEC 8882-3 : 1991, *Information technology — Telecommunications and information exchange between systems — X.25 DTE conformance testing — Part 3: Packet layer conformance test suite.*

ISO/IEC 9646-1 : 1991, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts. (See also CCITT Recommendation X.291 (1991)).*

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

ISO/IEC 9646-3 : 1992, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 3: The Tree and Tabular Combined Notation (TTCN).*

CCITT Recommendation X.25 (1980), *Interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks.*

CCITT Recommendation X.25 (1984), *Interface between Data Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) for Terminals Operating in the Packet Mode and Connected to Public Data Networks by Dedicated Circuit.*

3 Definitions

3.1 Reference model definitions

This part of ISO/IEC 8882 makes use of the following term defined in ISO 7498:

(N)-protocol-data-unit (N)-PDU

3.2 Conformance testing definitions

This part of ISO/IEC 8882 makes use of the following terms defined in ISO/IEC 9646-1:

a) Abstract Test Case

b) Conformance Test Suite

c) Conformance Testing

d) Implementation Under Test

e) Inopportune PDU

f) Lower Tester

g) Protocol Implementation Conformance Statement

h) Protocol Implementation eXtra Information for Testing

i) Remote Single Layer Test Method

j) System Under Test

k) Test Group

l) Test Step

m) Test Suite

3.3 X.25 DTE conformance testing definitions

For the purpose of this part of ISO/IEC 8882 the following definitions apply:

3.3.1 improper PDU: The (N)-PDU whose syntax does not conform to the format specifications of ISO 7776 or ISO/IEC 8208 or CCITT X.25.

3.3.2 proper PDU: The (N)-PDU whose syntax conforms to the format specification of CCITT X.25, ISO 7776 or ISO/IEC 8208 and is acceptable to the state or phase of the interface.

3.3.3 tester: Refer to Lower Tester.

3.3.4 test case: Refer to Abstract Test Case.

3.3.5 test selection: Test selection is the process of choosing test cases according to the specific criteria based on the IUT's PICS and PIXIT in order to constitute a conformance test suite for the IUT.

3.3.6 test subgroup: A set of test cases that share a common characteristic, such as testing for proper, improper, or inopportune PDUs. A test subgroup is the smallest testable set of test cases that can be selected.

3.3.7 sub-function: A subset of the PDUs and functional capabilities of the protocol level above the IUT that are needed to allow data transfer testing to be accomplished.

4 Abbreviations

The following abbreviations are used in this part of ISO/IEC 8882:

DCE	Data Circuit-Terminating Equipment
DTE	Data Terminal Equipment
DXE	DTE or DCE
IUT	Implementation Under Test
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
RS	Remote Single Layer
SUT	System Under Test
TPDU	Transport Protocol Data Unit
TTCN	Tree and Tabular Combined Notation

5 Test notation

The test notation used in ISO/IEC 8882-2 and ISO/IEC 8882-3 is TTCN as defined in the DIS version of ISO/IEC 9646-3. This version of ISO/IEC 9646-3 is contained in annex A. ISO/IEC 8882-2 and ISO/IEC 8882-3 contain an annex describing the differences between the DIS version of TTCN used and the version of TTCN defined in ISO/IEC 9646-3.

6 Test suite structure

The test suite structure used in ISO/IEC 8882-2 and ISO/IEC 8882-3 is defined in ISO/IEC 9646-2 and is illustrated below.

```
                Test Suite Structure

        Test Group

            Test Subgroup 1 (Proper PDUs)
                Test Case No.101
                Test Case No.102
                        .
                        .
                Test Case No.1nn

            Test Subgroup 2 (Improper PDUs)
                Test Case No.201
                Test Case No.202
                        .
                        .
                Test Case No.2nn
```

Test Subgroup 3 (Inopportune PDUs)
Test Case No.301
Test Case No.302
.
.
.
Test Case No.3nn

7 Testing methodology

The testing methodology is based on the OSI Conformance Testing Methodology and Framework. The test method used is the Remote Single layer (RS) method. To employ the RS method effectively, the concept of using sub-functions of higher layer protocols is introduced. Sub-functions are a subset of the PDUs and functional capabilities of the protocol layer above the IUT that are needed to allow data transfer testing to be accomplished. The required properties of the sub-functions used are:

a) That the number and sequence of data-PDUs received from the IUT after receiving a data-PDU from the tester is predictable, and that the number received from the IUT is greater than zero.

b) That the reactions of the IUT upon receipt of these data-PDUs are known.

c) That the sub-function allows either the tester or IUT to initiate transmission of the data-PDUs.

d) That the sub-function allows for the exchange of data-PDUs by the layer under test with minimal interference from other functions of the protocol layer(s) above the IUT (e.g., PDU retransmission, error recovery, etc.).

Examples of data transfer configurations are shown for the Data Link Layer and the Packet Layer in figures 1 and 2 respectively.

NOTE — The requirements on underlying protocols are specified in ISO/IEC 8208, clause 3.

7.1 Test principles

The testing of the Data Link and the Packet Layer protocols is done separately. The data link layer is normally tested first since the packet layer requires the correct operation of the data link layer. The RS method is the selected test method since it cannot be assumed that a tester will be able to test completely each level as a separate entity. The RS method requires that the tester shall recognize and respond to a PDU received from the higher level protocols. The specific PDUs which shall be accepted are defined in ISO/IEC 8882-2 and ISO/IEC 8882-3.

7.2 Data transfer

The sub-function chosen by the IUT provider should create an alternating exchange of data-PDUs between the IUT and the tester. This exchange will be repeated until the sequence numbers of the layer under test have been rotated. The sub-function chosen shall be defined in the PIXIT of the IUT, and shall include the sequence and contents of the user data fields required for the test. Two examples of the use of a sub-function to accomplish data transfer testing are shown in figures 3 and 4.

A more detailed explanation of data transfer testing is provided in ISO/IEC 8882-2 and ISO/IEC 8882-3. These explanations also address the data transfer testing of send-only and receive-only IUTs.

It is recognized that an IUT provider may not be able to accomplish data transfer testing by this means. In such instances the data transfer tests are deselected.

7.3 Other user data fields

When necessary, the content of user data fields in other than data-PDUs shall be provided to the tester by the owner of the IUT in order to execute successfully the conformance test suite. In this case, the IUT requires the tester to transmit user data fields in accordance with higher level protocols which are operating above the IUT. For example, user data fields of call set-up, clearing, and interrupt packets of the Packet Layer may be affected.

The content of such user data fields shall be provided by the IUT owner in the PIXIT.

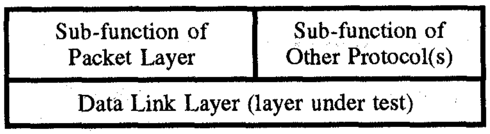
Sub-function of Packet Layer
Sub-function of Other Protocol(s)
Data Link Layer (layer under test)

Figure 1 — Data Link Layer Data Transfer Configuration

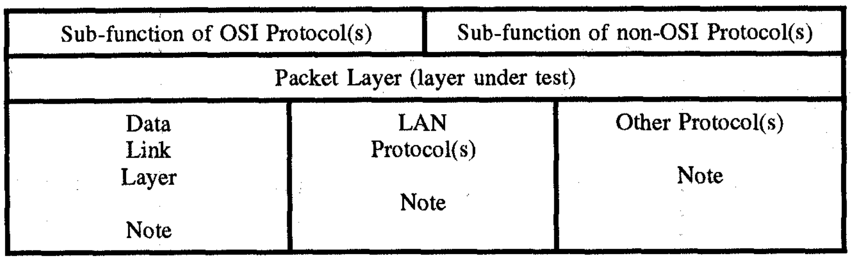
Sub-function of OSI Protocol(s)
Sub-function of non-OSI Protocol(s)
Packet Layer (layer under test)
Data Link Layer
Note
LAN Protocol(s)
Note
Other Protocol(s)
Note

Figure 2 — Packet Layer Data Transfer Configuration

7.4 Testing configuration

The SUT is connected to the tester, point-to-point, when participating in active testing. The points of observation and control for each test sequence are within the tester.

ISO/IEC 8882-2 and ISO/IEC 8882-3 include PIXIT proformas which, when completed, describe the dynamic conformance test environment.

7.5 Operational consideration

Testing is done in a controlled environment. It is not the intent of this document to define the operational characteristics of test devices used to achieve DTE Conformance Testing. However, it is highly desirable that the device be capable of segregating IUT test activity from normal operation of underlying layers. At a minimum, the tester should be capable of distinguishing between I-frame retransmission at the data link layer (due to T1 expiration) and a packet layer retransmission. Some recommended functions of the tester include:

a) Detection of failures of the physical layer.

b) The ability to respond transparently to timeout conditions at the data link layer.

c) Timely data link layer acknowledgement (to avoid retransmissions) when performing packet layer testing.

d) In the instance where an I-frame is retransmitted, the tester should properly acknowledge the frame and not pass it on to the packet layer. The tester shall be sensitive to failures

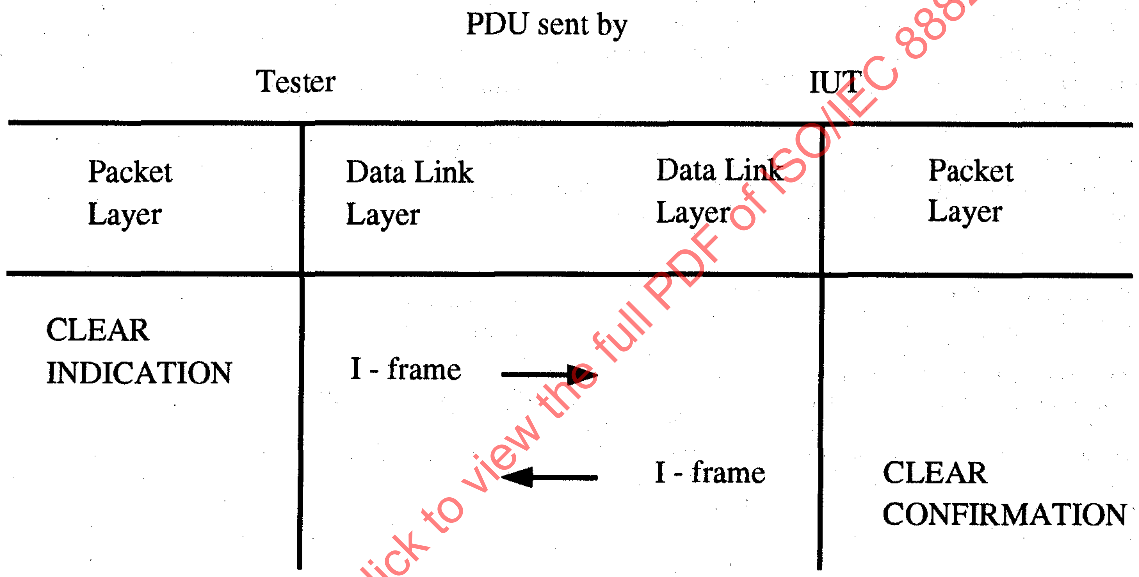

Figure 3 — Example of the use of a Packet Layer Sub-function

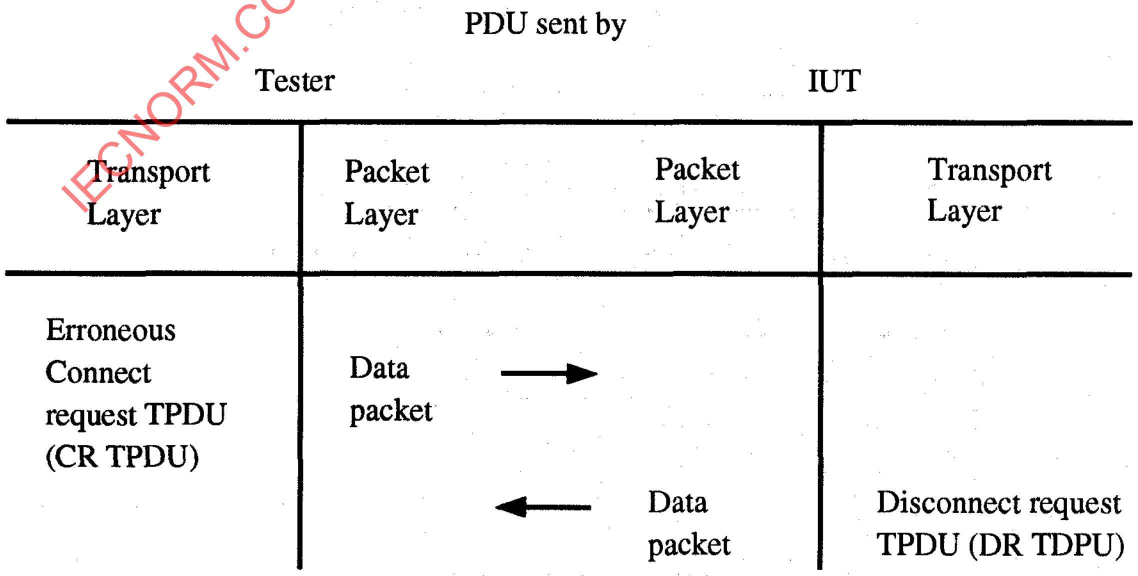

Figure 4 — Example of the use of a Transport Layer Sub-function directly over ISO/IEC 8208 (i.e. OSI Network Layer)

that interfere with the tests, and when such a condition is detected, the tester should abort the test.

e) The tester shall recognise the possibility of receiving unexpected PDUs which do not affect the results of the test case. These specific PDUs for each layer are defined in ISO/IEC 8882-2 and ISO/IEC 8882-3. In addition, other unexpected PDUs may be received which do affect the results of a test case. These PDUs will require further analysis and potentially, re-execution of the test case. Receipt of such PDUs may be due to interference from sources outside the realm of the X.25 environment (e.g. the IUT operating system, IUT operator).

7.6 DTE initiated actions

Generally the tester forces the IUT to transmit a particular PDU. However, in order to execute some test groups, it is required that the IUT initiate the transmission of particular PDUs. When a DTE-initiated action is required, it is specified in the appropriate test group. Direct control of such actions may not be feasible for the IUT owner. In such instances these tests are deselected.

7.7 Timing considerations

There are two types of timing considerations which should be taken into account — timing considerations for the tester and timing considerations for the SUT.

a) Tester Considerations: The tester shall allow for the time required by the IUT to progress from one test case to the next. This timing consideration should be accomodated for in the test preamble.

   For example, the time required by the IUT to initiate a CALL REQUEST after completing a CALL CLEARING operation, and the time required by the IUT to re-establish the data link after completing a disconnect operation. The precise timing requirements of the IUT shall be specified in the PIXIT, as defined in ISO/IEC 8882-2 and ISO/IEC 8882-3.

b) SUT Considerations: Where the protocol standard identifies a need for timers, values for those timers shall be those stated in the PIXIT.

7.8 Optional facility testing

Full testing of optional facilities is not possible because

a) optional facilities may be managed by levels above X.25; and

b) multiple combinations of optional facilities may be required depending on the applications running above X.25.

Optional facilities are tested individually. Where the IUT cannot support this method of testing these tests are deselected.

7.9 Transient states

It is recognized that for those IUTs that process PDUs sequentially, certain states are not realizable. Specifically, the testing of the IUT during the DXE defined states (for example, for the packet layer, r3 — Restart Indication, p3 — Incoming Call, p7 — Clear Indication, and d3 — Reset indication) may result in the testing of some other states (p1 — Ready, p4 — Data Transfer, d1 — Flow Control Ready). For example, to test the response to an error packet (inopportune or improper packet) in the DXE Restart Indication (r3) state, the tester will send a Restart Indication, immediately followed by the error packet. The tester expects the IUT to discard the error packet and then send a Restart Request in response to the error packet. However, the IUT generally responds immediately to the Restart Indication with a Restart Confirmation and processes the next packet from the packet level state r1. When these states are not observable in the IUT, transient test cases are deselected. The specific handling of transient state testing is described in ISO/IEC 8882-2 and ISO/IEC 8882-3.

8 Structure of other parts of ISO/IEC 8882

In order to ensure consistency between ISO/IEC 8882-2 and ISO/IEC 8882-3 the following items shall be included in those standards.

a) A PIXIT pro forma

b) A statement of Acceptable Unexpected responses.

c) A statement of Tester Timing Considerations.

d) PICS and PIXIT based abstract test selection rules.

e) A definition of the test cases.

f) A statement of conformance.

g) An annex describing the differences between the version of TTCN used and the version of TTCN defined in ISO/IEC 9646-3.

Annex A
(informative)

DIS level text for ISO/IEC 9646 - Part 3
The Tree and Tabular Combined Notation (TTCN)

Important - This Annex contains an extract from the DIS text of ISO/IEC 9646-3. The clause, figure and table numbering has been changed to align with the numbering in this standard. Example and proforma numbering is unchanged from the original text. References to the annexes in the original text have been placed in braces; for example "{Annex A}". All errors contained in the original text which were subsequently corrected in the published International Standard are also present here.

Introduction

This Part of the multi-part 'standard/recommendation' (hereafter abbreviated to 'standard*') defines a test notation, called the Tree and Tabular Combined Notation (TTCN), for use in the specification of 'OSI or related CCITT X.series or T.series' (hereafter abbreviated to 'OSI*') generic and abstract conformance test suites.

In constructing a generic or 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 generic and abstract test cases can be expressed in test suite standards*;

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 this multi-part standard*.

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

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

A.1 Scope

This Part of the multi-part standard* defines an informal test notation, called TTCN, for OSI* conformance test suites, which is independent of test methods, layers and protocols, and which reflects the abstract testing methodology defined in Parts 1 and 2 of this multi-part standard*.

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

This Part of this multi-part standard* 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 standard*, although those test cases may need to be included in a conformance test suite standard*.

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

This Part of this multi-part standard* applies to the specification of conformance test suites for OSI* protocols in OSI layers 2 to 7, specifically including ASN.1 based protocols. The specification of conformance test suites for multi-peer or Physical layer protocols is outside the scope of this standard*.

The relation between TTCN and formal description techniques is outside the scope of this standard*.

The specification of test cases in which more than one behaviour description is to be run concurrently, is outside the scope of this standard*.

NOTE 2 - Use of parallel trees and synchronization between them is expected to be covered by an Addendum to this standard*.

Although this Part of this multi-part standard* specifies requirements on abstract test suites written in TTCN, including their operational semantics, the means of realization of executable test suites from abstract test suites is outside the scope of this Part. Nevertheless, this Part specifies requirements on what a test suite standard* may specify about a conforming realization of the test suite.

NOTE 3 - ISO 9646-4 specifies requirements concerning test realization including ETS derivation.

A.2 Normative References

ISO 9646-1, *Information Processing Systems - Open Systems Interconnection - OSI Conformance Testing Methodology and Framework. - Part 1: General Concepts.* (See also CCITT Recommendation X.290)

ISO 9646-2, *Information Processing Systems - Open Systems Interconnection - OSI Conformance Testing Methodology and Framework. - Part 2: Abstract Test Suite Specification.* (See also CCITT Recommendation X.290)

ISO 646, *Information Processing Systems - Open Systems Interconnection - ISO 7-bit Coded Character Set for Information Exchange*

ISO 8824 (1989), *Information Processing Systems - Open Systems Interconnection - Abstract Syntax Notation One (ASN.1).* (See also CCITT Recommendation X.208)

ISO 8825 (1989), *Information Processing Systems - Open Systems Interconnection - Basic Encoding Rules for ASN.1.* (See also CCITT Recommendation X.209)

NOTE - These versions of ASN.1 include ASN.1 Extensions Addenda.

ISO 7498-1 , *Information Processing Systems - Open Systems Interconnection - Basic Reference Model.* (See also CCITT Recommendation X.200)

ISO TR 8509 , *Information Processing Systems - Open Systems Interconnection -Service Conventions.* (See also CCITT Recommendation X.210)

A.3 Definitions

A.3.1 Basic Terms from ISO 9646-1

The following terms defined in ISO 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 statement
h) conformance testing
i) conformance test suite
j) coordinated test method
k) distributed test method
l) embedded testing
m) executable test case
n) executable test suite
o) external test methods
p) fail verdict
q) foreseen outcome
r) generic test case
s) generic test suite
t) implementation under test

u) inconclusive verdict
v) inopportune test event
w) local test methods
x) lower tester
y) multi-layer testing
z) pass verdict
aa) PICS proforma
ab) PIXIT proforma
ac) point of control and observation
ad) protocol data unit
ae) protocol implementation
af) real tester
ag) remote test method
ah) syntactically invalid test event
ai) system under test
aj) test case
ak) test coordination procedures
al) test event
am) test group
an) test management protocol
ao) test outcome
ap) test purpose
aq) test realizer
ar) test step
as) test suite
at) unforeseen outcome
au) upper tester
av) valid test event
aw) verdict

A.3.2 Terms from ISO 7498-1

The following terms defined in ISO 7498-1 apply:

a) (N)-layer
b) (N)-protocol
c) (N)-protocol control information
d) (N)-protocol data unit
e) (N)-service
f) (N)-service access point
g) (N)-user data transfer syntax

A.3.3 Terms from ISO TR 8509

The following terms defined in ISO TR 8509 apply:

a) service primitive
b) service provider
c) service user

A.3.4 Terms from ISO 8825

The following term defined in ISO 8825 applies:

encoding

A.3.5 Terms from ISO 8824

The following terms defined in ISO 8824 apply:

a) NumericString
b) PrintableString
c) TeletexString
d) VideotexString
e) VisibleString
f) IA5String
g) GraphicString
h) GeneralString

A.3.6 TTCN Specific Terms

For the purposes of this standard* the following definitions apply:

A.3.6.1 Abbreviation identifier: A name for an abbreviation, which identifies its definition.

