ETSI
TECHNICAL
REPORT

ETR 141

October 1994

Source: ETSI TC-MTS

Reference: DTR/MTS-00020

ICS: 33.080

Key words: TTCN, conformance testing, guide

Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications The Tree and Tabular Combined Notation (TTCN) style guide

ETSI

European Telecommunications Standards Institute

ETSI Secretariat

Postal address: F-06921 Sophia Antipolis CEDEX - FRANCE
Office address: 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE
X.400: c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 92 94 42 00 - Fax: +33 93 65 47 16

Contents

Foreword

This ETSI Technical Report (ETR) has been produced by the Methods for Testing and Specification (MTS) Technical Committee of the European Telecommunications Standards Institute (ETSI).

Introduction

The main aspects found within this Tree and Tabular Combined Notation (TTCN) style guide have been established, discussed and implemented within the completed European second Conformance Testing Services program (CTS 2) File Transfer, Access and Management (FTAM) project. With the end of such a project, the responsibility of the maintenance of such Abstract Test Suite (ATS) descriptions falls on different other organizations e.g. agreement groups (former recognition arrangement), regional workshops or standardization bodies. This fact, together with the also important aspect of having very similar looking ATS descriptions, lead the CTS 2 FTAM project to the decision to produce rules for the use of TTCN and fix them in a handbook (see Annex B of the CTS 2 FTAM pilot trial handbook [31]), which was taken as a basis for the later work submitted to the European Workshop for Open Systems (EWOS), resulting in EWOS ETG 025 [19].

This ETR has been developed mainly to focus on a common style which can be used for higher layer protocols (layer 5 and above) specifically using the Remote Single layer Embedded (RSE) or Distributed Single Layer (DSL) test method. The current document is an extension of this original style guide with respect to aspects that were considered to be missing, and with the aim to make it more universal, especially to make it fully applicable and useful for ATSs on lower layer protocols.

Blank page

1 Scope

This ETSI Technical Report (ETR) is intended to support a developer of an Abstract Test Suite (ATS) using the Tree and Tabular Combined Notation (TTCN). This notation has been defined in ISO/IEC 9646-3 [3] for the purpose of writing Abstract Test Case (ATC)/ATS descriptions, used to test Open System Interconnection (OSI) protocols.

Annex E of ISO/IEC 9646-3 [3] defines a TTCN style guide (called, in this ETR, the "ISO TTCN style guide" to distinguish it from this ETR) which is intended to be used in order to avoid a basic inconsistency between TTCN styles used by different test suites specifiers. The aim of this ETR goes beyond the approach of the ISO TTCN style guide. Besides the aspect of re-usage and readability, it also covers the aspect of quality of ATSs and emphasises the analogy to software development, and by this, the necessity to make a design before writing an ATS.

The conformance testing methodology is described in ISO/IEC 9646-1 to ISO/IEC 9646-7 [1] to [8] and in ETS 300 406 [14]. Several tutorials have been developed in the past years, for the complete conformance testing process (see ETR 021 [18]), as well as for TTCN itself. This TTCN style guide is **not a tutorial in this sense** - a tutorial tries to explain **what** the features of TTCN are, whereas for readers of this ETR, it is assumed that the features of TTCN are known, therefore, guidelines are given on **how** to best use these features to achieve the intended quality aspects.

According to the informative nature of this ETR, no requirements are expressed, but rules are stated recommending particular procedures or options defined for the TTCN notation. Each rule is followed by one or more examples, which emphasise the meaning of the rule.

Most of the examples are taken from existing test suites, see e.g. ISO/IEC 8882-2 [9], ISO/IEC 8882-3 [10], EWOS-PT19/SNI023 [21], File Transfer, Access and Management (FTAM) AFT11 Responder ATS [17], I-ETS 300 313 [15] and I-ETS 300 322 [16], but where appropriate, fictional elements (i.e. elements not extracted from any test suite) are also introduced.

All test suites except the latter two Integrated Services Digital Network (ISDN) suites are written in the IS version of TTCN, which is the basis of this TTCN style guide. Because not all of the features of TTCN have changed from the DIS to the IS version, and because it is within the scope of this ETR to cover several protocols, especially from the lower layers, examples from ISDN are included. Also, because of this coverage, the use of one type of example that goes consistently through all the different features of TTCN discussed in this ETR is not attempted.

Clause 4 treats the general design aspects of an ATS. Clause 5 provides some statements about parameterization of ATSs from the viewpoint of profiles.

Clauses 6 to 14 provide guidelines for the following subjects:

- naming conventions;
- type definitions;
- test suite operations;
- aliases;
- constraint definitions;
- test cases;
- test steps;
- default trees;
- TTCN extensions.

Clause 15 treats some other aspects that are less comprehensive, but still worth mentioning. Annexes A and B contain the lists of all the rules and examples including the page numbers where they are stated, to provide a quick cross-reference.

NOTE: The guidelines described here are not provided in order to restrict the power of TTCN, or to give preference to a style used by any of the authors, but are provided in order to ease the production of more uniform looking ATSs, which are simple to read and understand, and are of a high quality.

2 References

This ETR incorporates by dated or undated reference, provisions from other publications. These references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this ETR only when incorporated in this ETR by amendment or revision. For undated references the latest edition of the publication referred to applies.

[1]	ISO/IEC 9646-1 (1994): "Information technology - OSI conformance testing methodology and framework - Part 1: General concepts" (see also CCITT Recommendation X.290 (1991)).
[2]	ISO/IEC 9646-2 (1994): "Information technology - OSI conformance testing methodology and framework - Part 2: Abstract test suite specification". (see also CCITT Recommendation X.291 (1991)).
[3]	ISO/IEC 9646-3 (1992): "Information technology - OSI conformance testing methodology and framework - Part 3: Tree and tabular combined notation".
[4]	ISO/IEC 9646-3 (1994): "Information technology - OSI conformance testing methodology and framework - Part 3: Tree and tabular combined notation - Amendment 1: TTCN extensions" (to be published as DAM1).
[5]	ISO/IEC 9646-4 (1994): "Information technology - OSI conformance testing methodology and framework - Part 4: Test realization".
[6]	ISO/IEC 9646-5 (1994): "Information technology - OSI conformance testing methodology and framework - Part 5: Requirements on test laboratories and clients for the conformance assessment process".
[7]	ISO/IEC 9646-6 (1994): "Information technology - OSI conformance testing methodology and framework - Part 6: Protocol Profile Test Specification".
[8]	DIS ISO/IEC 9646-7 (1992): "Information technology - OSI conformance testing methodology and framework - Part 7: Implementation Conformance Statements".
[9]	ISO/IEC 8882-2: (199x): "Information Technology - Telecommunication and information exchange between systems - X.25 DTE conformance testing Part 2: Data link layer conformance test suite" (2nd edition).
[10]	ISO/IEC 8882-3 (199x): "Information technology - Telecommunication and information exchange between systems - X.25 DTE conformance testing Part 3: Packet layer conformance test suite" (2nd edition).
[11]	ISO/IEC 8824 (1990): "Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1)".
[12]	ISO/IEC 8825 (1990): "Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)".
[13]	ISO 7776 (1986): "Information processing systems - Data communication - High-level data link control procedures - Description of the X.25 LAPB-compatible DTE data link procedures".
[14]	Draft prETS 300 406 (1994): "Methods for Testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".

[15]	I-ETS 300 313: "Integrated Services Digital Network (ISDN); Digital Subscriber Signalling System No. one (DSS1); Abstract Test Suite (ATS) for user of data-link-layer protocol for general application".
[16]	I-ETS 300 322: "Integrated Services Digital Network (ISDN); Digital Subscriber Signalling System No. one (DSS1); Abstract Test Suite (ATS) for user of signalling-network-layer protocol for circuit-mode basic call control".
[17]	FTAM AFT11 "Responder ATS" (under preparation by EWOS PT 15).
[18]	ETR 021 (1991): "Advanced Testing Methods (ATM); Tutorial on protocol conformance testing (Especially OSI standards and profiles)".
[19]	EWOS ETG 025: "The TTCN Style Guide and Quality Criteria - Edition 1 for EWOS".
[20]	EWOS-PT19/SNI021 (1994): "Abstract Test Suite for Transport Class 0".
[21]	EWOS-PT19/SNI023 (1994): "Abstract Test Suite for Transport Class 4".
[22]	EWOS-PT19/SNI024 (1994): "Abstract Test Suite for Transport Class 2".
[23]	EWOS TA/93/008: "EWOS Test Specification Quality Assurance Handbook" (preliminary title, to be published).
[24]	EWOS TA/92/006: "Profile Test Specifications and Conformance Test Reports".
[25]	EWOS EGCT/94/063 (1994): "Exception Report, Production of test specifications for the transport layer in profiles".
[26]	EWOS Technical Report ETG 022 (1992): "Test Specifications for Embedded Protocols in Application Profiles".
[27]	EWOS PT08 Report: "Planning for Conformance Testing for FTAM".
[28]	EWOS EGCT/92/008: "Liaison Statement from ULCT group to EWOS on FTAM Functional Test Specification".
[29]	EWOS EGCT/91/032: "EWOS/ETSI Scheme for Maintenance of Profile Test Specifications".
[30]	McCall J.A., Richards P.K., Walters G.F. "Factors in Software Quality" Vol. I-III, Rome Air Development Centre, 1977.
[31]	CTS 2 FTAM Pilot Trial Handbook.