A.3.6.2 Attach statement: A TTCN statement which attaches a sub-tree to a calling tree.

A.3.6.3 Base constraint: Specifies a set of default values for each and every field in an ASP or PDU type declaration.

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

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

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

A.3.6.7 Calling tree: The behaviour tree to which a sub-tree is attached.

A.3.6.8 Constraints part: That component of a TTCN test suite concerned with the specification of the values of ASP parameters and parameter groups and PDU fields and field groups.

A.3.6.9 Constraints reference: A reference to a constraint, given in a behaviour line.

A.3.6.10 Decode expression: A specification of the decoding of PDUs embedded in ASPs or other PDUs.

A.3.6.11 Default behaviour: The events, and other TTCN statements, which may occur at any level of the associated tree, and which arc indicated in the default behaviour proforma.

A.3.6.12 Defaults library: The set of the default behaviours in a test suite.

A.3.6.13 Defaults reference: A structured name which specifies the location of the default in the defaults library.

A.3.6.14 Dotted identifier: An identifier, consisting of a base constraint identifier concatenated with one or more modified constraint identifiers, separated by dots.

A.3.6.15 Encode expression: A specification of the encoding of PDUs embedded in ASPs or other PDUs.

A.3.6.16 Field groups: A collection of one or more PDU fields which may occur in more than one PDU type declaration and which is defined in a separate declaration.

A.3.6.17 Implicit send event: A mechanism used in Remote methods for specifying that the IUT should be made to initiate a particular PDU or ASP.

A.3.6.18 Inheritance: The means by which constraint values specified for a base constraint are passed to a modified constraint.

A.3.6.19 Local tree: A behaviour tree defined in the same proforma as its calling tree.

A.3.6.20 Modified constraint: A subsequent constraint defined for an ASP or a PDU that already has a Base constraint, and which makes modifications on that Base constraint.

A.3.6.21 Multiple constraint: Declaration of a set of constraints for an ASP or PDU of a given type arranged in a single table.

A.3.6.22 Operational semantics: Semantics explaining the execution of a TTCN behaviour tree.

A.3.6.23 Otherwise event: The TTCN mechanism for dealing with unforeseen events in a controlled way.

A.3.6.24 Parameter groups: A collection of one or more ASP parameters which may occur in more than one ASP type declaration and which is defined in a separate declaration.

A.3.6.25 Pseudo-event: A pseudo-event is a TTCN expression or Timer operation appearing in the behaviour description.

A.3.6.26 Receive event: The receipt of an ASP or PDU at a named or implied PCO.

A.3.6.27 Root tree: The main behaviour tree of a test case, occurring at the level of entry into the test case.

A.3.6.28 Send event: The sending of an ASP or PDU to a named or implied PCO.

A.3.6.29 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.

A.3.6.30 Single constraint: Declaration of a constraint for a single ASP or PDU of a given type arranged in a single table.

A.3.6.31 Snapshot semantics: A semantic model to minimize 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.

A.3.6.32 Static chaining: The linking from the declaration of an ASP parameter or PDU field to the declaration of another ASP or PDU.

A.3.6.33 Static semantics: Semantic rules that restrict the usage of the TTCN syntax.

A.3.6.34 Sub-tree: An identifiable part of a behaviour tree which can be separated, then attached and executed at various places in that (or some other) behaviour tree.

A.3.6.35 Test case identifier: A short name for the test case.

A.3.6.36 Test case reference: A full name for the test case behaviour description, which defines its conceptual location in the test suite structure.

A.3.6.37 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.

A.3.6.38 Test step library: The set of the test step dynamic behaviour descriptions in the test suite.

A.3.6.39 Test step objective: An informal statement of what the test step is meant to accomplish.

A.3.6.40 Test Suite constant: One of a set of constants, *not* derived from the PICS or PIXIT, which will remain constant throughout the test suite.

A.3.6.41 Test suite parameter: One of a set of constants derived from the PICS or PIXIT which globally parameterize a test suite.

A.3.6.42 Test suite variable: One of a set of variables declared globally to the test suite, and which retain their values between test cases.

A.3.6.43 Timeout event: An event which is used within a behaviour tree to check for expiration of a specified timer.

A.3.6.44 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.

A.3.6.45 Tree header: That which prefixes a local behaviour tree. The header contains a tree identifier, and a specification of any parameters and their types used in the tree.

A.3.6.46 Tree identifier: A name identifying a local behaviour tree.

A.3.6.47 Tree indentation: A method of indicating the tree structure of a behaviour description. It is reflected in the behaviour description by indentation of text.

A.3.6.48 Tree leaf: The TTCN statement in a behaviour tree or sub-tree which has no specified subsequent behaviour.

A.3.6.49 Tree node: A single TTCN statement.

A.3.6.50 Tree notation: The notation used in TTCN to represent test cases as trees.

A.3.6.51 TTCN abbreviation: A method of indicating a textual substitution to be performed in a dynamic behaviour table.

A.3.6.52 TTCN statement: A TTCN statement is an event, a pseudo-event or construct which is specified in a behaviour description.

A.3.6.53 Unforeseen test event: A test event which has not been identified as a possible outcome in the test suite. It is normally handled using the OTHERWISE event.

A.3.6.54 Unqualified send event: A send event that does not have a Boolean expression or EncodedAs expression on the same statement line.

A.4 Abbreviations

A.4.1 Abbreviations Defined in ISO 9646-1

For the purposes of this Part of ISO 9646, the following abbreviations defined in clause 4 of ISO 9646-1 apply:

ASP : abstract service primitive
OSI : open systems interconnection
OSI* : OSI related CCITT X.series or T.series
PCO : point of control and observation
PDU : protocol data unit
PICS : protocol implementation conformance statement

PIXIT : protocol implementation extra information for testing
SAP : service access point
standard* : standard or recommendation
SUT : system under test

A.4.2 Abbreviations Defined in ISO 9646-2

For the purposes of this Part of ISO 9646, the following abbreviations defined in clause 4 of ISO 9646-2 apply:

DS : distributed single-layer (test method)
FDT : formal description technique
TTCN : tree and tabular combined notation

A.4.3 Other Abbreviations

For the purposes of this Part of ISO 9646, the following abbreviations also apply:

ASN.1 : abstract syntax notation one
BNF : The extended Backus-Naur form used in TTCN
CEId : connection endpoint identifier
FIFO : first in first out
TTCN.GR : tree and tabular combined notation, graphical form
TTCN.MP : tree and tabular combined notation, machine processable form

A.5 The Syntax Forms of TTCN

TTCN is provided in two forms:

a) a graphical form (TTCN.GR) suitable for human readability;

b) a machine processable form (TTCN.MP) suitable for transmission of TTCN descriptions between machines and possibly suitable for other automated processing.

TTCN.GR is defined using tabular proformas. TTCN.MP differs from TTCN.GR only in syntax; keywords, instead of boxes, are used as information delimiters. The syntax of TTCN.MP is defined in Annex A of this standard* by means of a BNF grammar.

As an aid to clarifying the TTCN.GR many TTCN.MP productions are embedded in the text of this standard*, and are marked: SYNTAX DEFINITION. To improve the readability of this document some productions will appear in the text in several places.

The two forms of TTCN are equivalent. If there is a conflict between the two forms, this is an error, and should be reported back to the standards* organization via a defect report. In such cases, however, the TTCN.MP shall take precedence over the TTCN.GR form pending correction by the standards* organization.

A.6 Compliance

6.1 Test suites that claim to comply with this Part of this multi-part standard* shall state that they comply with the requirements for either TTCN.GR or TTCN.MP.

6.2 Test suites that claim to comply with the requirements of TTCN.GR shall comply with the TTCN.GR syntax requirements stated in clauses A.8 through A.16 and clause {A.3}. Generic test suites may also use the options specified in clause A.18.

6.3 Test suites that claim to comply with the requirements of TTCN.MP shall comply with the TTCN.MP syntax requirements stated in clause {A.3}.

6.4 Test suites that claim to comply with this Part of this multi-part standard* shall comply with the static semantic requirements specified in clauses A.8 through A.15 and have operational semantics in accordance with the definition of the operational semantics in {Annex B}, such that they are semantically valid.

6.5 A test suite standard* that claims to comply with this Part of this multi-part standard* shall require that any realization of that test suite that claims to conform to that test suite standard* shall:

a) have operational semantics equivalent to the operational semantics of the test suite as defined by {Annex B};

b) be able to produce a conformance log that as a minimum meets all the requirements in clause 17;

c) comply with ISO 9646-4.

A.7 Conventions

A.7.1 Introduction

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

A.7.2 TTCN.GR Table Proformas

The TTCN.GR notation is defined using a number of different types of table. The following conventions will be used in the description of proformas for these tables:

a) Bold text (**like this**) shall appear verbatim in each actual table in a TTCN test case;

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

EXAMPLE 1 - *SuiteIdentifier* corresponds to production 3 in {Annex A}

A.7.3 Syntactic Metanotation

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

Table A.1: The TTCN.MP Syntactic Metanotation

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

A.7.4 TTCN.MP Syntax Definitions

A.7.4.1 Complete tables defined in TTCN.GR are represented in TTCN.MP by productions of the kind:

$BEGIN_KEYWORD .... $END_KEYWORD

EXAMPLE 2 - TS_PARdcls ::= **$Begin_TS_PARdcls** {TS_PARdcl}+ **$End_TS_PARdcls**

Normally, these productions contain at least one mandatory field.

A.7.4.2 Lines of a table, i.e. sets of fields in a table, are represented by productions of the kind:

$KEYWORD .... $END_KEYWORD

BEGIN does not appear in the opening keyword.

EXAMPLE 3 - TS_PARdcl ::= **$TS_PARdcl** TS_PARid TS_PARtype PICS_PIXIT [Comment] **$End_TS_PARdcl**

A.7.4.3 Individual fields in a line are represented by:

$KEYWORD ....

There is no closing keyword.

EXAMPLE 4 - TS_PARid ::= **$TS_PARid** TS_PARidentifier

NOTE - Symbols such as TS_PARid can only be used to name a field. The contents of the field must be called, in that case, TS_PARidentifier, further defined as an identifier.

EXAMPLE 5 - TS_PARidentifier ::= Identifier

A.7.4.4 Sets of tables, up to and including the test suite, are represented by productions of the kind:

$KEYWORD .... $END_KEYWORD

EXAMPLE 6 - ASPdcls ::= **$ASPdcls** [TTCN_ASPdcls] [ASN1_ASPdcls] **$End_ASPdcls**

EXAMPLE 7 - Suite ::= **$Suite** SuiteId SuiteOverview Declarations DynamicPart ConstraintsPart **$End_Suite**.

A.7.4.5 All other productions defining non-terminal symbols have no keywords at the beginning or the end of the right-hand expression.

EXAMPLE 8 - TimerIdentifier ::= Identifier

Most of the terminal symbols used in the TTCN.MP grammar are defined in clause {A.3.9}.

A.7.5 TTCN.MP and TTCN.GR Symbols

a) TTCN keywords (terminal symbols) that belong only to the TTCN.MP form start with the dollar character ($):

EXAMPLE 9 - $SuiteId

b) TTCN keywords (terminal symbols) that belong to both the TTCN.MP and the TTCN.GR forms do not start with the dollar character ($):

EXAMPLE 10 - the TTCN keyword **REPEAT**

A.7.6 Uniqueness of Names in TTCN Test Suites

A.7.6.1 Identifiers used within TTCN test suites are case sensitive. Whenever in the TTCN an identifier used from a protocol standard* contains "-" (dash) the dash shall be replaced by "_" (underscore).

A.7.6.2 All identifier names of the following items shall have unique meaning throughout the test suite (i.e. declarations and constraints).

a) Predefined TTCN types;

b) user defined TTCN types;

c) User defined operators;

d) Test suite parameters;

e) Test suite constants;

f) Test suite variables;

g) Test case variables;

h) PCO types;

i) PCO names;

j) Timer identifiers;

k) Abbreviation names;

l) ASP types;

m) Parameter group types;

n) PDU types;

o) Field group types;

p) ASP constraint names;

q) Parameter group constraint names;

r) PDU constraint names;

s) Field group constraint names;

t) Test case references;

u) Test case identifiers;

v) Test step references;

w) Test step identifiers;

x) Default references;

y) Default identifiers.

A.7.6.3 When ASN.1 is used in a TTCN test suite, ASN.1 identifiers from the following list shall be unique throughout the test suite, regardless of whether the ASN.1 definition is explicit or implicit by reference:

a) identifiers occurring in an ASN.1 ENUMERATED type as distinguished values;

b) identifiers occurring in a "NamedNumberList" of an ASN.1 INTEGER type;

c) "UserTypeIdentifiers" of a User ASN.1 Type Definition.

A.7.6.4 The names of ASP parameters shall be unique within the ASP in which they are declared. The names of PDU fields shall be unique within the PDU in which they are declared.

A.7.6.5 The names of parameters within a parameter group shall be unique within each ASP where it will expand. The names of fields within a field group shall be unique within each PDU (and ASP, if applicable) where it will expand.

A.7.6.6 Labels used within a tree shall be unique within a tree.

A.7.6.7 The tree header names used for local test steps shall be unique within the dynamic behaviour description in which they appear.

A.7.6.8 The formal parameter names which may optionally appear as part of the following shall be unique within that formal parameter list:

a) User defined operations declaration;

b) Tree header of a local tree;

c) Test step identifier;

d) Defaults identifier;

e) Parameterized constraint declaration.

If usage of a formal parameter name within a dynamic behaviour description clashes with any other identifier defined in the test suite, the formal parameter name shall take precedence.

A.8 TTCN Test Suite Structure

A.8.1 Introduction

TTCN allows a test suite to be hierarchically structured in accordance with clause 8.1 of Part 1 of this multipart standard*. The components of this structure are:

a) test groups;

b) test cases;

c) test steps;

d) test events.

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 hierarchically structure test steps and defaults. (This hierarchical structure is optional).

A.8.2 Test Group References and Test Case References

A.8.2.1 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 9646-1, clause 8, figure 10): references to TTCN test groups shall have the following syntax:

SYNTAX DEFINITION:

- TestGroupReference ::= SuiteIdentifier "/" {TestGroupIdentifier "/"}+

EXAMPLE 11 - A Transport group reference: TRANSPORT/CLASS0/CONN_ESTAB/

A.8.2.2 References to the test cases shall have the following syntax:

SYNTAX DEFINITION:

- TestCaseReference ::= (TestGroupReference TestCaseName) | (SuiteIdentifier "/" TestCaseName)

EXAMPLE 12 - Transport Test Case References:

TRANSPORT/INIT

TRANSPORT/CLASS0/CONN_ESTAB/LT_INIT

where TRANSPORT is the name of the test suite, CLASS0 and CONN_ESTAB are test groups, and INIT and LT_INIT are test case names

A.8.2.3 The test case references define the structure of the test suite.

A.8.3 Test Step Group References and Test Step References

A.8.3.1 Decomposition of a test case (or test step) into test steps is achieved by attaching behaviour sub-trees using the TTCN ATTACH statement. In TTCN the terms test step and sub-tree are synonymous. The position of a test step in the test suite structure is

not explicit in the test step reference. The concept of a test step may be expressed in several ways in TTCN. A test step may be:

- implicit in the behaviour tree;
- local to a test case or test step;
- globally accessible as a member of the test step library.

A.8.3.2 In the case of an implicit test step it is impossible to identify or attach the test step. A local test step has no test step reference - it is identifiable through a local tree name. In the case of a global test step, the test step reference specifies the test step's location in the test step library. The test step library has no influence on the test suite structure itself. Test step groups and test step references shall have the following syntax:

SYNTAX DEFINITION:

- TestStepGroupReference ::= SuiteIdentifier "/" {TestStepGroupIdentifier "/"}+
- TestStepReference ::= (TestStepGroupReference TestStepName) | (SuiteIdentifier "/" TestStepName)

EXAMPLE 13 - Transport Test step references:

```
TRANSPORT/STEP_A
TRANSPORT/STEP_LIBRARY/CLASS0/CONN_ESTAB/STEP_B
```

A.8.4 Default Group References and Default Behaviour References

Default behaviours (if any) are located in a default behaviours library.

A default reference specifies the default's location in the default library. The defaults library has no influence on the test suite structure itself. Default groups and default references shall have the following syntax:

SYNTAX DEFINITION:

- DefaultGroupReference ::= SuiteIdentifier "/" {DefaultGroupIdentifier "/"}
- DefaultReference ::= (DefaultGroupReference DefaultName) | (SuiteIdentifier "/" DefaultName)

EXAMPLE 14 - Transport default references:

TRANSPORT/DEF_A

TRANSPORT/DEFAULT_LIBRARY/CLASS0/DEF_B

A.9 Components of a TTCN Test Suite

An abstract test suite written in TTCN shall have the following four sections in the following order:

a) Suite Overview (clause A.10),

which is the information needed for the general presentation and understanding of the test suite, such as test references and a description of its overall purpose;

b) Declarations Part (clause A.11),

which is the set of components that comprise the test suite (e.g. PCOs, Timers, ASPs, PDUs, and their parameters or fields) is described. This section shall contain the definition of any abbreviations to be used later in the test suite;

c) Constraints Part (clauses A.12, A.13, A.14),

which is the set 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 Modular Method; or

3) both TTCN tables and the ASN.1 Modular Method.

d) Dynamic Part (clause A.15),

which comprises three sections that contain tables specifying test behaviour expressed mainly in terms of the occurrence of ASPs 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).

SYNTAX DEFINITION:

- Suite ::= **$Suite** SuiteId SuiteOverview Declarations ConstraintsPart DynamicPart **$End_Suite**

A.10 Suite Overview

This section 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 9646-2 clause 14);

e) a reference to where in the Abstract Test Suite specification the mapping of the PICS and PIXIT entries used in test case selection is specified;

f) an indication of the test method or methods to which the test suite applies, plus for the Coordinated Methods a reference to where the Test Management Protocol is specified. When TTCN is being used for an abstract test suite then there shall only be one test method, but when it is being used for a generic test suite then it may be that different test cases will be written in the style of different test methods;

g) other information which may aid understanding of the test suite, such as how it has been derived; this should be included as a comment;

h) a three-part test suite index:

1) a test case index, consisting of the test case identifier, test case reference, page number and a description of each test case (shortened form of the test purpose). The test cases shall be organized according to the structure of the test suite, optionally giving test group identifier or test group reference, page and test group objective (in the description column) where appropriate in the structure to show the test groups;

NOTE - There is no need to include test group identifiers in the first column because they are included in the test group references

2) a test step index, consisting of the test step identifier, test step reference, page number and a description of each test step (shortened form of the test step objective). The test steps shall be organized according to the structure of the test step library;

3) a defaults index, consisting of the defaults identifier, defaults reference, page number and a description of each default (shortened form of the default objective). The defaults shall be organized according to the structure of the defaults library.

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

Test Suite Overview			
Suite Name:	*SuiteIdentifier*		
Standards ref:	*Reference*		
PICS proforma ref:	*Reference*		
PIXIT proforma ref:	*Reference*		
PICS/PIXIT use:	*Reference*		
Test Method(s):	*FreeText*		
Comments:	*FreeText*		
Test Group or Case Identifier	**Test Group or Case Reference**	**Page**	**Description**
. . *TestCaseIdentifier* | *TestGroupIdentifier* .	. . *TestCaseReference* | *TestGroupReference* .	. . *Number* . .	. . *FreeText* . .
Test Step Identifier	**Test Step Reference**	**Page**	**Description**
. . *TestStepIdentifier* . .	. . *TestStepReference* . .	. . *Number* . .	. . *FreeText* . .
Default Identifier	**Default Reference**	**Page**	**Description**
. . *DefaultIdentifier* . .	. . *DefaultReference* . .	. . *Number* . .	. . *FreeText* . .

Proforma 1: Test Suite Overview

SYNTAX DEFINITION:

- SuiteOverview ::= **$Begin_SuiteOverview** SuiteId StandardsRef PICSref PIXITref HowUsed TestMethods [Comment] SuiteIndex **$End_SuiteOverview**
- SuiteIdentifier ::= Identifier
- Reference ::= BoundedFreeText
- FreeText ::= {ExtendedAlphaNum}
- SuiteIndex ::= **$SuiteIndex** TestCaseIndex TestStepIndex DefaultIndex **$End_SuiteIndex**
- TestCaseIndex ::= **$TestCaseIndex** {((TestCaseId [TestCaseRef]) | ([TestGroupId] [TestGroupRef])) Description}+ **$End_TestCaseIndex**
- TestGroupRef ::= **$TestGroupRef** TestGroupReference
- TestStepIndex ::= **$TestStepIndex** {TestStepID TestStepRef Description} **$End_TestStepIndex**
- DefaultIndex ::= **$DefaultIndex** {DefaultID DefaultRef Description} **$End_DefaultIndex**

If a dollar symbol ($) is needed in free text then it shall be preceded by the special character backslash (\), otherwise Free Text shall not include a dollar symbol ($). Free Text shall not end with the character backslash.

A.11 Declarations

A.11.1 Introduction

The purpose of the declarations section is to describe all the components used in the test suite. All components referenced in the dynamic part shall have been declared in the declarations part. These components are:

a) User defined types (clause A.11.2.3);

b) User defined operations (clause A.11.4.3);

c) Test suite parameters (clause A.11.5);

d) Test suite constants (clause A.11.6);

e) Test suite variables (clause A.11.7.1);

f) Test case variables (clause A.11.7.2);

g) PCO declarations (clause A.11.8);

h) Timer declarations (clause A.11.9);

i) Abbreviations (clause A.11.10).

j) ASP type declarations (clause A.11.11);

k) ASP parameter group type declarations (clause A.11.11.3);

l) PDU type declarations (clause A.11.12);

m) PDU field group type declarations (clause A.11.12.3);

SYNTAX DEFINITION:

- Declarations ::= **$Declarations** [UserTYPEdefs] [UserOPdefs] [TS_PARdcls] [TS_CONSTdcls] [TS_VARdcls] [TC_VARdcls] PCOdcls [TIMERdcls] [Abbreviations] [ASPdcls] PDUdcls **$End_Declarations**

A.11.2 General TTCN Types

A.11.2.1 Introduction

TTCN supports both a number of predefined types and mechanisms that allow the definition of user declared types. These types may be used throughout the test suite and may be referenced when test suite parameters, constants, variables, ASP parameters or PDU fields are defined.

SYNTAX DEFINITION:

- Type ::= PredefinedType | UserDefinedType

A.11.2.2 Predefined TTCN Types

A number of commonly used types are predefined for use in TTCN. These types may be referenced even though they do not appear in a type declaration in a test suite. All other types used in a test suite shall be declared in the User Type Declarations and referenced by name.

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

b) **BOOLEAN Predefined Type**: a type consisting of two distinguished values;

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

d) **HEXSTRING Predefined Type:** a type whose distinguished values are the ordered sequences of zero, one, or more semi-octets, a semi-octet being an ordered sequence of four bits;

e) **OCTETSTRING Predefined Type**: a type whose distinguished values are the ordered sequences of zero, one, or more octets, each octet being an ordered sequence of eight bits; thus an OctectString is an even number of HEXSTRING digits;

f) **Character String Predefined Types**: types whose distinguished values are zero, one, or more characters from some character set; the character string types listed in table A.2 may be used; they are defined in section two of ISO 8824.

Table A.2: Predefined Character String Types

NumericString
PrintableString
TeletexString (i.e. T61String)
VideotexString
VisibleString (i.e. ISO646String)
IA5String
GraphicString
GeneralString

SYNTAX DEFINITION:

- PredefinedType ::= **INTEGER** | **BOOLEAN** | PredefinedStringType
- PredefinedStringType ::= **BITSTRING** | **HEXSTRING** | **OCTETSTRING** | CharacterString
- CharacterString ::= **NumericString** | **PrintableString** | **TeletexString** | **VideotexString** | **VisibleString** | **IA5String** | **GraphicString** | **GeneralString**

A.11.2.3 User Defined Types for a Specific Test Suite

A.11.2.3.1 Introduction

Types specific to a test suite may be introduced by the TTCN user. These may be defined using the TTCN user type definitions table and/or ASN.1.

SYNTAX DEFINITION:

- UserTYPEdefs ::= **$UserTYPEdefs** [TTCN_TYPEdefs] {ASN1_TYPEdef} **$End_UserTYPEdefs**

A.11.2.3.2 User Type Definitions - Tabular Form

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

a) a name for the type;

b) the base type (if any),

which is required when defining a new type equivalent to or a subset of a previously defined, or predefined, type (see c2A, c2B); the test case writer shall ensure that the base type is compatible with the new type being defined;

the base type field may be left blank when the definition of the base type is implicit in the definition of a restricted length type (see c2B);

c) a definition of the type, provided in one of the following manners:

1) by listing the set of distinguished values of another type; these values comprise the new type;

2) by specifying a subset of the distinguished values of another type; this may be done in a number of ways:

A) by specifying a subrange within a list of values, or integer;

B) by restricting the length of a predefined or user defined *string* type (e.g. BITSTRING, HEXSTRING, IA5STRING).

The length shall be specified in the unit represented by the corresponding string type: BITSTRING in number of bits, HEXSTRING in number of hexadecimal digits, OCTETSTRING in number of octets and other character strings in number of characters.

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

User Type Definitions			
Name	**Base Type**	**Definition**	**Comments**
. . *UserTypeIdentifier* . .	. . *BaseType* . .	. . *TypeDefinition* . .	. . *FreeText* . .

Proforma 2: User Type Definitions

SYNTAX DEFINITION:

- TTCN_TYPEdefs ::= **$Begin_TTCN_TYPEdefs** {TTCN_TYPEdef}+ **$End_TTCN_TYPEdefs**
- TTCN_TYPEdef ::= **$TTCN_TYPEdef** UserTypeId [Base] TypeDef [Comment] **$End_TTCN_TYPEdef**
- UserTypeId ::= **$UserTypeId** UserTypeIdentifier
- UserTypeIdentifier ::= Identifier
- Base ::= **$Base** Type
- FIELDgroupConstraintTypeDef ::= **$TypeDef** TypeDefinition
- TypeDefinition ::= TypeIfNoBaseUsed | TypeIfBaseUsed
- TypeIfNoBaseUsed ::= TypeAndLength
- TypeIfBaseUsed ::= Range | VALlist
- Range ::= "(" SignedNumber To SignedNumber ")"
- To ::= **TO** | ".."
- VALlist ::= "(" SignedValue {Comma SignedValue} ")"
- Comma ::= ","
- TypeAndLength ::= StringType "[" Number "]"
- SignedNumber ::= ["-"] Number
- SignedValue ::= ["-"] Value

Where a range of values is used as a TTCN type definition, that range shall be stated with the lower of the two values on the left.

User Type Definitions			
Name	**Base Type**	**Definition**	**Comments**
Transport_Classes	INTEGER	(0, 1, 2, 3, 4)	Classes that may be used for Transport layer connection
Class_Number	INTEGER	(0 .. 4)	

Example 15: Example User Type Definition

A.11.2.3.3 User Type Definitions in ASN.1

User types specified in ASN.1 (ISO 8824) shall be defined in the following proforma:

User ASN.1 Type Definition
Type Name: *UserTypeIdentifier*
ASN.1 Definition or Reference
. . . *ASN1Ref* | *ASN1Def* . . .

Proforma 3: User ASN.1 Type Definition

SYNTAX DEFINITION:

- ASN1_TYPEdef ::= **$Begin_ASN1_TYPEdef** UserTypeId ASN1RefOrDef **$End_ASN1_TYPEdef**
- ASN1RefOrDef ::= **$ASN1RefOrDef** (ASN1Ref | ASN1Def) **$End_ASN1RefOrDef**
- ASN1Ref ::= **#REF** {SymbolsFromModule}+
- SymbolsFromModule ::= SymbolList **FROM** ModuleReference
- SymbolList ::= TypeReference { Comma TypeReference}
- TypeReference ::= */* Defined in ISO 8824 */*
- ModuleReference ::= */* Defined in ISO 8824 */*
- ASN1Def ::= {TypeAssignment}+ */* Defined in ISO 8824 */*

When an ASN.1 reference is used the free text entry shall be preceded by the keyword **#REF**. This applies throughout TTCN where ASN.1 references are used.

A.11.2.4 Type Compatibility for ASN.1 Types

Data objects are assignment compatible:

a) if their type names are identical;

b) or if one type is derived from the other type by:

1) subtyping (i.e. subrange types, length restriction);

2) or renaming the original type;

EXAMPLE 16 - int ::= INTEGER

Data objects of type int and INTEGER are assignment compatible.

3) or changing the tag of the original type;

EXAMPLE 17 - int2 ::= [5] int

int and int2 are assignment compatible.

Constants that are local to the type definition may be defined using ASN.1 (Named Values) are only compatible to the type they were defined in.

If type A is assignment compatible with type B, and type B is assignment compatible with type C, then type A is always assignment compatible with type C (i.e. assignment compatibility is transitive).

A.11.2.5 Type Compatibility for TTCN Types

For types (user defined types, ASP types, PDU Types) defined using the tabular method the rules a) and b1) from clause A.11.2.4 apply.

NOTE - Renaming cannot be achieved in the tabular method.

A.11.2.6 Relationship Between ASN.1 and TTCN Types

The following TTCN predefined types are assignment compatible with their corresponding ASN.1 types:

a) INTEGER;

b) BOOLEAN;

c) BITSTRING;

d) OCTETSTRING;

e) and all character string types.

A.11.3 Value Denotation

A.11.3.1 Values of Predefined Types

The values of predefined types shall be denoted as follows:

a) **INTEGER Values**: 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 Values**: values of the BOOLEAN type are TRUE or FALSE;

c) **BITSTRING Values**: 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 18 - '01101'B

d) **HEXSTRING Values**: values of type HEXSTRING shall be denoted by an arbitrary number (possibly zero) of the characters:

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 character is used to denote the value of a semi-octet using a hexadecimal representation;

EXAMPLE 19 - 'AB01D'H

e) **OCTETSTRING Values**: values of type OCTETSTRING shall be denoted by an arbitrary, but even, number (possibly zero) of the characters:

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 character is used to denote the value of a semi-octet using a hexadecimal representation;

EXAMPLE 20 - 'FF96'O

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

SYNTAX DEFINITION:

- Value ::= Number | BooleanValue | Bstring | Hstring | Ostring | Cstring
- Number ::= (NonZeroNum {Num}) | **0**
- NonZeroNum ::= **1** | .... | **9**
- Num ::= **0** | .... | **9**
- BooleanValue ::= **TRUE** | **FALSE**
- Bstring ::= "'" {Bin | Wildcard} "'" **B**
- Bin ::= **0** | **1**
- Hstring ::= "'" {Hex | Wildcard} "'" **H**
- Hex ::= Num | **A** | .... | **F**
- Ostring ::= "'" {Oct | Wildcard} "'" **O**
- Oct ::= Hex Hex
- Cstring ::= """ {Char | Wildcard | "\"} """
- Char ::= /* *a character defined by the relevant character string type* */

- ExtValue ::= SignedValue | Wildcard | "-"
- Wildcard ::= "?" | "*"
- SignedValue ::= ["-"] Value
- Identifier ::= Alpha{AlphaNum | "_" }
- AlphaNum ::= Alpha | Num
- Alpha ::= **A** | .... | **Z** | **a** | .... | **z**

A.11.3.2 Values of ASN.1 Predefined Types

The values of the ASN.1 predefined types INTEGER, BOOLEAN, BITSTRING, OCTETSTRING and all character string types shall be denoted using the same notation defined in clause A.11.3.1 for the corresponding TTCN predefined types.

Values of an ASN.1 ENUMERATED type shall be denoted using the Enumerated Identifiers.

NOTE - TTCN does not support a value notation for the ASN.1 predefined type REAL nor for structured ASN.1 types.

A.11.3.3 Values of User Defined Types and ASN.1 User Defined Types

Values of types that are derived from TTCN predefined or ASN.1 predefined types by subsetting shall be denoted in the same manner as the values of the type they were derived from.

A.11.4 TTCN Operators

A.11.4.1 Introduction

TTCN supports both a number of predefined operators and mechanisms that allow the definition of user operators. These operators may be used throughout any dynamic behaviour descriptions.

A.11.4.2 Predefined TTCN Operators

A.11.4.2.1 Introduction

The predefined TTCN operators fall into three categories:

a) arithmetic;

b) relational;

c) Boolean.

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

Table A.3: Precedence of Operators

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

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

<u>SYNTAX DEFINITION:</u>

- AddOp ::= "+" | "-" | **OR**
- MultiplyOp ::= "*" | "/" | **MOD** | **AND**
- UnaryOp ::= "+" | "-" | **NOT**
- Relop ::= "=" | "<" | ">" | "<>" | ">=" | "<="

A.11.4.2.2 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.

NOTE - Since ASN.1 and TTCN Predefined INTEGER are assignment compatible no further classification of the result type is needed.

In the case where "-" is used as the unary operator the rules for operands apply as well. The result is the negative value of the operand if it was positive and vice versa.

A.11.4.2.3 Predefined Relational Operators

The predefined relational operators are:

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

They represent the relations of equality, less than, greater than, not equal too, greater than or equal to and less than or equal to. Operands of "=" and "<>" may be of an arbitrary type. The two operands shall be assignment 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 CHARACTER STRINGS may contain the wildcard characters "*" and "?". In this case the comparison is performed according to the pattern matching rules defined in clause A.12.3.3.

A.11.4.2.4 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) or derivations of 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.

A.11.4.3 User Defined Operators for a Specific Test Suite

Operators specific to a test suite may be introduced by the TTCN user. To define a new operation, the following information shall be provided:

a) a name for the operation;

b) a list of the input parameters and their types;
this is a list of the formal parameter names and types. Each parameter name shall be followed by a colon and then the name of the parameter's type.

If 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 parameter's type.

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.

EXAMPLE 21 - The following are equivalent methods of specifying a parameter list using two INTEGER parameters and one BOOLEAN parameter:

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

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

c) the type of the result;

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

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

User Operation Definition	
Operation Name:	*OPidentifier [FormalPARlist]*
Result Type:	*Type*
Description	
. . *FreeText* . .	

Proforma 4: User Operation Definition

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

When a User Defined Operation is invoked, the types of the actual parameters shall match the types of the formal parameters.

SYNTAX DEFINITION:

- UserOPdefs ::= **$UserOPdefs** {UserOPdef}+ **$End_UserOPdefs**
- UserOPdef ::= **$Begin_UserOPdef** OPid OPresult OPdescription **$End_UserOPdef**
- OPid ::= **$OPid** OPidentifier [FormalPARlist]
- OPidentifier ::= Identifier
- OPresult ::= **$OPresult** Type
- OPdescription ::= **$OPdescription** BoundedFreeText

The definition of a string operation is given below:

User Operation Definition	
Operation Name:	substr (source:IA5STRING; start_index, length:INTEGER)
Result Type:	IA5STRING
Description	
substr(source, start_index, length) is the string of length *length* starting from index *start_index* of the source string *source*. For example: substr("abcde",3,2) = "cd" substr("abcde",4,99999) = "de"	

Example 22: Definition of Operation substr

A.11.5 Test Suite Parameters

The purpose of this section is to declare constants derived from the PICS and/or PIXIT which globally parameterize the test suite. These constants are referred to as test suite parameters.

NOTE - In real cases of testing, test suite parameters will be bound to a value when the PICS/PIXIT processing occurs. The method of PICS and/or PIXIT processing is implementation dependent and is not specified in the TTCN notation.

The following information relating to each test suite parameter shall be provided in this section:

a) its name;

b) its type;

c) PICS/PIXIT entry reference,
which is a reference to an individual PICS/PIXIT proforma entry that will clearly identify where the value to be used for this test suite parameter will be found.

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

Test Suite Parameters			
Name	**Type**	**PICS/PIXIT Ref.**	**Comments**
. . *TS_PARidentifier* . .	. . *Type |* *ReferenceType* .	. . *Reference* . .	. . *FreeText* . .

Proforma 5: Test Suite Parameters

SYNTAX DEFINITION:

- TS_PARdcls ::= **$Begin_TS_PARdcls** {TS_PARdcl}+ **$End_TS_PARdcls**
- TS_PARdcl ::= **$TS_PARdcl** TS_PARid TS_PARtype PICS_PIXIT [Comment] **$End_TS_PARdcl**
- TS_PARid ::= **$TS_PARid** TS_PARidentifier
- TS_PARidentifier ::= Identifier
- TS_PARtype ::= **$TS_PARtype** (Type | ReferenceType)
- PICS_PIXIT ::= **$PICS_PIXIT** Reference

Test Suite Parameters			
Name	**Type**	**PICS/PIXIT Ref.**	**Comments**
PAR1 PAR2 PAR3	INTEGER INTEGER INTEGER	PICS question xx PICS question yy PIXIT question zz	

Example 23: Test Suite Parameter Declarations

A.11.6 Test Suite Constants

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

The following information relating to each test suite constant shall be provided in this section:

a) its name;

b) its type;

c) its value.

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

Test Suite Constants			
Name	**Type**	**Value**	**Comments**
.	.	.	.
.	.	.	.
TS_CONSTidentifier	*Type*	*Value*	*FreeText*
.	.	.	.
.	.	.	.

Proforma 6: Test Suite Constants

SYNTAX DEFINITION:

- TS_CONSTdcls ::= **$Begin_TS_CONSTdcls** {TS_CONSTdcl}+ **$End_TS_CONSTdcls**
- TS_CONSTdcl ::= **$TS_CONSTdcl** TS_CONSTid TS_CONSTtype TS_CONSTvalue [Comment] **$End_TS_CONSTdcl**
- TS_CONSTid ::= **$TS_CONSTid** TS_CONSTidentifier
- TS_CONSTidentifier ::= Identifier
- TS_CONSTtype ::= **$TS_CONSTtype** Type
- TS_CONSTvalue ::= **$TS_CONSTvalue** SignedValue

Test Suite Constants			
Name	**Type**	**Value**	**Comments**
TS_CONST1	BOOLEAN	TRUE	
TS_CONST2	IA5String	"A string"	

Example 24: Declaration of Test Suite Constants

A.11.7 TTCN Variables

A.11.7.1 Test Suite Variables

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

A test suite variable is used whenever it is necessary to pass information from one test case to another.

All test suite variables to be used in a test suite shall be declared. The following information shall be provided for each variable declaration:

a) its name;

b) its type;

c) its initial value (if any),

where the initial value column is used when it is desired to assign an *optional* initial value to a test suite variable at its point of declaration.

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

Test Suite Variables			
Name	**Type**	**Value**	**Comments**
. . *TS_VARidentifier* . .	. . *Type* | *ReferenceType* .	. . *Value* . .	. . *FreeText* . .

Proforma 7: Test Suite Variables

SYNTAX DEFINITION:

- TS_VARdcls ::= **$Begin_TS_VARdcls** {TS_VARdcl}+ **$End_TS_VARdcls**
- TS_VARdcl ::= **$TS_VARdcl** TS_VARid TS_VARtype [TS_VARvalue] [Comment] **$End_TS_VARdcl**
- TS_VARid ::= **$TS_VARid** TS_VARidentifier
- TS_VARidentifier ::= Identifier
- TS_VARtype ::= **$TS_VARtype** (Type | ReferenceType)
- TS_VARvalue ::= **$TS_VARvalue** SignedValue

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

EXAMPLE 25 - An example of test suite variables, with the comments indicating their intended use.

Test Suite Variables			
Name	**Type**	**Value**	**Comments**
state	IA5STRING	"idle"	Used to indicate the final stable state of the previous test case, if any, in order to help determine which pre-amble to use.

A.11.7.2 Test Case Variables

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

All test case variables to be used in a test suite shall be declared. The following information shall be provided for each variable declaration:

a) its name;

b) its type;

c) its initial value (if any),
where the initial value column is used when it is desired to assign an *optional* initial value to a test case variable at its point of declaration.

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

Test Case Variables			
Name	**Type**	**Value**	**Comments**
.	.	.	.
.	.	.	.
TC_VARidentifier	*Type* | *ReferenceType*	*Value*	*FreeText*
.	.	.	.
.	.	.	.

Proforma 8: Test Case Variables

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

SYNTAX DEFINITION:

- TC_VARdcls ::= **$Begin_TC_VARdcls** {TC_VARdcl}+ **$End_TC_VARdcls**
- TC_VARdcl ::= **$TC_VARdcl** TC_VARid TC_VARtype [TC_VARvalue] [Comment] **$End_TC_VARdcl**
- TC_VARid ::= **$TC_VARid** TC_VARidentifier
- TC_VARidentifier ::= Identifier
- TC_VARtype ::= **$TC_VARtype** (Type | ReferenceType)
- TC_VARvalue ::= **$TC_VARvalue** SignedValue

A.11.7.3 Binding of TTCN Variables

A.11.7.3.1 Binding of Test Suite Variables

Initially test suite variables are unbound. They may become bound (or be rebound) in the following contexts:

a) at the point of declaration;

b) when the test suite variable appears on the left-hand side of an assignment statement (clause A.15.10.4);

c) when the variable appears in a constraints reference (clause A.15.16).

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

A.11.7.3.2 Binding of Test Case Variables

Initially test case variables are unbound. They may become bound (or be rebound) in the following contexts:

a) at the point of declaration;

b) when the test case variable appears on the left-hand side of an assignment statement (clause A.15.10.4);

c) when the test case variable appears in a constraints reference (clause A.15.16).

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

A.11.8 PCO Declarations

This section lists the set of points of control and observation (PCOs) to be used in the test suite and explains where in the testing environment these PCOs exist. In accordance with part 1 of this standard* the number of PCOs relates to the test method: *one* PCO for the Remote and Coordinated test methods, and *two* PCOs for the Local, Distributed, Loop-back Relay and Transverse test methods.