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of this ETR the definitions of ISO/IEC 9646 [1] to [8] apply.

3.2 Symbols

The angled brackets around text, "<" "text" ">", are used in examples to indicate that the enclosed text is a symbolic variable, the value of which is defined elsewhere.

3.3 Abbreviations

For the purposes of this ETR the following abbreviations apply:

ACSE	Association Control Service Element
ASN.1	Abstract Syntax Notation One
ASP	Abstract Service Primitive
ATC	Abstract Test Case
ATS	Abstract Test Suite
CM	Coordination Message
CS	Conditional Statement
CTS	Conformance Testing Services
CTS 2	second CTS program
DSL	Distributed Single Layer
EBNF	Extended Backus-Naur Form
EGCT	(EWOS) Expert Group for Conformance Testing
EWOS	European Workshop for Open Systems
FPDU	FTAM PDU
FTAM	File Transfer, Access and Management
ICS	Implementation Conformance Statement
ISDN	Integrated Services Digital Network
ISO style guide	TTCN style guide defined in Annex E of ISO/IEC 9646-3 [3]
IUT	Implementation Under Test
MTC	Master Test Component
NM	Network Management
PCO	Point of Control and Observation
PCTR	Protocol Conformance Test Report
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation Extra Information for Testing
PM	Protocol Machine
PTC	Parallel Test Component
QoS	Quality of Service
RSE	Remote Single layer Embedded
SS7	Signalling System No.7
TCP	Test Coordination Procedure
TMP	Test Management Protocol
TPDU	Transport PDU
TSS&TP	Test Suite Structure & Test Purposes
TTCN	Tree and Tabular Combined Notation
TTCN.GR	TTCN Graphical Form
TTCN.MP	Machine Processable TTCN
UCS	Unconditional Statement

4 General ATS design aspects

4.1 Software engineering aspects

The objective of software engineering is to provide methods, tools and procedures for controlling the process of software development and for assuring the development of high-quality software. ATSs can be seen as software in a more abstract sense. Therefore, some of the procedures and methods defined by software engineering may also be applicable for the development process of ATSs.

Software engineering is mainly based on the so-called software life-cycle consisting of the following stages:

- **analysis.**
  In this stage, two sub-phases can be distinguished: planning; and requirements definition. The requirements definition identifies the basic functions of the software. During planning the project plan is established containing the life-cycle to be used, the organizational structure of the project, the preliminary development schedule, preliminary staffing requirements, etc.;

- **design.**
  There are two different types of design activities: the external design; and the internal design. The external design covers the externally observable characteristics. e.g. high level process structure of the software product The internal design involves specifying the internal structure and processing details of the software product;

- **coding.**
  The coding phase is concerned with the translation of the design specifications into source code;

- **testing.**
  The goal of testing is to assess and to improve the quality of the software product;

- **operation and maintenance.**
  As soon as the software product is completed and delivered to customers, the operation and maintenance phase starts. Activities during this phase involve enhancement of the software product, adapting the product to new environments and correcting problems.