TTCN behaviour statements specified for execution at the upper tester PCO shall not place requirements beyond those allowed by ISO 9646-2.

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

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

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

A SEND event is successful by being passed from the lower tester to the service provider (from upper tester to IUT).

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

NOTES -

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

2 - Connection Endpoint Identifiers (CEIds) within a single PCO are expected to be treated in an addendum to this standard*. It is expected that there will be two FIFO queues (one input, one output) for each CEId.

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

a) its name,

where the name shall be used in the behaviour descriptions to specify where particular events occur;

b) its type,

where the type is used to identify the layer boundary where the PCO is located;

c) its role,

which is an explanation of which type of tester is placed at the PCO. The predefined identifier **UT** indicates that the PCO is an upper tester PCO and **LT** specifies a lower tester PCO.

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

PCO Type Declarations			
Name	**Type**	**Role**	**Comments**
.	.	.	.
.	.	.	.
PCOidentifier	*PCOtypeIdentifier*	*UT ∣ LT*	*FreeText*
.	.	.	.
.	.	.	.

Proforma 9: PCO Declarations

<u>SYNTAX DEFINITION:</u>

- PCOdcls ::= **$Begin_PCOdcls** {PCOdcl}+ **$End_PCOdcls**
- PCOdcl ::= **$PCOdcl** PCOid PCOtypeId PCOrole [Comment] **$End_PCOdcl**
- PCOid ::= **$PCOid** PCOidentifier
- PCOidentifier ::= Identifier
- PCOtypeId ::= **$PCOtypeId** PCOtypeIdentifier
- PCOtypeIdentifier ::= Identifier
- PCOrole ::= **$PCOrole** (**UT** ∣ **LT**)

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

Example 26: Typical PCO Declarations

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

- at the same location (i.e. in the lower tester or in the upper tester);
- SAPs of the same service.

When a PCO corresponds to several SAPs the calling address (when initiating) or called address (when receiving) is used to identify the individual SAP.

EXAMPLE 27 - A typical example in which one PCO corresponds to several SAPs could be an Internet lower tester which uses one PCO representing all the subnetwork points of attachment for sending several Internet PDUs over different routes.

It should be noted that a PCO may not be related to a SAP at all. For instance, this could be the case when a layer is composed of sublayers (e.g. in the Application layer, or in the lower layers, where a subnetwork point of attachment is not a SAP).

A.11.9 Timer Declarations

A test suite may make use of several timers. Each timer has an associated length of time for its expiration.

The following information shall be provided for each timer:

a) the timer name,
where this name shall be unique within the declarations part;

b) the optional timer duration,
where the default duration of the timer may be a test suite parameter, test suite constant, or an explicit value; timer duration may be omitted if the value cannot be established prior to execution of the test suite.

c) the time units,
where the time units shall be one of the following:

1) **ps** (i.e. picosecond);

2) **ns** (i.e. nanosecond);

3) **us** (i.e. microsecond);

4) **ms** (i.e. millisecond);

5) **sec** (i.e. second);

6) **min** (i.e. minute).

Time units are determined by the test suite designer and are fixed at the time of specification. Different timers may use different units within the same test suite. A PICS or PIXIT entry requesting the IUT provider to indicate the duration of a timer shall include the units declared in the timer declarations proforma for that specific timer.

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

Timer Declarations			
Timer Name	**Duration**	**Units**	**Comments**
. . *TimerIdentifier* . .	. . *TimerDuration* . .	. . *TimeUnit.* .	. . *FreeText* . .

Proforma 10: Timer Declarations

SYNTAX DEFINITION:

- TIMERdcls ::= **$Begin_TIMERdcls** {TIMERdcl}+ **$End_TIMERdcls**
- TIMERdcl ::= **$TIMERdcl** TimerId [Duration] TimeUnit [Comment] **$End_TIMERdcl**
- TimerId ::= **$TimerId** TimerIdentifier
- TimerIdentifier ::= Identifier
- Duration ::= **$Duration** TimerDuration
- TimerDuration ::= Number | TS_PARidentifier | TS_CONSTidentifier
- TimeUnit ::= **$TimeUnit (ps | ns | us | ms | sec | min)**

Timer Declarations			
Timer Name	**Duration**	**Units**	**Comments**
wait	15	sec	General purpose wait
no_response	A	min	used to wait for IUT to connect or react to connection establishment, longer duration than general purpose wait. Gets value from PIXIT
delay_timer		ms	Duration to be established during execution of the test suite.

Example 28: Timer Declarations

A.11.10 TTCN Abbreviations Declarations

A.11.10.1 Introduction

This section defines any abbreviations that are to be used in the behaviour description column of the test suite. Abbreviations are used as a macro facility, performing simple textual substitution operations.

An abbreviation definition shall provide the following information:

a) an abbreviation identifier;

b) its expansion,

which is to be substituted for every occurrence of the identifier.

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

Abbreviation Declarations		
Abbreviation	**Expansion**	**Comments**
. . *AbbreviationIdentifier* . .	. . *FreeText* . .	. . *FreeText* . .

Proforma 11: Abbreviations

SYNTAX DEFINITION:

- Abbreviations ::= **$Begin_Abbreviations** {Abbreviation}+ **$End_Abbreviations**
- Abbreviation ::= **$Abbreviation** AbbreviationId Expansion [Comment] **$End_Abbreviation**
- AbbreviationId ::= **$AbbreviationId** AbbreviationIdentifier
- AbbreviationIdentifier ::= Identifier
- Expansion ::= **$Expansion** BoundedFreeText

Abbreviation Declarations		
Abbreviation	**Expansion**	**Comments**
CR	N_DATAind< NSDU ~ CR_TPDU>	CR denotes any N_DATA indication whose Network Service Data Unit is the encoding of a Connect Request Transport PDU. NOTE - for the "~" operator definition see clause 11.15.3.
CC	N_DATAreq <NSDU ^ CC_TPDU>	CC denotes any N_DATA request whose Network Service Data Unit is the encoding of a Connect Confirm Transport PDU. NOTE - for the "^" operator definition see clause 11.15.2.

Example 29: Abbreviation Declaration from a Transport Test Suite.

A.11.10.2 Scope and Expansion of Abbreviations

The following rules apply:

a) an abbreviation is an identifier that shall follow the syntax rules defined in the TTCN.MP. This means that an abbreviation is delimited by any character (symbol) not allowed in a TTCN identifier. Abbreviations shall be delimited by: white space, "!", "?", "(", ")", "[", "]", "<", ">", ":", "=", "+", "-", "*", "/", ",", ";", ".", "~", "^".

EXAMPLE 30 - !CR(X:=1) and ! CR (X:=1) are both legal uses of the abbreviation CR, which is delimited by the characters "!" and "(", and "!" and white space respectively.

b) abbreviations are not recursive - if an abbreviation appears in the expansion of that abbreviation it shall not be expanded (i.e. it is a one pass expansion);

EXAMPLE 31 - the CR in the CR_TPDU of the expansion above is not expanded.

c) an abbreviation may be used to replace any piece of text within a single TTCN statement of a behaviour tree. It shall only be used in a behaviour description column or a constraints reference column;

d) the test case writer shall ensure that when an abbreviation is expanded in a TTCN statement and/or constraints reference that the syntax of the resulting TTCN statement and/or constraints reference follows the TTCN.MP syntax for TTCN statements and/or constraints references;

e) tree indentation shall not be part of the expansion. The level of indentation is taken to be the same as the level of indentation of the TTCN statement in which the abbreviation appears.

A.11.11 ASP Type Declarations

A.11.11.1 Introduction

The purpose of this part of the abstract TTCN test suite is to declare the types of ASPs that may be sent or received at the declared PCOs. ASP type declarations may include ASN.1 type declarations, if appropriate. Normally, the declared information can be found in the appropriate service definition. However, declaring it explicitly allows the addition of commentary specific to testing and to a particular test suite, as well as providing for cases where no explicit OSI service definition exists (e.g. X.25).

A.11.11.2 Tabular ASP Type Declarations

The following information shall be supplied for each ASP:

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

b) the PCO type associated with the ASP,
where the PCO type shall be one of the PCO types used in the PCO declaration proforma. If only a single PCO is defined within a test suite, specifying the PCO type in an ASP type declaration is optional;

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

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,

where parameters may be of a type of arbitrarily complex structure; if a parameter is to be structured as a PDU, then its type may be stated as **PDU** to indicate that in the constraints for the ASP this parameter may be chained to a PDU constraints identifier;

if the name is a parameter group identifier then the type shall be stated as **GROUP** to indicate that the structure of the parameter group is to be found in the appropriate parameter group declaration.

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

ASP Type Declaration		
ASP Name: *ASPid&FullId*	**PCO Type:** *PCOtypeIdentifier*	**Comments:** *FreeText*
Service Parameter Information		
Parameter Name	**Type**	**Comments**
. . . *ASP_PARid&FullId* . . .	. . . *Type* | ***GROUP*** | ***PDU*** . . .	. . . *FreeText* . . .

Proforma 12: Abstract Service Primitive Type Declaration

SYNTAX DEFINITION:

- TTCN_ASPdcls ::= **$TTCN_ASPdcls** {TTCN_ASPdcl} {ASP_PARgroupDcl} **$End_TTCN_ASPdcls**
- TTCN_ASPdcl ::= **$Begin_TTCN_ASPdcl** ASPid [PCOtypeId] [Comment] [SPI] **$End_TTCN_ASPdcl**
- ASPid ::= **$ASPid** ASP_id&FullId
- ASP_id&FullId ::= ASPidentifier [FullIdentifier]
- ASPidentifier ::= Identifier
- FullIdentifier ::= "(" BoundedFreeText ")"
- PCOtypeId ::= **$PCOtypeId** PCOtypeIdentifier
- PCOtypeIdentifier ::= Identifier
- SPI ::= **$SPI** {ASP_PARdcl}+ **$End_SPI**
- ASP_PARdcl ::= **$ASP_PARdcl** ASP_PARid ASP_PARtype [Comment] **$End_ASP_PARdcl**
- ASP_PARid ::= **$ASP_PARid** ASP_PARid&FullId
- ASP_PARid&FullId ::= (ASP_PARidentifier [FullIdentifier]) | ASP_PARgroupIdentifier
- ASP_PARidentifier ::= Identifier
- ASP_PARtype ::= **$ASP_PARtype** (Type | **GROUP** | **PDU**)

EXAMPLE 32 - The figure below shows an example from the Transport Service [ISO 8072]. This could be part of the set of ASPs used to describe the behaviour of an abstract upper tester in a DS test suite for the Class 0 Transport. CDA,CGA and QOS are user defined types [ISO 8073].

ASP Type Declaration		
ASP Name: CONreq (T_CONNECTrequest)	**PCO Type:** TSAP	**Comments:**
Service Parameter Information		
Parameter Name	**Type**	**Comments**
Cda (Called Address)	CDA	... of upper tester
Cga (Calling Address)	CGA	... of lower tester
QoS (Quality of Service)	QOS	should ensure class 0 is used

Figure A.1: T_CONNECTrequest Abstract Service Primitive

A.11.11.3 Parameter Group Declarations

ASPs may be sub-structured by declaring one or more ASP parameters in a separate ASP parameter group declarations table. The parameters that are to be represented using this table shall be contiguous parameters in the original ASP type declaration. Values of parameter groups shall be specified in the constraints part.

This information shall be provided in the following proforma:

ASP Parameter Group Type Declaration		
Parameter Group Name: *ASP_PARgroupIdentifier*	**Comments:** *FreeText*	
Service Parameter Information		
Parameter Name	**Type**	**Comments**
. . *ASP_PARid&FullId* . .	. . *Type* | ***GROUP*** | ***PDU*** . .	. . *FreeText* . .

Proforma 13: Parameter Group Type Declaration

SYNTAX DEFINITION:

- TTCN_ASPdcls ::= **$TTCN_ASPdcls** {TTCN_ASPdcl} {ASP_PARgroupDcl} **$End_TTCN_ASPdcls**
- ASP_PARgroupDcl ::= **$Begin_ASP_PARgroupDcl** ASP_PARgroupId [Comment] SPI **$End_ASP_PARgroupDcl**
- ASP_PARgroupId ::= **$ASP_PARgroupId** ASP_PARgroupIdentifier
- ASP_PARgroupIdentifier ::= Identifier

A.11.11.4 ASP Type Declarations Using ASN.1

Where more appropriate, ASPs can be specified in ASN.1. This shall be achieved either by:

a) A precise reference to an ASN.1 ASP defined in an OSI* standard*;

b) An ASN.1 definition using the ASN.1 syntax as defined in ISO 8824.

This information shall be provided in the following proforma:

ASN.1 ASP Type Declaration	
ASP Name: *ASPid&FullId*	**PCO Type:** *PCOtypeIdentifier*
ASN.1 Definition or Reference	
. . . *ASN1Ref* | *ASN1Def* . . .	

Proforma 14: ASN.1 ASP Type Declaration

SYNTAX DEFINITION:

- ASN1_ASPdcls ::= **$ASN1_ASPdcls** {ASN1_ASPdcl} {ASN1_ASP_PARdcl} **$End_ASN1_ASPdcls**
- ASN1_ASPdcl ::= **$Begin_ ASN1_ASPdcl** ASPid [PCOtypeId] ASN1RefOrDef **$End_ASN1_ASPdcl**
- ASPid ::= **$ASPid** ASP_id&FullId
- ASP_id&FullId ::= ASPidentifier [FullIdentifier]
- ASPidentifier ::= Identifier
- FullIdentifier ::= "(" BoundedFreeText ")"

- PCOtypeId ::= **$PCOtypeId** PCOtypeIdentifier
- PCOtypeIdentifier ::= Identifier
- ASN1RefOrDef ::= **$ASN1RefOrDef** (ASN1Ref | ASN1Def) **$End_ASN1RefOrDef**
- ASN1Ref ::= **#REF** {SymbolsFromModule}+
- ASN1Def ::= {TypeAssignment}+ /* *Defined in ISO 8824* */

A.11.11.5 ASP Parameter Type Declarations Using ASN.1

ASP parameters specified in ASN.1 shall be defined in the following proforma:

ASN.1 ASP Parameter Type Declaration
ASP Parameter Name: *ASN1_ASP_PARidentifier*
ASN.1 Definition or Reference
. . . *ASN1Ref* | *ASN1Def* . . . .

Proforma 15: ASN.1 ASP Parameter Type Declaration

SYNTAX DEFINITION:

- ASN1_ASPdcls ::= **$ASN1_ASPdcls** {ASN1_ASPdcl} {ASN1_ASP_PARdcl} **$End_ASN1_ASPdcls**
- ASN1_ASP_PARdcl ::= **$Begin_ASN1_ASP_PARdcl** ASN1_ASP_PARid ASN1RefOrDef **$End_ASN1_ASP_PARdcl**
- ASN1_ASP_PARid ::= **$ASN1_ASP_PARid** ASN1_ASP_PARidentifier
- ASN1_ASP_PARidentifier ::= Identifier

A.11.12 PDU Type Declarations

A.11.12.1 Introduction

The purpose of this part of the abstract TTCN test suite is to declare the types of the PDUs that may be sent or received either directly or embedded in ASPs at the declared PCOs. PDU type declarations may include ASN.1 type declarations, if appropriate.

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

A.11.12.2 PDU Type Declarations Using Tables

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

a) its name,

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

b) the PCO type associated with the PDU,

if a PDU is sent or received only embedded in ASPs within the whole test suite, specifying the PCO type is optional; if only a single PCO is defined within a test suite, specifying the PCO type in a PDU type declaration is optional;

c) a list of the fields and field groups associated with the PDU.

NOTE - In order to be able to describe tests which exercise PDU encoding, it may be necessary to include fields (such as length indicators) into the PDU description, even though they may not be considered to be PDU fields in the protocol specification.

The following information shall be supplied for each field or field group:

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 optional field length,

   where a field is to be structured as a (higher-level) PDU, then its type may be stated as **PDU** to indicate that in the constraints for the PDU this parameter may be chained to another PDU constraint identifier;

   if the name is a field group identifier then the type shall be stated as **GROUP** to indicate that the structure of the field group is to be found in the appropriate field group declaration.

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

PDU Type Declaration		
PDU Name: *PDUid&FullId*	**PCO Type:** *PCOtypeIdentifier*	**Comments:** *FreeText*
PDU Field Information		
Field Name	**Type**	**Comments**
. . . *FIELDid&FullId* . . .	. . . *Type&Length* | ***GROUP* | *PDU*** . . .	. . . *FreeText* . . .

Proforma 16: Protocol Data Unit Type Declaration

SYNTAX DEFINITION:

- TTCN_PDUdcls ::= **$TTCN_PDUdcls** {TTCN_PDUdcl} {FIELDgroupDcl} **$End_TTCN_PDUdcls**
- TTCN_PDUdcl ::= **$Begin_TTCN_PDUdcl** PDUid [PCOtypeId] [Comment] PFI **$End_TTCN_PDUdcl**
- PDUid ::= **$PDUid** PDUid&FullId
- PDUid&FullId ::= PDUidentifier [FullIdentifier]
- PDUidentifier ::= Identifier
- PCOtypeId ::= **$PCOtypeId** PCOtypeIdentifier
- PCOtypeIdentifier ::= Identifier
- PFI ::= **$PFI** {FIELDdcl}+ **$End_PFI**
- FIELDdcl ::= **$FIELDdcl** FIELDid FIELDtype [Comment] **$End_FIELDdcl**
- FIELDid ::= **$FIELDid** FIELDid&FullId
- FIELDid&FullId ::= (FIELDidentifier [FullIdentifier]) | FIELDgroupIdentifier
- FIELDidentifier ::= Identifier
- FIELDgroupIdentifier ::= Identifier
- FIELDtype ::= **$FIELDtype** (Type&Length | **GROUP** | **PDU**)
- Type&Length ::= Type ["[" Length "]"]
- Length ::= SingleLength | RangeLength
- SingleLength ::= TCVid | Number | FIELDidentifier
- RangeLength ::= (TCVid | Number) To SingleLength

PDU Type Declaration		
PDU Name: INTC (Interrupt Confirm)	**PCO Type:** NSAP	**Comments:**
PDU Field Information		
Field Name	**Type**	**Comments**
GFI	BITSTRING	General Format Identifier
LCGN	BITSTRING	Logical Channel Group Number
LCN	BITSTRING	Logical Channel Identifier
PTI	OCTETSTRING	Packet Type Identifier
EXTRA	OCTETSTRING	To create long INTC packets

Example 33: A Typical PDU Type Declaration

A.11.12.3 Field Group Declarations

PDUs may be sub-structured by declaring one or more PDU fields in a separate PDU field group declarations table. The fields that are to be represented using this table shall be contiguous fields in the original PDU type declaration. Values of PDU fields shall be specified in the constraints part.

This information shall be provided in the following proforma:

PDU Field Group Type Declaration		
Field Group Name: *FIELDgroupIdentifier*	**Comments:** *FreeText*	
PDU Field Information		
Field Name	**Type**	**Comments**
. . . *FIELDid&FullId* . . .	. . . *TypeWithLength* | ***GROUP* | *PDU*** . .	. . . *FreeText* . . .

Proforma 17: PDU Field Group Declaration

SYNTAX DEFINITION:

- TTCN_PDUdcls ::= **$TTCN_PDUdcls** {TTCN_PDUdcl} {FIELDgroupDcl} **$End_TTCN_PDUdcls**
- FIELDgroupDcl ::= **$Begin_FIELDgroupDcl** FIELDgroupId [Comment] PFI **$End_FIELDgroupDcl**
- FIELDgroupId ::= **$FIELDgroupId** FIELDgroupIdentifier
- FIELDgroupIdentifier ::= Identifier

A.11.12.4 PDU Type declarations Using ASN.1

Where more appropriate PDUs can be specified in ASN.1. This is achieved either by:

a) a precise reference to an ASN.1 PDU defined in an OSI* standard*;

b) or an ASN.1 definition using the ASN.1 syntax as defined in ISO 8824.

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

ASN.1 PDU Type Declaration	
PDU Name: *PDUid&FullId*	**PCO Type:** *PCOtypeIdentifier*
ASN.1 Definition or Reference	
. . . *ASN1Ref* | *ASN1Def* . . .	

Proforma 18: ASN.1 PDU Type Definition

SYNTAX DEFINITION:

- ASN1_PDUdcls ::= **$ASN1_PDUdcls** {ASN1_PDUdcl} {ASN1_FIELDdcl} **$End_ASN1_PDUdcls**
- ASN1_PDUdcl ::= **$Begin_ASN1_PDUdcl** PDUid [PCOtypeId] ASN1RefOrDef **$End_ASN1_PDUdcl**
- PDUid ::= **$PDUid** PDUid&FullId
- PDUid&FullId ::= PDUidentifier [FullIdentifier]
- PDUidentifier ::= Identifier
- PCOtypeId ::= **$PCOtypeId** PCOtypeIdentifier
- PCOtypeIdentifier ::= Identifier
- ASN1RefOrDef ::= **$ASN1RefOrDef** (ASN1Ref | ASN1Def) **$End_ASN1RefOrDef**
- ASN1Ref ::= **#REF** {SymbolsFromModule}+
- ASN1Def ::= {TypeAssignment}+ /* *Defined in ISO 8824* */

ASN.1 PDU Type Declaration	
PDU Name: F_INIT (F_INITIALIZE_response)	**PCO Type:**
ASN.1 Definition or Reference	
SEQUENCE { state_result State_result DEFAULT success, action_result Action_Result DEFAULT success protocol_id Protocol_Version .... etc. }	

Example 34: FTAM ASN.1 Declaration Through Definition.

A.11.12.5 PDU Field Type Declarations Using ASN.1

PDU fields specified in ASN.1 shall be defined using the following proforma:

ASN.1 PDU Field Type Declaration
PDU Field Name: *ASN1_FIELDidentifier*
ASN.1 Definition or Reference
. . . *ASN1Ref* | *ASN1Def* . . .

Proforma 19: ASN.1 PDU Field Type Declaration

SYNTAX DEFINITION:

- ASN1_PDUdcls ::= **$ASN1_PDUdcls** {ASN1_PDUdcl} {ASN1_FIELDdcl} **$End_ASN1_PDUdcls**
- ASN1_FIELDdcl ::= **$Begin_ASN1_FIELDdcl** ASN1_FIELDid ASN1RefOrDef **$End_ASN1_FIELDdcl**
- ASN1_FIELDid ::= **$ASN1_FIELDid** ASN1_FIELDidentifier
- ASN1_FIELDidentifier ::= Identifier

When an ASN.1 reference is used the free text entry shall be preceded by the keyword **#REF**. This applies throughout TTCN where ASN.1 references are used.

A.12 The Constraints Part

A.12.1 Introduction

It is necessary to specify, in detail, the values of ASP parameters and PDU fields . These encodings shall be described using either:

- the Tabular Method (clause A.13); or
- the ASN.1 Modular Method (clause A.14).

Reference to particular constraints is made in the constraints reference column of the various dynamic behaviour tables.

A.12.2 Using Test Suite Parameters and Constants in Constraints

Test suite parameters and test suite constants that have been declared in the declarations part of the test suite may be used in the constraints part. The following rules shall be observed, with respect to the direction of the PDU:

a) sent PDUs:

- the value of the test suite parameter or test suite constant is sent;

b) received PDUs:

- the value of the test suite parameter or test suite constant shall be the value received if the constraint is to apply;

Neither test suite- nor test case variables shall be used in constraints, unless passed as parameters.

NOTE - Use of test suite and test case variables in constraints is expected to be treated in an addendum to this standard*

A.12.3 Using Special Value Symbols in Constraints

A.12.3.1 Introduction

Three special symbols ("-", "?", "*") may be used in constraints to specify "don't care" values and to explicitly omit fields.

A.12.3.2 Parameter or Field Constraints

a) The OMIT symbol, "-", ("dash"), specified as the value of a parameter or field in a constraint, indicates that the parameter or field shall be absent in a SEND event and in a RECEIVE event, if the constraint is to match.

NOTE - How the absence of a parameter or field is expressed is a local encoding matter.

b) The ANY_SINGLE_VALUE symbol, "?", in a RECEIVE event, replacing a value in a single parameter or field of a defined type in a constraint, indicates that the constraint will match with any received value in that parameter or field and belonging to the type.

c) The ANY_OR_OMIT symbol, "*", in a RECEIVE event, replacing a value in a single parameter or field of a defined type in a constraint, indicates that the constraint will match with any received value belonging to the type, and/or if the parameter or field is omitted.

If either of the two symbols "?" and "*" are used in a constraint called by a SEND event, then the fields/parameters in which they occur shall be explicitly overwritten with definite values before the event is to be sent.

The ASN.1 equivalent symbols are:

OMIT	instead of "-" as in TTCN.GR
"?"	as in TTCN.GR
"*"	as in TTCN.GR

A.12.3.3 Pattern Matching in a String

The special characters can be used, in a very similar manner, to indicate special conditions of acceptance of a character string, BITSTRING, HEXSTRING, or OCTETSTRING in a constraint referenced by a RECEIVE event, or in an expression in which two strings are compared (using "=" or "<>"). Inside a string, a "?" in place of a single unit means that any single unit value is accepted; a "*" means that none, or any number of units is accepted. The symbol "*" shall match the longest sequence of units possible, according to the pattern as specified by the symbols surrounding the "*".

In a character string, when the symbols "?" or "*" are needed within the character string as characters, this shall be indicated by "\?" or "*". The character "\" itself shall be written "\\".

A.13 Specifying Constraints Using the Tabular Method

A.13.1 Introduction

This clause describes the tabular method to define constraints on PDUs and ASPs. First it is discussed how single constraint tables can be used to define constraints on flat PDUs or ASPs. Next a technique to combine multiple constraints definitions in one table is introduced. Finally it is pointed out how the PDU field or ASP parameter groups, as defined in the declaration of structured PDUs or ASPs, can be used to define structured constraints.

A.13.2 Tables for Single Constraints on PDUs

A.13.2.1 Constraint Declaration

In the TTCN tabular form a constraint is defined by specifying a value and optional length for each PDU field. This information shall be provided in the format shown in the following proforma:

PDU Constraint Declaration	
PDU Name: *PDUidentifier*	**Constraint Name:** *CONSid&PARlist*
Field Name	**Value**
. . *FIELDidentifier* . .	. . *Value&Length* . .
Comments: . . *FreeText* . .	

Proforma 20: PDU Constraint Declaration

SYNTAX DEFINITION:

- TTCN_PDUconstraints ::= **$TTCN_PDUconstraints** {TTCN_PDUconstraint} {FIELDgroupConstraint} **$End_TTCN_PDUconstraints**
- TTCN_PDUconstraint ::= **$Begin_TTCN_PDUconstraint** PDUid CONSid {FVI}+ [Comment] **$End_TTCN_PDUconstraint**
- PDUid ::= **$PDUid** PDUid&FullId
- PDUid&FullId ::= PDUidentifier [FullIdentifier]
- PDUidentifier ::= Identifier
- CONSid ::= **$CONSid** CONSid&PARlist
- CONSid&PARlist ::= CONSidentifier [FormalPARlist]
- CONSidentifier ::= ConstraintIdentifier {Dot ConstraintIdentifier}
- Dot ::= "."
- ConstraintIdentifier ::= Identifier
- FVI ::= **$FVI** {PDU_VALdcl}+ **$End_FVI**
- PDU_VALdcl ::= **$PDU_VALdcl** FIELDid CONSvalue **$End_PDU_VALdcl**
- FIELDid ::= **$FIELDid** FIELDid&FullId
- FIELDid&FullId ::= (FIELDidentifier [FullIdentifier]) | FIELDgroupIdentifier
- FIELDidentifier ::= Identifier
- FIELDgroupIdentifier ::= Identifier

- CONSvalue ::= **$CONSvalue** Value&Length
- ConstraintValue ::= ExtValue | TS_PARidentifier | TS_CONSTidentifier | PARidentifier | Relop SignedValue | VALlist | Range | CONSidentifier [RestrictedCrefList]
- RestrictedCrefList ::= "(" (SignedValue | TS_PARidentifier | TS_CONSTidentifier | PARidentifier) { Comma (SignedValue | TS_PARidentifier | TS_CONSTidentifier | PARidentifier)}")"
- Value&Length ::= ConstraintValue ["[" LengthConstraint "]"]
- LengthConstraint ::= SingleLengthConstraint | RangeLengthConstraint
- SingleLengthConstraint ::= Number | TS_PARidentifier | TS_CONSTidentifier | FIELDidentifier | PARidentifier
- RangeLengthConstraint ::= (Number | TS_PARidentifier | TS_CONSTidentifier) To SingleLengthConstraint

When defining constraints on an ASP or PDU, and the original ASP or PDU was defined as having both a short name and a full identifier, the constraint need not repeat the full identifier.

A.13.2.2 Constraint Values

Each field entry in the field name column shall have been declared in the relevant ASP or PDU type declaration. When defining constraints on an ASP or PDU, and any of the original ASP parameters or PDU fields was defined as having both a short name and full identifier, the constraint need not repeat the full identifier. Values assigned to each field shall be of the type specified in the ASP or PDU declaration. Optionally, the length of a PDU field may be specified. The value may be an explicit value or, where the type definition allows it, a range or list of values (e.g. "a" .. "z"). In the case of numeric types a relational operator may be used (e.g. >10). Test suite parameters and test suite constants may also be used as constraint values. Special symbols may also be used in constraints as may parameter or field group names.

PDU Constraint Declaration	
PDU Name: PDU_B	**Constraint Name:** C1
Field Name	**Value**
FLD1	>3
FLD2	TRUE
FLD3	"A STRING"

Example 35: A Constraint, called C1, on the PDU called PDU_B

A.13.2.3 Base Constraints

For every PDU type declaration at least one base constraint shall be specified. A base constraint specifies a set of base, or default, values for each and every field defined in the appropriate declaration. There may be any number of base constraints for any particular PDU.

When a subsequent constraint is defined for a PDU that has a base constraint, any fields not re-specified in the modified constraint will default to the values specified in the base constraint. The name of the modified constraint will be the concatenation of the name of the base constraint and the modified constraint, separated by a dot ("."). A base constraint is recognizable in that it does not have a dot (".") in its name. There is no limit on the depth of modified constraints.

EXAMPLE 36 - C0.C1.C2.C3

Value fields may be explicitly omitted if the symbol "-" is present as a field value. The rules for building a constraint from a base constraint are thus:

a) field and value not specified in constraint ⇒ value in parent constraint used (i.e. the value is inherited);

b) field and value specified in the constraint ⇒ specified value replaces the inherited value;

c) field has "-" specified as a value ⇒ omit this field and value from the constraint.

EXAMPLE 37 - Suppose that we have the following PDU type declaration:

PDU Type Declaration		
PDU Name: PDU_A	**PCO Type:**	**Comments:** This is the declaration of the protocol data unit named PDU_A
PDU Field Information		
Field Name	**Type**	**Comments**
FLD1 FLD2 FLD3 FLD4	INTEGER HEXSTRING BITSTRING BOOLEAN	

A base constraint for PDU_A could be:

PDU Constraint Declaration	
PDU Name: PDU_A	**Constraint Name:** C0
Field Name	**Value**
FLD1 FLD2 FLD3 FLD4	0 'FF'H '00'B TRUE
Comments:	

A modified constraint to the base constraint C0 could be:

PDU Constraint Declaration	
PDU Name: PDU_A	**Constraint Name:** C0.C1
Field Name	**Value**
FLD1	1
Comments: FLD1 has value 1	

The constraint C0.C1 is exactly the same as C0 except that the value of FLD1 is now the integer value '1'.

We can further build on C0.C1:

PDU Constraint Declaration	
PDU Name: PDU_A	**Constraint Name:** C0.C1.C2
Field Name	**Value**
FLD2 FLD3	- ?
Comments: Field FLD2 omitted. Any legal value accepted for FLD3	

The constraint C0.C1.C2 specifies the following values for PDU_A:

FLD1 = 1
FLD2 is omitted
FLD3 = any (legal) value is accepted
FLD4 = TRUE

Because of the requirement for uniqueness of constraint identifiers and components of dotted identifiers within the test suite a reference to a PDU constraint can be made from a behaviour description using the last identifier in the dotted identifier. However, the full dotted identifier shall still appear in the constraint's name field.

EXAMPLE 38 - the identifiers C0, C1 and C2 are unique. The constraints reference PDU_A [C1] is the same as saying PDU_A [C0.C1] and the constraints reference PDU_A [C2] is the same as saying PDU_A [C0.C1.C2]. The constraints references can be further shortened by omitting PDU_A, simply specifying C0, C1 or C2 as appropriate.

A.13.2.4 Parameterized Constraints

Constraint values may be parameterized. In such cases the constraint name shall be followed by a parameter list and the parameterized fields shall have these parameters as values. Each parameter name shall be followed by a colon and then the name of the parameter's type.

If more than one parameter of the same type is used, the parameter 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 sub-list shall be followed by a colon and then the name of the parameter sub-list's type. When more than one parameter and type pair (or parameter sub-list and type pair) is used, the pairs shall be separated from each other by semicolons.

Test suite and test case variables shall only be used as actual parameters to a constraint in a constraints reference made in a dynamic behaviour description. Actual values, test suite parameters or test suite constants may, however, be used as actual parameters to a constraint in constraint references both in dynamic behaviour descriptions and in constraint declarations. When a test suite parameter, or test suite constant, or test suite or test case variable is used as an actual parameter in a constraint, then the following rules apply:

a) in the case of a SEND event the value of the actual parameter is sent;

b) in the case of a RECEIVE event the value of the actual parameter shall be the same as the value received if the constraint is to apply.

EXAMPLE 39 - The parameterized base constraint C0. A possible invocation of C0 from a test case or test step may be: PDU_A [C0 (0, TRUE)].

PDU Constraint Declaration	
PDU Name: PDU_A	**Constraint Name:** C0 (P1:INTEGER, P2:BOOLEAN)
Field Name	**Value**
FLD1 FLD2 FLD3 FLD4	P1 'FF'H '00'B P2
Comments:	

A.13.3 Tables for Multiple Constraints on PDUs

A.13.3.1 Introduction

In cases where a constraint contains only a few fields, or when there are only a small number of constraints, the constraints may be collected in a list and presented in the multiple version of the constraints table proforma:

PDU Constraints Declarations				
PDU Name: *PDUidentifier*				
Constraint Name	**Field Name**			**Comments**
	FIELDidentifier$_1$		*FIELDidentifier*$_n$	
CONSid&PARlist$_1$	*Value&Length*$_{1,1}$	....	*Value&Length*$_{1,n}$	*FreeText*$_1$
CONSid&PARlist$_2$	*Value&Length*$_{2,1}$		*Value&Length*$_{2,n}$	*FreeText*$_2$
⋮	⋮	....	⋮	⋮
CONSid&PARlist$_m$	*Value&Length*$_{m,1}$		*Value&Length*$_{m,n}$	*FreeText*$_m$

Proforma 21: Multiple PDU Constraints Declarations

The multiple constraints proforma has field names across the top of the table, and different instances of the PDU constraints in rows within the table. If there are *n* fields in the PDU type declaration then there shall be *n* field columns in the multiple constraint table.

EXAMPLE 40 - Constraints using the multiple constraints proforma. Given PDU_B's declaration to be:

PDU Type Declaration		
PDU Name: PDU_B	**PCO Type:** XSAP	**Comments:**
PDU Field Information		
Field Name	**Type**	**Comments**
FLD1 FLD2 FLD3	INTEGER BOOLEAN IA5STRING	

the constraints on PDU_B using the multiple constraints proforma could be:

PDU Constraints Declarations				
PDU Name: PDU_B				
Constraint Name	**Field Name**			**Comments**
	FLD1	FLD2	FLD3	
CN1	>3	TRUE	"A string"	
CN2	(4,5,6)	FALSE	"A string"	
CN3	0	?	-	

The constraints reference in the dynamic part might then contain entries such as PDU_B[CN1] and PDU_B[CN2]

EXAMPLE 41 - An example of the inheritance mechanism using the multiple constraint proforma:

PDU Constraints Declarations					
PDU Name: PDU_A					
Constraint Name	**Field Name**				**Comments**
	FLD1	FLD2	FLD3	FLD4	
CN0	0	'FF'H	'00'B	TRUE	
CN0.CN1	1				
CN0.CN1.CN2		-	?		

Constraint CN0 is the base constraint in which all fields have a value specified. Constraint CN1 is the modified constraint derived from base constraint CN0. The modified constraint CN0.CN1 is exactly the same as C0 except that the value of FLD1 is the INTEGER value '1'.

The constraint CN0.CN1.CN2 further modifies the constraint built by CN0.CN1 to specify the following values for PDU_A:

FLD1 = 1

FLD2 is omitted

FLD3 = any (legal) value is accepted

FLD4 = TRUE

Given these constraint definitions, the constraints reference column in the dynamic part could contain entries such as PDU_-A[CN0], CN1 and PDU_A[CN2].

A.13.3.2 Parameterized Multiple Constraints

Multiple constraints may also be parameterized. In such cases the parameterized constraint shall have its constraint name followed by the parenthesized list of parameter names and types. The parameter names shall also appear in the column entry of the corresponding fields.

EXAMPLE 42 - A parameterized multiple constraint. The invocation of the constraints on PDU_X in a test step may be made as follows: S1, S2, S3, S4, S5(0), S5(1) or S5(Var) where Var is a test case or test suite variable.

PDU Constraints Declarations			
PDU Name: PDU_X			
Constraint Name	**Field Name**		**Comments**
	P1	P2	
S1	0	0	
S2	0	1	
S3	1	0	
S4	1	1	
S5(A:INTEGER)	1	A	

A.13.4 Structured Constraints for PDUs

A.13.4.1 Introduction

The field grouping mechanism allows one or more contiguous fields of a PDU to be combined and referred to through a single reference. That is, a group of field identifiers may be replaced by a field group identifier and the corresponding group of fields are represented by a single field group constraint identifier. Field group identifiers shall be declared in the declarations section of the abstract test suite. The expansion of field groups is equivalent to a macro expansion.

A.13.4.2 Single Field Group Constraints

Single field group constraints information shall be provided in the following proforma:

PDU Field Group Constraint Declaration	
Field Group Name: *FIELDgroupIdentifier*	**Constraint Name:** *CONSid&PARlist*
Field Name	**Value**
. . *FIELDidentifier* . .	. . *Value&Length* . .
Comments: . . *FreeText* . .	

Proforma 22: PDU Field Group Constraint Declaration

It should be noted that field group identifiers are not legitimate PDU fields, they are only references (pointers) to a field group (i.e. lists of fields). Thus, field group identifiers shall not be used in assignment statements nor in Boolean expressions.

SYNTAX DEFINITION:

- TTCN_PDUconstraints ::= **$TTCN_PDUconstraints** {TTCN_PDUconstraint} {FIELDgroupConstraint} **$End_TTCN_PDUconstraints**
- FIELDgroupConstraint ::= **$Begin_FIELDgroupConstraint** FIELDgroupId CONSid {FVI}+ [Comment] **$End_FIELDgroupConstraint**
- FIELDgroupId ::= **$FIELDgroupId** FIELDgroupIdentifier
- FIELDgroupIdentifier ::= Identifier
- CONSid ::= **$CONSid** CONSid&PARlist
- CONSid&PARlist ::= CONSidentifier [FormalPARlist]
- FVI ::= **$FVI** {PDU_VALdcl}+ **$End_FVI**
- PDU_VALdcl ::= **$PDU_VALdcl** FIELDid CONSvalue **$End_PDU_VALdcl**
- FIELDid ::= **$FIELDid** FIELDid&FullId
- FIELDid&FullId ::= (FIELDidentifier [FullIdentifier]) | FIELDgroupIdentifier

A.13.4.3 Multiple Field Group Constraints

Alternatively, field group constraints may be provided in a table for multiple field groups. Multiple field group constraints shall be provided in the following proforma:

PDU Field Group Constraints Declarations				
Field Group Name: *FIELDgroupIdentifier*				
Constraint Name	**Field Name**			**Comments**
	FIELDidentifier$_1$	....	*FIELDidentifier*$_n$	
CONSid&PARlist$_1$	*Value&Length*$_{1,1}$	....	*Value&Length*$_{1,n}$	*FreeText*$_1$
CONSid&PARlist$_2$	*Value&Length*$_{2,1}$		*Value&Length*$_{2,n}$	*FreeText*$_2$
⋮	⋮	....	⋮	⋮
CONSid&PARlist$_m$	*Value&Length*$_{m,1}$	....	*Value&Length*$_{m,n}$	*FreeText*$_m$

Proforma 23: Multiple PDU Field Group Constraints Declarations

EXAMPLE 43 - The PDU_Y consists of five fields named Y1 through Y5. The fields Y1, Y2 and Y3 have been combined into the field group called A. In the following, figure A.2 shows the constraints defined on PDU_Y. Figures A.3, A.4 and A.5 convey exactly the same information; figure A.12- shows the field group A's constraint specification using the single constraint proforma, while figure A.12+ shows A's constraint using the multiple constraint proforma. Both figures also use the inheritance mechanism.

For the following figures, it can be seen that if the constraint YY1 was used, the values for field Y1 through Y5 would be 0,0,0,0,1 respectively, where the values for fields Y1 through Y3 are derived from the field group A using constraint A1. If the constraint YY2 was used, the values for Y1 through Y5 would be 0,3,0,1,0 respectively, where the values for fields Y1 through Y3 are derived from the field group A using constraint A2.

PDU Constraints Declarations				
PDU Name: PDU_Y				
Constraint Name	**Field Name**			**Comments**
	A	Y4	Y5	
YY1	A1	0	1	
YY2	A2	1	0	
YY3	A2	0	1	

Figure A.2: A PDU Constraints Table that Uses a Field Group

A1 is a base constraint of field group A:

PDU Field Group Constraint Declaration	
Field Group Name: A	**Constraint Name:** A1
Field Name	**Value**
Y1 Y2 Y3	0 0 0
Comments:	

Figure A.3: Field Group A's Constraint A1 in the Single Form

The field group constraint, A2, is a modified constraint derived from A1:

PDU Field Group Constraint Declaration	
Field Group Name: A	**Constraint Name:** A1.A2
Field Name	**Value**
Y2	3
Comments:	

Figure A.4: Field Group A's Constraint A2 in the Single Form

PDU Field Group Constraints Declarations				
Field Group Name: A				
Constraint Name	**Field Name**			**Comments**
	Y1	Y2	Y3	
A1	0	0	0	
A1.A2		3		

Figure A.5: Field Group A's Constraints A1 and A2 in the Multiple Form

When using field groups within PDU constraint declarations, each field name used within the field group declaration must exactly match the name (or short name, if both the short name and full name were defined) of the PDU field which it represents from the original PDU type declaration.