As can be seen in the following, this software life-cycle also applies for the development of ATSs:

- **analysis.**
  The establishment of a project plan is seen as being very useful for the development of complex ATSs. The requirements definition for ATSs is given by the conformance requirements expressed in the base standard or the profile specification, and is reflected in the Implementation Conformance Statement (ICS) and requirements lists;

- **design.**
  The external design of an ATS, particularly the high level process structure, is already given by the test suite structure and test purposes document. The internal design of an ATS involves activities like defining the abstract test method, defining the mapping between test cases and test purposes, defining structure and objectives for test steps, determination of Abstract Service Primitive (ASP) types and Protocol Data Unit (PDU) types, determination of constraints to be declared, determination of parameters being used for test steps and constraints, decision if types are declared via Abstract Syntax Notation One (ASN.1) or by using the tabular method, etc.;

- **coding.**
  Coding of ATS involves expressing the dynamic behaviour of test cases/steps via test events, defining ASP and PDU types, declaring constraints, etc.;

- **testing.**
  Testing in the context of ATSs means validating that the test cases/steps within a ATS fulfil the given purpose/objective. At the moment there are only few tools known supporting the validation of ATSs to a certain degree;

- **operation and maintenance.**
  After an ATS has been standardized, defect reports may arise concerning the ATS. The handling of such defect reports is subject of this operation and maintenance phase.

A multitude of software engineering techniques are defined applicable for the different phases of a software life-cycle. Some of these techniques may (and should) be used in the ATS development process. Examples are: stepwise refinement design; structured coding techniques; testing via inspections.

Some of the aspects mentioned are handled in other clauses of this ETR in more detail.

4.2 Quality assurance aspects

The quality of ATSs written in TTCN is an important aspect and the style of writing ATSs can affect the quality in many ways.

As mentioned in subclause 4.1, the production of an ATS is similar to the production of software. For the latter, quality models have been developed in recent years, e.g. by McCall [30]. The material presented in this subclause is essentially taken from the EWOS test specification quality assurance handbook, EWOS TA/93/008 [23], in which the applicability of the quality model for software production in McCall [30] to ATS development has been investigated[1] and the model has been developed further.

NOTE: The applicability of EWOS TA/93/008 [23] goes far beyond the verification of a TTCN style.

Table 1 lists the quality factors contributing to the overall quality of an ATS. All factors, except perhaps "Correctness (F1)", are directly affected by the recommendations given in this ETR. Syntactic and semantic correctness is assumed from an ATS, whether or not the author(s) make use of a style guide, but it is not the direct aim of such a guide.

Table 2 gives a set of more concrete criteria having an influence on the quality factors. Table 3 gives the relation of which quality criterion is applicable to which quality factor.

The rules and recommendations stated in this ETR are intended to support at least one quality criterion, but the rules are not intended to originate from the convenience of an author. However, it lies in the nature of the criteria (respectively in the ATS production itself), that extensive stressing of the use of some of the criteria may limit the usage of other criteria. Such conflicts cannot be solved in a style guide, but the weightings of the quality factors need to be defined according to the individual policy and requirements of an ATS development project.

Table 1: Quality factors

No	Factor	Definition
F0	Usability	the effort required to learn and understand an ATS.
F1	Correctness	the extent to which an ATS satisfies its specification and fulfils the user's mission objectives.
F2	Maintainability	the effort required to locate and fix an error in an ATS.
F3	Testability	the effort required to test an ATS to ensure it performs its intended function.
F4	Flexibility	the effort required to modify an ATS.
F5	Portability	the effort required to transfer an ATS from one system environment to another.
F6	Reusability	the extent to which an ATS can be used within other applications (profiles or protocols, test methods).

[1] McCall [30] identifies a total of 11 factors for software programs. According to EWOS Test Specification Quality Assurance Handbook [23], efficiency, integrity, interoperability and reliability are not seen to be factors for the quality of ATSs.

Table 2: Quality criteria

No	Criterion	Definition
C0	Traceability	those attributes of an ATS that provide a link from the requirements to the realization with respect to the specific development and operational environment.
C1	Completeness	those attributes of an ATS that provide full realization of the required functionality.
C2	Consistency	those attributes of an ATS that provide a uniform design techniques and notation.
C3	Simplicity	those attributes of an ATS that provide a realization of functionality in the most understandable manner (usually the avoidance of practices which increase complexity).
C4	Generality	those attributes of an ATS that provide breadth to the defined functionality.
C5	Instrumentation	those attributes of an ATS that provide for the possibility to identify errors or unexpected situations.
C6	Self-descriptiveness	those attributes of an ATS that provide an explanation of the realization.
C7	Operability	those attributes of an ATS that determine the operation and procedures concerned with the operation of the ATS.
C8	Training	those attributes of an ATS that provide a transition from current operation or initial familiarization.
C9	System independence	those attributes of an ATS that determine its dependency on the system environment.