A.13.5 Tables for Single ASP Constraints

Single ASP constraint tables are very similar to single PDU constraint tables. The parameter values for single ASP constraints shall be specified in the format shown in the following proforma:

ASP Constraint Declaration	
ASP Name: *ASPidentifier*	**Constraint Name:** *CONSid&PARlist*
Parameter Name	**Value**
. . *ASP_PARid&FullId* . .	. . *Value&Length* . .
Comments: . . *FreeText* . .	

Proforma 24: ASP Constraint Declaration

This proforma is used for ASPs and their parameters in the same way that PDU constraint declaration proforma is used for PDUs and their fields. Thus, for further information see clause A.13.2.

<u>SYNTAX DEFINITION:</u>

- TTCN_ASPconstraints ::= **$TTCN_ASPconstraints** {TTCN_ASPconstraint} {ASP_PARgroupConstraint} **$End_TTCN_ASPconstraints**
- TTCN_ASPconstraint ::= **$Begin_TTCN_ASPconstraint** ASPid CONSid {PVI}+ [Comment] **$End_TTCN_ASPconstraint**
- PVI ::= **$PVI** {ASP_VALdcl}+ **$End_PVI**
- ASP_VALdcl ::= **$ASP_VALdcl** ASP_PARid CONSvalue **$End_ASP_VALdcl**

A.13.6 Tables for Multiple ASP Constraints

Multiple ASP constraint tables are very similar to multiple PDU constraint tables. The parameter values for multiple ASP constraints shall be specified in the following proforma:

ASP Constraints Declarations				
ASP Name: *ASPidentifier*				
Constraint Name	**Parameter Name**			**Comments**
	ASP_PARidentifier$_1$		*ASP_PARidentifier*$_n$	
CONSid&PARlist$_1$	*Value&Length*$_{1,1}$	....	*Value&Length*$_{1,n}$	*FreeText*$_1$
CONSid&PARlist$_2$	*Value&Length*$_{2,1}$		*Value&Length*$_{2,n}$	*FreeText*$_2$
⋮	⋮	....	⋮	⋮
CONSid&PARlist$_m$	*Value&Length*$_{m,1}$		*Value&Length*$_{m,n}$	*FreeText*$_m$

Proforma 25: Multiple ASP Constraints Declarations

This proforma is used for ASPs and their parameters in the same way that PDU constraint declaration proforma is used for PDUs and their fields. Thus, for further information see clause A.13.3.

A.13.7 Structured Constraints on ASPs

Structured constraints on ASPs using parameter groups are similar to structured PDU constraints using field groups. The constraints on parameter groups shall be specified in the format shown in the following proforma:

ASP Parameter Group Constraint Declaration	
Parameter Group Name: *ASP_PARgroupIdentifier*	**Constraint Name:** *CONSid&PARlist*
Parameter Name	**Value**
. . *ASP_PARid&FullId* . .	. . *Value&Length* . .
Comments: . . *FreeText* . .	

Proforma 26: ASP Parameter Group Constraint Declaration

This proforma is used for ASP parameter groups and their parameters in the same way that the PDU field group constraint proforma is used for PDUs and their field groups. For further information see clause A.13.4.

SYNTAX DEFINITION:

- TTCN_ASPconstraints ::= **$TTCN_ASPconstraints** {TTCN_ASPconstraint} {ASP_PARgroupConstraint} **$End_TTCN_ASPconstraints**

- ASP_PARgroupConstraint ::= **$Begin_ASP_PARgroupConstraint** ASP_PARgroupId CONSid {PVI}+ [Comment] **$End_ASP_PARgroupConstraint**
- CONSid ::= **$CONSid** CONSid&PARlist

Alternatively, field group constraints may be provided in a table for multiple field groups. Multiple field group constraints shall be provided in the following proforma:

ASP Parameter Group Constraints Declarations				
Parameter Group Name: *ASP_PARgroupIdentifier*				
Constraint Name	**Parameter Name**			**Comments**
	ASP_PARidentifier$_1$		*ASP_PARidentifier*$_1$	
CONSid&PARlist$_1$	*Value&Length*$_{1,1}$	. . . . .	*Value&Length*$_{1,n}$	*FreeText*$_1$
CONSid&PARlist$_2$	*Value&Length*$_{2,1}$		*Value&Length*$_{2,n}$	*FreeText*$_2$
⋮	⋮	. . . .	⋮	⋮
CONSid&PARlist$_m$	*Value&Length*$_{m,1}$	. . . . .	*Value&Length*$_{m,n}$	*FreeText*$_m$

Proforma 27: Multiple Parameter Group Constraints Declarations

This proforma is used for ASP parameter groups and their parameters in the same way that the PDU field group constraints proforma is used for PDUs and their field groups. For further information see clause A.13.4.

A.13.8 Field Length Specifications

The length of a field may be specified in a single or multiple constraint table by including a number or range in the value column. The length may also be determined dynamically from the contents of another field of the same constraint or as a parameter passed into the constraint. The following example illustrates the various methods of defining a length of a PDU field.

The units for the length specification shall be interpreted as shown below for TTCN predefined types and their assignment compatible types:

Table A.4: Units of Length Used in Field Length Specifications

Type	**Units of Length**
INTEGER	bits
BOOLEAN	bits
BITSTRING	bits
HEXSTRING	hex digits
OCTETSTRING	octets
character string	characters

EXAMPLE 44 - An example of length specification:

PDU Type Declaration		
PDU Name: PDU_Z	**PCO Type:**	**Comments:**
PDU Field Information		
Field Name	**Type**	**Comments**
FLD1 FLD2 FLD3	INTEGER [8] HEXSTRING [FLD1] OCTETSTRING	

Where a constraint on PDU_Z could be:

PDU Constraints Declarations				
PDU Name: PDU_Z				
Constraint Name	**Field Name**			**Comments**
	FLD1	FLD2	FLD3	
CN0	?	?	? [5]	CN0 uses a constant length
CN1	?	?	? [1..3]	CN1 specifies a length of range 1 to 3
CN2(X:INTEGER)	?	?	? [X]	CN2 specifies a variable length
CN3(Y:INTEGER)	?	? [3]	? [Y]	

Explanation of PDU_Z constraint:

Constraint CN0 allows the length field of field FLD1 to be an 8 bit integer as declared in the PDU type declaration and field FLD2 to have a length based on the contents of field FLD1. Field FLD3 will be 5 octets long. Constraint CN1 uses a range for the length of field FLD3 indicating that FLD3 may be between 1 and 3 octets long. Constraint CN2 uses a parameter passed to the constraint that will be used to indicate the length of FLD3. CN3 illustrates a more complex usage of length, where field FLD2 is of length 3 hexdigits (overriding the declaration) and field FLD3 uses the parameter Y.

A.14 ASN.1 Constraints

A.14.1 Introduction

This clause describes a method to define constraints in ASN.1, in a way similar to the definition of tabular constraints. The normal ASN.1 value definition is extended to allow use of wild cards and ranges. Mechanisms to replace or omit parts of ASN.1 constraints, to be used in modified constraints, are defined.

ISO 8824 "Specification of Abstract Syntax Notation One" contains mechanisms to define subtypes of existing types. These subtypes can be used for the same purposes as TTCN tabular constraints. The approach taken is, however, very different from that used in tabular constraints. Therefore these subtypes are treated in TTCN as ASN.1 types and can not be referenced from the test case/step/default table constraint reference column.

A.14.2 ASN.1 Constraint Tables

The ASN.1 constraint shall be named so that they can be referenced in the dynamic part. It is possible to parameterize ASN.1 constraints, in which case the constraint name shall be followed by a formal parameter list.

ASN.1 constraints are typed. It is required for documentation purposes to mention this type in the header of the ASN.1 constraint table. The type can be any ASN.1 type as defined in Type in ISO 8824.

If an ASN.1 constraint definition is a modification of an existing ASN.1 constraint, the name of the ASN.1 constraint that is taken as the basis of this modification shall be referenced in the table.

Single ASN.1 ASP constraint declarations shall be specified in the format shown in the following proforma:

ASN.1 ASP Constraint Declaration	
ASP Name: *ASPidentifier*	**Constraint Name:** *CONSid&PARlist*
ASN.1 Value	
. . . *ASN1_ConstraintValue* . . .	

Proforma 28: ASN.1 ASP Constraint Declaration

SYNTAX DEFINITION:

- ASN1_ASPconstraints ::= **$ASN1_ASPconstraints** {ASN1_ASPconstraint} {ASN1_ASP_PARconstraint} **$End_ASN1_ASPconstraints**
- ASN1_ASPconstraint ::= **$Begin_ASN1_ASPconstraint** ASPid CONSid ASN1_Def **$End_ASN1_ASPconstraint**
- ASN1_Def ::= **$ASN1_Def** ASN1_ConstraintValue **$End_ASN1_Def**

Single ASN.1 PDU constraint declarations shall be specified in the format shown in the following proforma:

ASN.1 PDU Constraint Declaration	
PDU Name: *PDUidentifier*	**Constraint Name:** *CONSid&PARlist*
ASN.1 Value	
. . . *ASN1_ConstraintValue* . . .	

Proforma 29: ASN.1 PDU Constraint Declaration

The body of the single ASN.1 ASP and PDU constraints tables contains the ASN.1 constraint definition.

SYNTAX DEFINITION:

- ASN1_PDUconstraints ::= **$ASN1_PDUconstraints** {ASN1_PDUconstraint} {ASN1_FIELDconstraint} **$End_ASN1_PDUconstraints**
- ASN1_PDUconstraint ::=**$Begin_ASN1_PDUconstraint** PDUid CONSid ASN1_Def **$End_ASN1_PDUconstraint**
- ASN1_Def ::= **$ASN1_Def** ASN1_ConstraintValue **$End_ASN1_Def**

The following proformas shall be used for multiple ASN.1 ASP and PDU constraints declarations respectively:

ASN.1 ASP Constraints Declarations	
ASP Name: *ASPidentifier*	
Constraint name	**ASN.1 Value**
CONSid&PARlist	*ASN1_ConstraintValue*
⋮	⋮
CONSid&PARlist	*ASN1_ConstraintValue*

Proforma 30: Multiple ASN.1 ASP Constraints Declarations

ASN.1 PDU Constraints Declarations	
PDU Name: *PDUidentifier*	
Constraint name	**ASN.1 Value**
CONSid&PARlist	*ASN1_ConstraintValue*
⋮	⋮
CONSid&PARlist	*ASN1_ConstraintValue*

Proforma 31: Multiple ASN.1 PDU Constraints Declarations

A.14.3 ASN.1 Constraint Value Definition

A.14.3.1 Introduction

An ASN.1 constraint value definition consists of two parts. The first part, Encoding, is optional. It can be used to define encoding directives. The second part, the ASN.1 value, is the actual description of the constraint.

SYNTAX DEFINITION:

- ASN1_ConstraintValue ::= [Encoding] ASN1_ValueTemplate

A.14.3.2 Encoding Directives

The ASN.1 basic encoding rules, ISO 8825, offer multiple ways to encode length. By giving encoding directives, the test suite specifier can restrict the possible ways of encoding length offered by ISO 8825.

If any legal encoding of the value is acceptable, the specification of encoding constraints for the value may be omitted. Otherwise, the encoding specification shall have the following syntax:

SYNTAX DEFINITION:

- Encoding ::= ["[" LengthEncoding "]"
- LengthEncoding ::= **LI** | **SD** | **LD** | **LD** Number | **IN**

where:

LI specifies that any legal encoding of the correct length may be used.

SD specifies that the *Short Definite* length type shall appear in the encoding.

LD specifies that the *Long Definite* length type shall appear in the encoding. The length shall be padded out to *number* octets if *number* appears.

IN specifies that the *Indefinite* length type shall appear in the encoding.

The length encoding shall consist of a whole number of octets.

A.14.3.3 Value Templates

It is possible to define complete ASN.1 values as constraints. In order to allow the use of parameters, wild cards and ranges for specifying constraints using a format similar to the ASN.1 value definition, a number of BNF productions are copied from ISO 8824, and some of these productions have been extended. The productions that are identical to productions used in ISO 8824 (1989) are indicated by "/* ISO 8824 */". The complete BNF is in {Annex A}.

The complete specification of a constraint using ASN.1 becomes:

SYNTAX DEFINITION:

- ASN1_ConstraintValue ::= [Encoding] ASN1_Value
- ASN1_Value ::=

```
        BooleanValue                           /* ISO 8824 */
      | IntegerValue                           /* ISO 8824 */
      | BitStringValue                         /* ISO 8824 */
      | OctetStringValue                       /* ISO 8824 */
      | NullValue                              /* ISO 8824 */
      | SequenceValue                          /* ISO 8824 */
      | SequenceOfValue                        /* ISO 8824 */
      | SetValue                               /* ISO 8824 */
      | SetOfValue                             /* ISO 8824 */
      | ChoiceValue                            /* ISO 8824 */
      | SelectionValue                         /* ISO 8824 */
      | TaggedValue                            /* ISO 8824 */
      | AnyValue                               /* ISO 8824 */
      | ObjectIdentifierValue                  /* ISO 8824 */
      | CharacterStringValue                   /* ISO 8824 */
      | EnumeratedValue                        /* ISO 8824 */
      | RealValue                              /* ISO 8824 */
      | UsefulValue                            /* ISO 8824 */
      | ReplaceValue
      | ConstraintIdentifier [RestrictedCrefList]
      | Wildcard
      | Parameter
```

/* Where they have been used within the ISO 8824 productions: BooleanValue .. UsefulValue , the following ISO 8824 productions have been redefined: */

- Value ::= ASN1_Value
- identifier ::= ASN1_Identifier

/* Additional non-ISO 8824 productions: */

- ReplaceValue ::= **IN** ConstraintIdentifier [RestrictedCrefList] "{" {Replacement} "}"
- Replacement ::= (**REPLACE** ReferenceList **BY** ASN1_Value) | (**OMIT** ReferenceList)
- ReferenceList ::= ASN1_Identifier | (ASN1_Identifier Dot ReferenceList)
- Parameter ::=TS_PARidentifier | TS_CONSTidentifier | PARidentifier
- ASN1_Identifier ::= Identifier

The constraints specified using ASN.1 can be used both for specifying PDUs or ASPs which are to be sent, or for specifying patterns against which an incoming PDU or ASP may be matched.

ASN.1 constraints may use the wildcard characters "?" and "*", in patterns of any of the string types or in place of explicit values of any element of an ASN.1 type, as defined in clause A.12.3.2.

A.14.3.4 Structured ASN.1 Constraints

The mechanism similar to that available in the tabular method, to structure constraint definitions by defining PDU field group and ASP parameter group constraints, can be used in ASN.1 constraints to define PDU field and ASP parameter constraints.

Single ASN.1 ASP parameters shall be specified in the following proforma:

ASN.1 ASP Parameter Constraint Declaration	
Parameter Name: *ASN1_ASP_PARidentifier*	**Constraint Name:** *CONSid&PARlist*
ASN.1 Value	
. . . *ASN1_ConstraintValue* . . .	

Proforma 32: ASN.1 ASP Parameter Constraint Declaration

SYNTAX DEFINITION:

- ASN1_ASPconstraints ::= **$ASN1_ASPconstraints** {ASN1_ASPconstraint} {ASN1_ASP_PARconstraint} **$End_ASN1_ASPconstraints**
- ASN1_ASP_PARconstraint ::= **$Begin_ASN1_ASP_PARconstraint** ASN1_ASP_PARid CONSid ASN1_Def **$End_ASN1_ASP_PARconstraint**

Single ASN.1 PDU fields shall be specified in the following proforma:

ASN.1 PDU Field Constraint Declaration	
Field Name: *ASN1_FIELDidentifier*	**Constraint Name:** *CONSid&PARlist*
ASN.1 Value	
. . . *ASN1_ConstraintValue* . . .	

Proforma 33: ASN.1 PDU Field Constraint Declaration

SYNTAX DEFINITION:

- ASN1_PDUconstraints ::= **$ASN1_PDUconstraints** {ASN1_PDUconstraint} {ASN1_FIELDconstraint} **$End_ASN1_PDUconstraints**
- ASN1_FIELDconstraint ::= **$Begin_ASN1_FIELDconstraint** ASN1_FIELDid CONSid ASN1_Def **$End_ASN1_FIELDconstraint**

Multiple ASN.1 ASP parameter constraints may also be specified in the following proforma:

ASN.1 ASP Parameter Constraints Declarations	
Parameter Name: *ASP_PARidentifier*	
Constraint name	**ASN.1 Value**
CONSid&PARlist	. *ASN1_ConstraintValue* .
⋮	⋮
CONSid&PARlist	. *ASN1_ConstraintValue* .

Proforma 34: Multiple ASN.1 ASP Parameter Constraints Declarations

Multiple ASN.1 PDU field constraints may also be specified in the following proforma:

ASN.1 PDU Field Constraints Declarations	
Field Name:*ASN1_ FIELDidentifier*	
Constraint name	**ASN.1 Value**
CONSid&PARlist	*ASN1_ConstraintValue*
⋮	⋮
CONSid&PARlist	*ASN1_ConstraintValue*

Proforma 35: Multiple ASN.1 PDU Field Constraints Declarations

A.14.3.5 Parameterized ASN.1 Constraints

It is possible to parameterize ASN.1 constraints. The parameter mechanism is identical to that of the tabular constraint definition and is described in clause A.13.2.3.

A.14.4 Modified Constraints

ASN.1 constraints can be defined by modifying an existing ASN.1 constraint. Portions of a defined constraint can be redefined to create a new constraint by using the REPLACE/OMIT mechanism in a REPLACE value as the ASN.1 value for the new constraint, where REPLACE value is defined as follows:

<u>SYNTAX DEFINITION:</u>

- ReplaceValue ::= **IN** ConstraintIdentifier [ActualParameterList] "{" {Replacement} "}"
- Replacement ::= (**REPLACE** ReferenceList **BY** ASN1_ValueTemplate) | (**OMIT** ReferenceList)
- ReferenceList ::= ASN1_Identifier | (ASN1_Identifier Dot ReferenceList)

NOTE - OMIT corresponds to: REPLACE ReferenceList BY { }

A.14.5 Using Formal Parameters

When using an ASN.1 constraint in the dynamic part of the test, the constraint reference is used along with a list of actual parameters. The latter correspond to the list of formal parameters which were used to specify the referenced constraint. Each actual parameter may either be a variable identifier, test suite parameter, test suite constant or an actual value.

EXAMPLE 45 - This example shows how to specify an ASN.1 constraint using formal parameters and then reference it in the dynamic part. Given the following ASN.1 type definition of the PDU:

ASN.1 PDU Type Declaration	
PDU Name: APDU	**PCO Type:**
ASN.1 Definition or Reference	
SEQUENCE {INTEGER BOOLEAN}	

A typical instance of this PDU could be:

ASN.1 PDU Constraint Declaration	
PDU Name: APDU	**Constraint Name:** ThePDU(X:INTEGER; Y:BOOLEAN)
ASN.1 Value	
[SD] {X,Y} -- A Sequence PDU of an integer and a boolean	

In this example x and y are identifiers which act as formal parameters whose sole purpose is to specify those parts of the PDU which can be substituted by actual parameters whenever the PDU is used in sending data or matching received data within a test case.

A.14.6 Mixing REPLACE/OMIT and Formal Parameters

Care should be taken when using both the REPLACE/OMIT and formal parameter styles that inconsistencies between formal parameters in the constraint header and the occurrence (or non-occurrence) of these parameters in the body of the constraint do not arise. In any case, parameterized fields shall not be replaced or omitted in a modified constraint.

A.15 The Dynamic Part

A.15.1 Introduction

The Dynamic Part contains the main body of the test suite: the test case, the test step and the default behaviour descriptions.

SYNTAX DEFINITION:

- DynamicPart ::= **$DynamicPart** TestCases [TestStepLibrary] [DefaultsLibrary] **$End_DynamicPart**

A.15.2 Specification of Test Case Dynamic Behaviour

A.15.2.1 The Test Case Proforma

The following information shall be supplied for the dynamic behaviour of each test case:

a) a Test Case Reference (clause A.8.2),

giving a full name to the test case behaviour description and defines its location in the test suite structure; a test case reference shall conform to the requirements of clause A.8.2;

b) a Test Case Identifier,

used to provide a shorter name for a test case; it may be used interchangeably with a test case reference;

c) a brief description of the Test Purpose,

which shall be an informal statement of purpose of the test case, possibly summarizing the full test purpose given in the relevant test suite structure and test purposes standard or equivalent section of the test suite standard;

d) identification of the Default behaviour to be used (clause A.15.14),

   which shall be the identifier (including an actual parameter list if necessary) of a default behaviour description, if any, which applies to the test case behaviour specification;

e) a Behaviour Description (clause A.15.5),

   which shall describe the behaviour of the lower tester and/or upper tester in terms of test events (and their parameters) using the tree notation described in clause A.15.6;

f) Labels (clause A.15.14),

   placed in the labels column to identify TTCN statements either for statement line numbering and/or to allow jumps using the GOTO statement;

g) Constraints References (clause A.15.16),

   placed in the constraints reference column to associate TTCN statements in a behaviour tree with a reference to specific ASP and/or PDU values defined in the constraints part (clause A.12);

h) Verdict or result information (clause A.15.17),

   placed in the verdicts column to be associated with TTCN statements in a behaviour tree;

i) Comments,

   placed in the comments column to ease the understanding of the TTCN statements by providing short remarks or references to additional text in the optional extended comments field;

j) Extended comments,

   used to give longer comments and general comments.

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

Test Case Dynamic Behaviour				
Reference: *TestCaseReference* **Identifier:** *TestCaseIdentifier* **Purpose:** *FreeText* **Default:** *DefaultsReference*				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**
StatementLine	.	.	.	.
.	.	.	.	.
.	.	.	.	.
TreeHeader	*Label*	*Constraints Reference*	*Verdict*	*FreeText*
.	.	.	.	.
StatementLine	.	.	.	.
.	.	.	.	.
Extended Comments: *FreeText*				

Proforma 36: Test Case Dynamic Behaviour

NOTE - a synchronization requirements field may be added to the proforma at a future date. Synchronization is expected to be addressed in an addendum to this part of the multipart standard*.

The four columns to the right of the table may be headed: **L**, **Cref**, **V** and **C**, but shall not be omitted. This enables the behaviour tree column to be as wide as possible in cases of physical paper size limitations.

SYNTAX DEFINITION:

- TestCase ::= **$Begin_TestCase** TestCaseRef TestCaseId TestPurpose [DefaultsRef] BehaviourDescription [ExtComments] **$End_TestCase**
- TestCaseRef ::= **$TestCaseRef** TestCaseReference
- TestCaseReference ::= (TestGroupReference TestCaseName) | (SuiteIdentifier "/" TestCaseName)
- TestGroupReference ::= SuiteIdentifier "/" {TestGroupIdentifier "/"}+
- TestCaseName ::= Identifier
- TestCaseId ::= **$TestCaseId** TestCaseIdentifier
- TestCaseIdentifier ::= Identifier
- TestPurpose ::= **$TestPurpose** BoundedFreeText
- DefaultsRef ::= **$DefaultsRef** DefaultsReference
- DefaultsReference ::= DefaultIdentifier [ActualPARlist]
- ExtComments ::= **$ExtComments** BoundedFreeText

A.15.2.2 Structure of the Test Case Behaviour

Each test case contains a precise description of sequences of (anticipated) events and related verdicts. This description is structured as a tree, with TTCN statements as nodes in that tree and verdict assignments at its leaves.

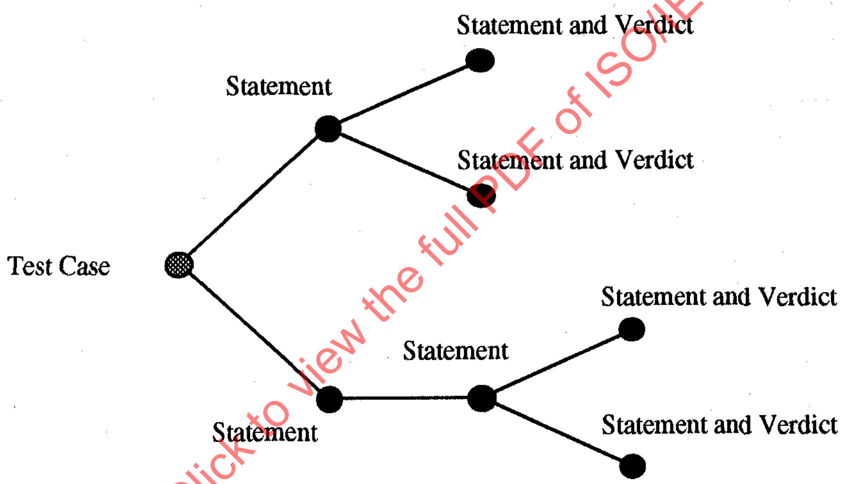

Figure A.6: Test Case Behaviour Structure

In many cases it is more efficient to use test steps as a means for substructuring this tree:

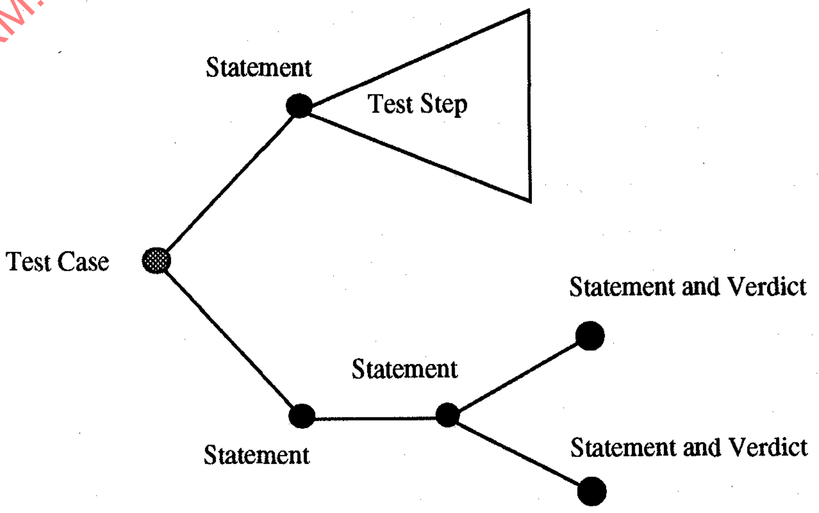

Figure A.7: Structured Test Case Behaviour

In TTCN this explicit modularization is expressed using test steps and the ATTACH statement.

A.15.3 Specification of Test Step Dynamic Behaviour

Test step dynamic behaviours are identical to test case dynamic behaviours, except for the following differences:

a) the table has the heading "Test Step Dynamic Behaviour";

b) the test step reference specifies the location of the test step in the test step library; a test step reference shall conform to the requirements of clause A.8.3;

c) the test step identifier may be used interchangeably with a test step reference;

d) an optional list of formal parameters, and their associated types, may be specified immediately following the test step identifier. These parameters may be used to pass PCOs, constraints or other data objects into the root tree of the test step;

e) test steps do not have a test purpose, instead they have an objective. This is an informal statement of the objective of the test step.

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

Test Step Dynamic Behaviour				
Reference: *TestStepReference* **Identifier:** *TestStepIdentifier [FormalPARlist]* **Objective:** *FreeText* **Default:** *DefaultsReference*				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**
StatementLine	.	.	.	.
.	.	.	.	.
.	.	.	.	.
TreeHeader	*Label*	*Constraints Reference*	*Verdict*	*FreeText*
.	.	.	.	.
StatementLine	.	.	.	.
.	.	.	.	.
Extended Comments: *FreeText*				

Proforma 37: Test Step Dynamic Behaviour

SYNTAX DEFINITION:

- TestStep ::= **$Begin_TestStep** TestStepRef TestStepId Objective [DefaultsRef] BehaviourDescription [ExtComments] **$End_TestStep**
- TestStepRef ::= **$TestStepRef** TestStepReference
- TestStepReference ::= (TestStepGroupReference TestStepName) | (SuiteIdentifier "/" TestStepName)
- TestStepGroupReference ::= SuiteIdentifier "/" {TestStepGroupIdentifier "/"}+
- TestStepName ::= Identifier
- TestStepId ::= **$TestStepId** TestStepIdentifier [FormalPARlist]
- TestStepIdentifier ::= Identifier
- Objective ::= **$Objective** BoundedFreeText

A.15.4 Specification of Dynamic Default Behaviours

A TTCN test specification shall specify alternative behaviour for *every* possible event. It often happens that in a behaviour tree every sequence of alternatives ends in the same behaviour. This behaviour may be 'factored out' as default behaviour to this tree. Such default behaviour descriptions are located in the global defaults library.

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

Default Dynamic Behaviour				
Reference: *DefaultReference* **Identifier:** *DefaultIdentifier [FormalPARlist]* **Objective:** *FreeText*				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**
. . . *StatementLine* . . .	. . . *Label* . . .	. . . *Constraints Reference* . .	. . . *Verdict* . . .	. . . *FreeText* . . .
Extended Comments: *FreeText*				

Proforma 38: Default Dynamic Behaviour

The default dynamic behaviour proforma shall be identical to the test step dynamic behaviour proforma, except for the following differences:

a) the table is headed ""Default Dynamic Behaviour";

b) the default reference specifies the location of the default in the defaults library. A default reference shall conform to the requirements of clause A.8.4. These complete references shall be unique within the test suite;

c) the default identifier may be used interchangeably with a default reference;

d) a default dynamic behaviour proforma does not have an entry in its heading for a defaults reference since defaults are not allowed to have defaults;

e) it shall contain only one behaviour tree.

It should be noted that both PCOs and other actual parameters may be passed to default behaviour descriptions in the same way that they may be passed to test steps. The same rules on scope and substitution of these parameters apply as described for tree attachment (clause A.15.13.2).

NOTE - For the meaning of defaults see clause A.15.18.

SYNTAX DEFINITION:

- Default ::= **$Begin_Default** DefaultRef DefaultId Objective BehaviourDescription [ExtComments] **$End_Default**
- DefaultRef ::= **$DefaultRef** DefaultReference
- DefaultReference ::= (DefaultGroupReference DefaultName) | (SuiteIdentifier "/" DefaultName)
- DefaultGroupReference ::= SuiteIdentifier "/" {DefaultGroupIdentifier "/"}+
- DefaultGroupIdentifier ::= Identifier
- DefaultName ::= Identifier
- DefaultId ::= **$DefaultId** DefaultIdentifier [FormalPARlist]
- DefaultIdentifier ::= Identifier

A.15.5 The Behaviour Description

The behaviour description column of a dynamic behaviour table contains the specification of the combinations of TTCN statements that are deemed possible by the test suite specifier. The set of these combinations is called the behaviour tree. Each TTCN statement is a node in the behaviour tree.

A.15.6 The Tree Notation

In the TTCN notation, each TTCN statement shall be shown on a separate statement line. The statements can be related to one another in two ways:

- as sequences of TTCN statements;
- as alternative TTCN statements.

Sequences of TTCN statements are represented one statement line after the other, each new TTCN statement being indented once from left to right, as time is assumed to progress.

EXAMPLE 46 - TTCN statements in sequence:

```
EVENT_A
    CONSTRUCT_B
        EVENT_C
```

Test TTCN statements at the same level of indentation and belonging to the same predecessor node represent the possible alternative TTCN statements which may occur at that time. Alternative TTCN statements shall be given in the order in which the appropriate tester shall repeatedly attempt them until one occurs.

EXAMPLE 47 - alternative TTCN statements:

```
CONSTRUCT_B1
STATEMENT_B2
EVENT_B3
```

EXAMPLE 48 - combining sequences and alternatives to build a tree:

```
EVENT_A
    CONSTRUCT_B1
    STATEMENT_B2
    EVENT_B3
        EVENT_C
```

The term "set of alternatives" is used generically to describe:

a) true "sets" of more than 1 element, but also "sets" containing only a single element;

b) behavior that can be considered as actual alternatives - that is, mutually exclusive behavior - as well as behavior lines that may, because of their static or dynamic semantics, include unreachable behavior lines. An example of unreachable behavior due to static semantics is when behavior lines are coded (sequentially) after, and at the same level of indentation as, a TTCN statement such as assignment, which always succeeds. An example of unreachable behavior due to dynamic semantics occurs whenever an event matches one of a set of alternatives on a pass through the set of alternatives, because all the subsequent members of the set of alternatives are then unreachable. In all cases, subsequent behaviour to non-matched alternatives is unreachable because of dynamic semantics.

A.15.7 Tree Names and Parameter Lists

A.15.7.1 Introduction

Each behaviour description shall contain at least one behaviour tree. So that trees may be unambiguously referred to (such as in an ATTACH statement) each tree has a tree name.

The first tree appearing within a behaviour description is called the root tree. The name of a root tree is the identifier appearing in the header of its dynamic behaviour table. That is, the tree name of the root tree of a test step is the test step identifier for that test step, and likewise for root trees in test case dynamic behaviours and default dynamic behaviours.

Trees other than the root tree which appear within dynamic behaviour tables are termed local trees. Local trees are prefixed by a tree header which contains the tree name.

SYNTAX DEFINITION:

- TreeHeader ::= **$TreeHeader** Header
- Header ::= TreeIdentifier [FormalPARlist]
- TreeIdentifier ::= Identifier
- FormalPARlist ::= "(" FormalPARsubList {SemiColon FormalPARsubList} ")"

- FormalPARsubList ::= PARidentifier {Comma PARidentifier} Colon ParameterType
- PARidentifier ::= Identifier
- SemiColon ::= ";"
- Colon ::= ":"
- ParameterType ::= Type | ReferenceType | PCOtypeIdentifier

A.15.7.2 Trees with Parameters

All trees, except test case root trees, may make use of parameters. These parameters may be used to provide PCOs, constraints, variables, or other such items for use within the tree. Test case root trees shall never make use of parameters.

If a tree makes use of parameters, then a list of formal parameters and their types shall appear within parentheses directly following the tree name. For example, the formal parameter list for a test step root tree shall appear within parentheses immediately following the test step identifier in the header of the test step dynamic behaviour table. Similarly, the formal parameter list for a local tree shall appear immediately after the tree name in the tree header.

In constructing the formal parameter list, each formal parameter shall be followed by a colon and the name of the formal parameter's type. If more than one formal parameter of the same type is present, these may be combined into a sub-list. When such a sub-list is used, the formal parameters within the sub-list shall be separated from each other by a comma. The final formal parameter in the sub-list shall be followed by a colon and the formal parameter's type.

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

Formal parameters may be of PCO type, constraint type, or one of the other TTCN predefined or user types.

EXAMPLE 49 - A local tree header using formal parameters: EXAMPLE_TREE(L:TSAP; X:INTEGER)

EXAMPLE 50 - A test step root tree using formal parameter sub-lists:

Test Step Dynamic Behaviour				
Reference: TTCN_EXAMPLES/CSP_101 **Identifier:** PREAMBLE(A, B:INTEGER; X:BITSTRING) **Objective:** To illustrate test step identifier with a formal parameter list. **Default:**				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**
!CONNECTrequest . .		CR1		

A.15.8 TTCN Statements

The tree notation allows the specification of test events initiated by the tester, test events received by the tester, TTCN pseudo-events and TTCN constructs. These are collectively known as TTCN statements.

SYNTAX DEFINITION:

- Statement ::= Event | PseudoEvent | Construct
- Event ::= Send | ImplicitSend | Receive | Otherwise | Timeout
- Construct ::= GoTo | Attach | Repeat
- PseudoEvent ::= [TTCNExpression] [TimerOperations]

The TTCN Test events are the ASPs or PDUs to be initiated or received by the lower- or upper tester, the OTHERWISE event and the TIMEOUT event. TTCN also supports the GOTO, ATTACH and REPEAT constructs.

Test events can be accompanied by Boolean expressions, assignments and timer operations. Boolean expressions, assignments and timer operations can also stand alone, in which case they are called pseudo-events.

A.15.9 TTCN Test Events

A.15.9.1 Sending and Receiving Events

TTCN supports the initiation (sending) of ASPs and PDUs to named PCOs and acceptance (receipt) of ASPs and PDUs at named PCOs. The PCO model is defined in clause A.11.8.

The names of events to be initiated by the appropriate tester shall be prefixed by an exclamation mark (!). Those events which it is possible for the appropriate tester to accept shall be prefixed by a question mark (?).

An unqualified SEND event is always successful.

Such names (composed of "!" or "?" followed by an event name) shall be prefixed by one of the PCO names appearing in the formal parameter list of the tree in which the event appears. The PCO name is used to indicate the PCO at which the test event may occur. If the test suite only uses one PCO the PCO prefix may be omitted.

SYNTAX DEFINITION:

- Send ::= [PCOidentifier | PARidentifier] "!" (ASPidentifier | PDUidentifier) [EncodedAs] [TTCNExpression] [TimerOperations]
- Receive ::= [PCOidentifier | PARidentifier] "?" (ASPidentifier | PDUidentifier) [DecodesAs] [TTCNExpression] [TimerOperations]

In the simplest form an ASP identifier or PDU identifier follows the "!" or "?", as in the following example:

EXAMPLE 51 - !SUBreq or ?CONind

NOTE - see clause A.15.9.7 for the definition of the lifetime of received event. This definition covers the minimum scope of received events and may be extended in future versions of TTCN.

A.15.9.2 Alignment of Test Events

A set of alternatives (i.e. events, pseudo-events and TTCN constructs) shall be written so that the first symbol of each of the alternatives is aligned in the same column (level of indentation) within the behaviour description .

EXAMPLE 52 - alignment of alternatives

```
L?A
L?B
STATEMENT_C
```

These three lines represent a set of alternatives. (The syntax of TTCN statements is discussed in subsequent clauses of this standard*).

A.15.9.3 Execution of the Behaviour Tree

A.15.9.3.1 Introduction

The test suite specifier shall organize the behaviour tree representing a test case or a test step according to the following rules regarding test execution:

- starting from the root of the tree, the lower or upper tester remains on the first level of indentation until an event occurs. If an event is to be initiated the tester initiates it; if an event is to be received , it is said to occur only if a received real event matches the description of the event in the behaviour tree;

- once an event has occurred, the tester remains on the next level of indentation. No return to a previous level of indentation can be made, except by using the GOTO statement;

- test events at the same level of indentation and following the same predecessor event represent the possible alternative events which may occur at that time. Alternative events shall be given in the order that the test suite specifier requires the lower or upper tester to attempt either to initiate or receive them, if necessary, repeatedly, until one occurs.

NOTE - The OTHERWISE event, pseudo-events and TTCN constructs bring further complexity to the above rules. Refer to the relevant clauses for execution rules.

Suppose that the following sequence of events can occur during a test whose purpose is to establish a connection, exchange some data, and close the connection. The events occur at the lower tester PCO L:

a) CONNECTrequest, CONNECTconfirm, DATArequest, DATAindication, DISCONNECTrequest;

Progress can be thwarted at any time by the IUT or the service provider. This generates two more sequences:

b) CONNECTrequest, CONNECTconfirm, DATArequest, DISCONNECTindication;

c) CONNECTrequest, DISCONNECTindication.

The three sequences of events can be expressed as a TTCN behaviour tree. There are five levels of alternatives, and only three tips (a to c), because the SEND events L! are always successful. Execution is to progress from left to right (sequence), and from top to bottom (alternatives). The following diagram illustrates this progression, and the principle of the TTCN behaviour tree. :

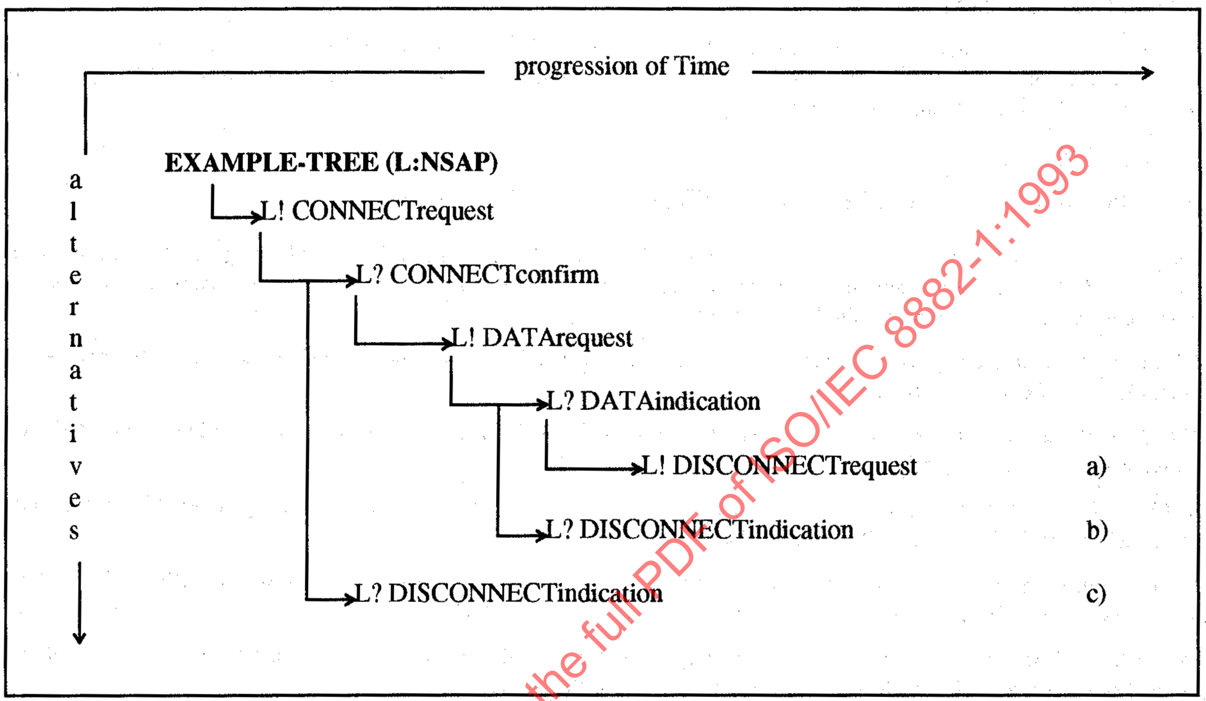

Figure A.8: Principle of a TTCN Behaviour Tree

There are no lines, arrows or tip names in the TTCN notation. The behaviour tree of this example would be represented in TTCN as in the following figure:

Test Case Dynamic Behaviour				
Reference: TTCN_EXAMPLES/TREE_EXAMPLE_1 **Identifier:** TREE_EX_1(L:NSAP) **Purpose:** To illustrate the use of trees. **Default:**				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**
L!CONNECTrequest		CR1		Request ...
L?CONNECTconfirm		CC1		... Confirm
L !DATArequest		DTR1		Send Data
L?DATAindication		DTI1		Receive Data
L !DISCONNECTrequest		DSCR1	pass	Accept
L ?DISCONNECTindication		DSCI1	inconc	Premature
L ?DISCONNECTindication		DSCR1	inconc	Premature

Figure A.9: A TTCN Behaviour Tree

NOTE - See clause A.15.19 to see how this example can be simplified by specifying the L? DISCONNECTindications as default behaviour.