Table 3: Applicability of quality criteria

Usability (F0) - operability (C7); - training (C8).	Flexibility (F4) - generality (C4); - self descriptiveness (C6).
Correctness (F1) - traceability (C0); - completeness (C1); - consistency (C2).	Portability (F5) - self descriptiveness (C6); - system independence (C9).
Maintainability (F2) - consistency (C2); - simplicity (C3); - self descriptiveness (C6).	Reusability (F6) - generality (C4); - self descriptiveness (C6); - system independence (C9).
Testability (F3) - simplicity (C3); - instrumentation (C5); - self descriptiveness (C6).	

4.3 Supporting documents

In previous ATS development projects it has become obvious that the existence of only the ISO-structured ATS document (see ISO/IEC 9646-3 [3]) is not sufficient. The reasons for this are mainly due to the fact that the development of an ATS should be treated like any other (software development) project, where it is required that as well the code itself other supporting documents exist. Examples of ATS supporting documents are:

- ATS development document;
- test realizing document;
- ATS cross reference document;
- ATS exception document.

At least the ATS development document and the test realizing document should be produced during/after ATS development.

Similar to software development projects there are design decisions that should be fixed before starting the development of an ATS. In particular, if there is a team of ATS developers working in parallel on separate parts of the same ATS, this is a very important aspect in order to assure consistency of the different parts of the ATS. Some potential candidates for design decisions are as follows:

- naming conventions (see Clause 6);
- type definitions via ASN.1 or tabular method;
- mapping of test purposes to test cases;
- coverage;
- test suite structure;
- verdict assignment;
- test suite parameters;
- test suite variables;
- post/preambles;
- default trees;
- test suite operations;
- test steps (structuring, nesting, etc.);
- constraints;
- use of timers.

The design decisions taken should be unambiguously documented in the ATS development document. The ATS development document is a "living document". That means that during the development of an ATS, the need for additional design decisions may arise, or existing design decisions may have to be changed to meet unforeseen requirements. In each case, the ATS development document needs to be updated and the updated document should be made available to each developer involved in the ATS development.

The above list cannot be exhaustive as, dependent on an ATS scope, additional design decisions may be required. For instance, if a protocol makes use of checksums, there should be a decision about how checksums are handled within the ATS.

Documenting all design decisions in the ATS development document makes the ATS maintenance easier. An ATS maintainer may not have been involved in the ATS development. Therefore, if there is a need for changes within the ATS after its completion, the maintainer can get a better understanding of the ATS by carefully studying the ATS development document.

During the development of an ATS, a document should be prepared containing cross reference lists. There may be cross reference lists for the following relations:

a)	test case	←→	test step;
b)	test case/step	←→	constraint;
c)	test case/step	←→	test suite operation;
d)	test case/step	←→	test suite variable;
e)	test case/step	←→	test suite parameter;
f)	test case/step	←→	timer.

As a minimum, the ATS cross reference document should contain relations a) and b). Cross reference lists, especially for relation b), would allow ATS developers and maintainers to easily identify the effect of a change on the test suite. It is strongly recommended that TTCN tools should support the preparation of such cross reference lists in future.

Before starting the preparation of the ATS development document and the ATS cross reference document, a contents checklist should be made available. The contents checklist should contain all the topics being covered by the corresponding document. After completion, the document can then be checked for completeness via the contents checklist.

As already mentioned, both the ATS development document and the ATS cross reference document support the ATS maintenance. The information contained in both documents may also be helpful for the test realizer. In addition to this information, a test realizing document should be prepared after completion of the ATS. The test realizing document should contain specific information useful for realizing the tests, i.e. when mapping the ATS to an ETS. Such information may be:

- timers used within the ATS and their value range;
- time critical test cases;
- parameterization/chaining of constraints;
- usage of modified constraints;
- PDU encoding, i.e. reference to the standards that the PDUs are derived from, assumptions on Test Management Protocol (TMP), if used (e.g. kind of transfer: serial; parallel; inbound; outbound);
- assumptions on initialization/resetting of upper tester, if used;
- reference list of documents the development of the ATS was based upon;
- reference to Protocol Conformance Test Report (PCTR) proforma;
- exceptions.