A.15.9.3.2 The Concept of Snapshot Semantics

The alternative events at the current level of indentation are processed in their order of appearance. TTCN operational semantics (see annex B) assume that the status of any of the events cannot change during the process of trying to match one of a set of alternatives. This implies that *snapshot semantics* are used for received events and TIMEOUTs - i.e. each time around a set of alternatives a snapshot is taken of which events have been received and which TIMEOUTs have fired. Only those identified in the snapshot can match on the next cycle through the alternatives.

A.15.9.4 The IMPLICIT SEND Event

In the Remote Test Methods, although there is no explicit PCO above the IUT, it is necessary to have a means of specifying, at a given point in the description of the behaviour of the Lower Tester, that the IUT should be made to initiate a particular PDU or ASP. For this purpose, the implicit send event is defined, with the following syntax:

SYNTAX DEFINITION:

- ImplicitSend ::= "<" **IUT** "!" (ASPidentifier | PDUidentifier) ">" [EncodedAs]

The **IUT** in the syntax takes the place of the PCOidentifier used with a normal SEND or RECEIVE, indicating that the specified ASP or PDU is to be sent by the IUT. The angle brackets signify that this is an implicit event, i.e. there is no specification of what is done to the IUT to trigger this reaction, only a specification of the required reaction itself.

An IMPLICIT SEND event is always considered to be successful, in the sense that any alternatives coded after, and at the same level of indentation as the IMPLICIT SEND are unreachable.

An IMPLICIT SEND shall only be used where the relevant OSI* standard(s)* permit the IUT to send the specified ASP or PDU at that point in its communication with the Lower Tester.

For every IMPLICIT SEND in a test suite, the test suite specifier shall create and reference a question in the partial PIXIT proforma that allows indication of whether the IMPLICIT SEND can be invoked on demand.

An IMPLICIT SEND event shall not be used unless the test method being used is one of the Remote methods. An IMPLICIT SEND event shall not be used unless the same effect could have been achieved using the DS test method.

When an IMPLICIT SEND event is specified, the associated internal events within the IUT necessary to meet the requirements of the standard for the protocol being tested are also performed, e.g. set timer, initialize state variables.

NOTE - For example, when testing a connection-oriented Transport Protocol implementation, it would be permissible to use IMPLICIT SEND to get the IUT to initiate a CR TPDU because in the DS test method that effect could be achieved by getting the Upper Tester to send a T-ConReq ASP. On the other hand, it would not be permissible to use IMPLICIT SEND to get the IUT to initiate an N-RstReq ASP because that effect could not be controlled through the Transport Service boundary. The reason for this restriction is to prevent test cases from requiring greater external control over an IUT than is provided for in the relevant protocol standard*.

The semantics of IMPLICIT SEND is that the SUT shall be controlled as necessary in order to cause the initiation of the specified ASP or PDU. The way in which the SUT is to be controlled should be specified in the PIXIT (or documentation referenced by the PIXIT).

Neither a final verdict nor a preliminary result shall be coded on an IMPLICIT SEND event.

At an appropriate point following an IMPLICIT SEND, there should be a RECEIVE event to match the ASP or PDU that should, as a result, have been sent by the IUT.

NOTE - Such a RECEIVE event is often likely to be specified at the next level of indentation immediately after the IMPLICIT SEND, but it may be specified further down the tree if it is expected that there are other PDUs already in transit towards the Lower Tester.

EXAMPLE 53 - EXAMPLE use of IMPLICIT SEND

Test Step Dynamic Behaviour				
Reference: TTCN_EXAMPLES/IMPLICIT_SEND1 **Identifier:** IMP1(LT:N_SAP) **Objective:** A partial tree to illustrate the use of IMPLICIT SEND. **Default:**				
Behaviour Description	**Label**	**Constraints Reference**	**Verdict**	**Comments**

.
.
<IUT ! CR >
  LT ? CR
    LT ! CC
      ...
    ...
  LT ? OTHERWISE
    ...
		
		CR1
		CR1
		CC1		

A.15.9.5 The OTHERWISE Event

The predefined event OTHERWISE is the TTCN mechanism for dealing with unforeseen test events in a controlled way. OTHERWISE has the syntax:

SYNTAX DEFINITION:

- Otherwise ::= [PCOidentifier | PARidentifier] "?" **OTHERWISE** [TTCNExpression] [TimerOperations]

OTHERWISE is used to denote that the appropriate tester shall accept *any* incoming event which has not previously matched one of the alternatives to the OTHERWISE.

If a tree uses multiple PCOs then the OTHERWISE shall be preceded by a PCO identifier. Incoming events, including OTHERWISE, are considered only in terms of the given PCO.

EXAMPLE 54 - use of OTHERWISE with PCO identifiers

```
PARTIAL_TREE (PCO1:XSAP; PCO2:YSAP)
PCO1? A
PCO2? B                    pass
PCO1? C                    inconc
PCO2? OTHERWISE            fail
```

Assume no event is received at PCO1, then receipt of event B at PCO2 results in a pass verdict. Receipt of any other event at PCO2 results in a fail verdict.

Due to the significance of ordering of alternatives, incoming events following an (unconditional) OTHERWISE on the same PCO will never be matched.

EXAMPLE 55 - incoming events following an OTHERWISE:

```
PARTIAL_TREE (PCO1:XSAP)
PCO1? A
PCO1? B
PCO1? OTHERWISE
PCO1? C
```

The OTHERWISE will match any incoming event other than A or B. The last alternative, ?C, can never be matched.

A.15.9.6 The TIMEOUT Event

A.15.9.6.1 Introduction

The TIMEOUT event allows expiration of a timer, or of all timers, to be checked in a test case. When a timer expires (conceptually immediately before a snapshot processing of a set of alternative events), a TIMEOUT event is placed into a timeout list. The timer becomes immediately inactive. Only one entry for any particular timer may appear in the list at any one time. Since TIMEOUT is not associated with a PCO, a single timeout list is used.

When a TIMEOUT event is processed, if a timer name is indicated, the timeout list is searched, and if there is a timeout event matching the timer name, that event is removed from the list, and the TIMEOUT event succeeds.

If no timer name is indicated, any timeout event in the timeout list matches. The TIMEOUT event succeeds if the list is not empty; and the list itself is immediately emptied.

TIMEOUT has the following syntax:

SYNTAX DEFINITION:

- Timeout ::= **?TIMEOUT** [TimerIdentifier] [TTCNExpression] [TimerOperations]

EXAMPLE 56 - ?TIMEOUT T

If the timer identifier is omitted, then the TIMEOUT applies to any timer which has expired.

A.15.9.6.2 Using TIMEOUT with OTHERWISE

Because TIMEOUT events do not occur at any particular PCO they are not covered by the OTHERWISE event:

EXAMPLE 57 - relationship of TIMEOUT and OTHERWISE:

```
PARTIAL_TREE (PCO1:XSAP)
PCO1? A                  pass
PCO1? OTHERWISE          fail
?TIMEOUT T               inconc
```

An inconclusive verdict is assigned if no incoming event is received at PCO1 (neither A nor anything else) and the timer T expires.

A.15.9.7 Lifetime of Events

Identifiers of ASP parameters and PDU fields associated with SEND and RECEIVE shall only be used to reference ASP parameter and PDU field values on the statement line itself.

Therefore, in the case of SEND events, relevant ASP parameters and PDU fields can be set, if required, by using them in appropriate assignments on the SEND line. Similarly, in the case of RECEIVE events, if relevant ASP parameter and PDU field values need to be subsequently referenced, either the whole ASP or PDU or a relevant part of it shall be assigned to variables on the RECEIVE line itself. These variables may then be referenced in subsequent lines.

A.15.10 TTCN Expressions

A.15.10.1 Introduction

There are two kinds of expressions: ASSIGNMENTS and BOOLEAN expressions.

SYNTAX DEFINITION:

- TTCNExpression ::= ({BooleanExpression}+ {Assignment}+) | {BooleanExpression}+ | {Assignment}+

Both assignments and Boolean expressions may contain explicit values and the following forms of reference to data objects:

a) Test suite parameters;

b) Test suite constants;

c) Test suite and test case variables;

d) Formal parameters of a test step, default or local tree;

e) ASPs and PDUs (on event lines).

SYNTAX DEFINITION:

- DataObjectIdentifier ::= TCVid | PARidentifier | ASP_PDUidentifier
- TCVid ::= TS_PARidentifier | TS_CONSTidentifier | VARidentifier
- ASP_PDUidentifier ::= ASPidentifier | PDUidentifier

A.15.10.2 References for ASN.1 Defined Data Objects

To refer to components of structured data objects the following access mechanisms are provided:

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

b) references to unnamed components are constructed by giving the position of the component within the type definition in parentheses; the component identifier shall be used if specified; numbering of such components of such a type starts with "1".

SYNTAX DEFINITION:

- DataObjectReference ::= (DataObjectIdentifier {ComponentReference}) | (ComponentIdentifier {ComponentReference})
- ComponentReference ::= RecordRef | ArrayRef | BitRef
- RecordRef ::= Dot (ComponentIdentifier | ComponentPosition)
- ComponentIdentifier ::= ASP_PARidentifier | FIELDidentifier
- ComponentPosition ::= "("Number")"

Omitting the data object identifier (i.e. starting the reference with a component identifier) is only allowed for references to the ASPs/PDUs associated with event lines. If this leads to ambiguity (e.g. there is an existing variable with the same name as a referenced ASP parameter or PDU field) the full reference shall be used.

```
EXAMPLE 58 -   example_type ::= SEQUENCE {
                        field_1   INTEGER,
                        field_2   BOOLEAN
                                  OCTETSTRING  }
```

If var1 is of ASN.1 type example_type, then we could write:

var1.field_1 -- refers to the first INTEGER field --

var1.(3) -- refers to the third (unnamed) field --

```
EXAMPLE 59 -   XY_PDU_type ::= SEQUENCE {
                        :
                        user_data    OCTETSTRING,
                        :  }
```

On the statement line: L? XY_PDU (buffer := user_data)

user_data may be used instead of XY_PDU.user_data if there is no other data object defined with the name user_data.

An index enclosed in square brackets is used to refer to a component of an ASN.1 SEQUENCE OF or SET OF type.

SYNTAX DEFINITION:

- ArrayRef ::= "[" ComponentNumber "]"
- ComponentNumber ::= SimpleExpression

The first component has the number "0".

The simple expression shall evaluate to a non-negative INTEGER.

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

SYNTAX DEFINITION:

- BitRef ::= Dot BitIdentifier | "[" BitNumber "]"
- BitIdentifier ::= Identifier
- BitNumber ::= SimpleExpression

The leftmost bit has the number "0".

The simple expression shall evaluate to a non-negative INTEGER.

EXAMPLE 60 - b_type ::= BITSTRING { ack(0), poll(3) }

where bit zero is called "ack" and bit three is called "poll"

If b_str is of ASN.1 type b_type, then we could write:

b_str.ack := TRUE

b_str[2] := FALSE

Note that b_str.poll := TRUE and b_str[3] := TRUE both assign the value TRUE to the "poll" bit.

A.15.10.3 References for Data Objects Defined by the Tabular Method

The same syntax as defined in A.15.10.2 shall be used to construct references to structured data objects defined in the tabular form (i.e. ASPs and PDUs). To specify a reference to an ASP parameter or PDU field the data object identifier shall be followed by a dot and a parameter or field identifier. When accessing parameters or fields of ASPs or PDUs on the same statement line they are received/sent, the data object reference (i.e. the ASP identifier or PDU identifier) may be omitted resulting in an abbreviated form.

A.15.10.4 Assignments

A.15.10.4.1 Introduction

The effect of an assignment is to bind the test case or test suite variable (or ASP parameter or PDU field) to the value of the expression if, and only if, the event occurs. The expression shall contain no unbound variables.

The rules for use of assignments within events are as follows:

a) all assignments occur in the order in which they appear, that is left to right processing;

b) on a SEND event all assignments are performed *after* the Boolean expression is evaluated and *before* the ASP or PDU is transmitted;

c) on SEND events assignments are allowed for the fields of the ASP or PDU being transmitted;

d) on a RECEIVE event assignments are performed *after* the event occurs and cannot be made to fields of the ASP or PDU just received.

EXAMPLE 61 - use of assignments with event lines:

```
(X:=1)
   (Y:=2)
      L!A (Y:=0, X:=Y) (A.field1:=Y)
         L?B (Y:=B.field2) (X:=X+1)
```

When PDU A is successfully transmitted the contents of the test case variables X and Y will be zero, and field1 of PDU A will also contain zero. Upon receipt of PDU B the test case variable Y would be assigned the contents of field2 from PDU B and the test case variable X would be incremented.

SYNTAX DEFINITION:

- Assignment ::= "(" SimpleAssignment {Comma SimpleAssignment} ")"
- SimpleAssignment ::= DataObjectReference "::=" Expression
- Expression ::= SimpleExpression [Relop SimpleExpression]
- SimpleExpression ::= Term {AddOp Term}
- Term ::= Factor {MultiplyOp Factor}
- Factor ::= [UnaryOp] Primary
- Primary ::= DataObjectReference | **R** | UserOperation | "(" Expression ")" | Number | Cstring | Hstring | Ostring | Bstring | BooleanValue | ASN1_EnumeratedIdentifier | ASN1_NamedIntegerIdentifier
- ASN1_EnumeratedIdentifier ::= Identifier
- ASN1_NamedIntegerIdentifier ::= Integer
- AddOp ::= "+" | "-" | **OR**
- MultiplyOp ::= "*" | "/" | **MOD** | **AND**
- UnaryOp ::= "+" | "-" | **NOT**
- UserOperation ::= OPidentifier [ActualPARlist]

The types on the left-hand side and the right-hand side of an assignment shall be assignment compatible (as defined in clause A.11.2.4).

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

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).

A.15.10.4.2 Assignment Rules for String Types

If length restricted string types are used within an assignment the following rules apply:

a) if the destination string type is defined to be shorter than the source string, the source string is truncated on the right to the maximum length of the destination string type;

b) if the source string is shorter than that allowed by the destination string type, then the source string is left aligned and padded with fill characters up to the maximum size of the destination string type.

Fill characters are:

" " (blank) for all character strings;

"0" (zero) for BITSTRINGs, HEXSTRINGs and OCTETSTRINGs.

A.15.10.5 Boolean Expressions

An event may be qualified by placing a Boolean expression after the event. This qualification shall be taken to mean that a match may only occur if both the event matches and the Boolean expression evaluates to TRUE.

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

SYNTAX DEFINITION:

- BooleanExpression ::= "[" Expression "]" /* *Expression shall evaluate to a Boolean Value* */
- Relop ::= "=" | "<" | ">" | "<>" | ">=" | "<="

A.15.10.6 Assignments and Boolean Expressions with Events

It is allowed to associate an event with either an assignment, or a Boolean expression or both. If an event is followed by an assignment, the assignment is executed only if the event occurs. If an event is followed by a Boolean expression, the event may occur only if the Boolean expression holds. If an event is followed by both, the event may occur only if the Boolean expression holds, and the assignment is only executed if the event occurs.

If a RECEIVE event is qualified by a Boolean expression and the event that has occurred potentially matches the specified event, then the Boolean expression shall be evaluated in the context of the event that has occurred. If the Boolean expression contains a reference to ASP parameters and/or PDU fields then the values of those parameters and/or fields are taken from the event that has occurred and shall also be consistent with the specified constraint; in such cases the value in the relevant constraints declaration of each ASP parameter and/or PDU field shall be a 'don't care' value, a list of values, or a range of values.

Boolean expressions may be broken down into a sequence of Boolean expressions. Similarly, assignments may be broken down into a sequence of assignments. The separate assignments are performed sequentially, left-to-right.

EXAMPLE 62 - ?CR [W=1][X<2] (Y:=3) (Z:=4) is equivalent to ?CR [W=1 AND X<2] (Y:=3, Z:=4)

EXAMPLE 63 - The OTHERWISE event may be used together with Boolean expressions and/or assignments. If a Boolean expression is used, this Boolean becomes an additional condition for accepting any incoming event. If an assignment statement is used, the assignment will take place only if all conditions for matching the OTHERWISE are satisfied. For example,

```
PARTIAL_TREE (PCO1:XSAP; PCO2:YSAP)
PCO1? A                                                pass
PCO2? B [X=2]                                          inconc
PCO1? C                                                pass
PCO2? OTHERWISE [X<>2] (Reason:="X not equal 2")       fail
PCO2? OTHERWISE (Reason:="X equals 2 but event not B") fail
```

Assume that no event is received at PCO1. Receipt of event B at PCO2 when X=2 gives an inconclusive verdict. Receipt of any other event at PCO2 when X<>2 results in a fail verdict and assigns a value of "X not equal 2" to the character string variable: Reason. If an event is received at PCO2 that satisfies neither of these scenarios then the final OTHERWISE will match.

EXAMPLE 64 - Use of a qualified SEND event

```
PARTIAL_TREE
?A              pass
!B[X=3]         inconc
```

? OTHERWISE fail

A verdict of fail will be assigned if neither A is received nor B is sent due to X<>3. If X=3 an inconclusive verdict is assigned and the OTHERWISE never occurs.

A.15.10.7 Assignments and Boolean Expressions Without Events

It is permitted to use assignments and Boolean expressions by themselves on a statement line in a behaviour tree, without any associated event. These stand-alone expressions are called pseudo-events. Only the following combinations are allowed:

a) assignments alone; the assignments are executed;

b) Boolean expressions alone; the Boolean expressions are evaluated, execution continues with subsequent behaviour (if any) only if the expressions hold;

c) Boolean expressions followed by assignments; the assignments are executed only if the Boolean expression holds.

A.15.11 The ENCODE and DECODE Expressions

A.15.11.1 Introduction

The ENCODE and DECODE expressions allow the specification of the encoding of PDUs embedded in ASPs or other PDUs.

It is recommended that the static chaining of constraints mechanism (clause A.15.11.1) is used for specifying ASPs with embedded PDUs. If desired this chaining mechanism can be used in combination with parameterized constraints. See {Annex C} for examples of these mechanisms.

NOTE - ENCODE and DECODE are discouraged, but they are still part of TTCN for reasons of upward compatibility. The static chaining mechanism was not available in DP versions of TTCN and therefore ENCODE and DECODE were used instead.

A.15.11.2 The ENCODE Expression

The ENCODE operator is written as ^, and should be read as "is encoded as". An ENCODE expression is only associated with a SEND events:

The ENCODE expression has the syntax:

SYNTAX DEFINITION:

- Send ::= [PCOidentifier | PARidentifier] "!" (ASPidentifier | PDUidentifier) [EncodedAs] [TTCNExpression] [TimerOperations]
- EncodedAs ::= "<" DataObjectReference "^" PDUidentifier [EncodedAs] ">"

The left-hand side of the ENCODE expression shall be the name given to the user data field in the appropriate ASP or PDU type declaration.

A.15.11.3 The DECODE Expression

The DECODE operator is written as ~, and should be read as "decodes as". A DECODE expression is only associated with a RECEIVE event:

The DECODE expression has the syntax:

SYNTAX DEFINITION:

- Receive ::= [PCOidentifier | PARidentifier] "?" (ASPidentifier | PDUidentifier) [DecodesAs] [TTCNExpression] [TimerOperations]
- DecodesAs ::= "<" DataObjectReference "~" PDUidentifier [DecodesAs] ">"

The left-hand side of the ENCODE expression shall be the name given to the user data field in the appropriate ASP or PDU type declaration.

A.15.12 Timer Management

A.15.12.1 Introduction

A set of operations are used to model timer management. These operations can appear in combination with events or as stand-alone pseudo-events. They can be applied to:

- a set of timers, which is specified by omitting the timer name;
- an individual timer, which is specified by following the timer operation by the timer name.

It is assumed that the timers used in a test suite are either inactive or running. All running timers are automatically cancelled at the end of each test case. There are three predefined TTCN timer operations: START, CANCEL and READ TIMER. More than one timer operation may be specified on a TTCN statement if necessary. This is indicated by separating the operations by commas.