If the exceptions detected during ATS development are of more general nature (i.e. not only concerning test realization) the exceptions should be described in a separate document, the ATS exception report. Such exceptions may be:

- test purposes not realized;
- restrictions on test suite parameters;
- shortcomings of TTCN influencing the ATS development;
- ambiguities of the base standard/profile (the ATS development is based on) influencing the ATS development;
- differences to already existing ATS based on the same base standard/profile;
- defect reports raised (concerning standards the ATS development is based on).

An example for such an ATS exception report is EWOS EGCT/94/063 [25], prepared by the EWOS project team on the production of test specifications for the transport layer in profiles (EWOS PT19).

In ETS 300 406 [14], it is stated that documentation of the conventions used in the ATS shall be provided. This documentation should contain as much information as possible, in order to help humans to understand the ATS. In this sense, all four documents discussed above contain information which can be used as input to the ATS conventions documentation.

NOTE: The notion of "document" in this context includes the possibility that the related information is part of the ATS specification document.

5 Starting ATS development

5.1 Introduction

The ATS production phase can be regarded as consisting of three phases:

a) starting phase;
b) test development phase;
c) ATS maintenance phase.

While topics a) and c) are outside the scope of this document, topic b) is discussed in more detail[2]. Since the starting phase has some influence on the test development phase, this phase is discussed in this subclause in some detail.

The starting phase comprises aspects as:

- determination of conformance requirements;
- achievement of the appropriate test coverage of the conformance requirements by defining suitable test groups;
- development of test group objectives;

[2] Topic c) is only covered in so far that rules are given in order to ease the later maintenance process.

- development of test purposes reflecting the test group objectives;
- determination of test suite parameters.

Some of the decisions made during the starting phase directly influence the ATS development and should therefore be documented in the ATS development document.

At the end of the starting phase a Test Suite Structure & Test Purposes (TSS&TP) document will be available, clearly outlining the test suite structure and the set of test purposes applicable to a complying ATS. However, Such a TSS&TP document does not handle last aspect given above, i.e. the determination of test suite parameters. Test suite parameters are constants derived from the Protocol Implementation Conformance Statement (PICS) and/or Protocol Implementation eXtra Information for Testing (PIXIT). Some ideas concerning test suite parameterization are discussed below.

Experience has shown that parameterization of an ATS is an efficient mechanism:

- to ensure applicability to different Implementations Under Test (IUTs);
- to make use of existing test cases, to realize a profile-specific ATS;
- to realize specific parts of application layer ATSs for common parts of embedded ATSs.

These three aspects need to be considered in advance of the test development phase as later changes to the ATS may result in an expensive task.

Subclauses 5.2 to 5.5 give an indication of why test suite parameterization is considered an efficient mechanism with respect to the aspects given above.

5.2 Ensuring applicability to different IUTs by ATS parameterization

The parameterization of an ATS, on parameters varying from IUT to IUT, is obvious and already introduced in various existing ATSs. Nevertheless it should be noted that besides these more obvious IUT specific parameters like:

- address parameter;
- access parameter;
- passwords.

Also other parameters should be considered, which are subject to change and therefore should not be fixed. Such parameters can be divided into two classes:

a) test suite parameters which are dependent upon operating system;
b) test suite parameters for determining the type of protocol parameters.

Class a) parameters result from the fact that an implementation maps certain parameters onto underlying operating system parameters and is therefore limited to these requirements. Examples are: a file name which should not start with a number; a password to contain at least one graphical symbol; etc. As the content of such a parameter is not the subject of testing, it should not be fixed within an ATS, but should be described in the test realizing document.

Class b) parameters result from the fact that alternative choices for types of parameters are defined in the protocol standard, e.g. passwords may be graphic or octet strings, content of files may be graphic/printable/visible strings. In most cases a specific profile/functional standard (in the following, abbreviated by "profile standard") will limit the amount of choices to one alternative or to a reduced set of alternatives. Within an ATS for this profile standard, these restrictions will not result in any testing problems, provided that the alternatives are chosen within these limits. However, the extension of such an ATS to a different profile standard, using different restrictions, will result in a problem. Therefore, a warning should be given in the ATS development document or in the exception report.

5.3 Profile standard parameterization

As most of the currently available standardized protocols reach a size and complexity which does not make sense to be implemented in full, arrangements (profile standards) have been agreed for subsetting the full functionality of the protocol into smaller units which can be implemented more easily.

ATSs (and also resulting ETSs) very often are developed on the basis of market requirements arising from the availability of profile standards. In this case, a real base standard ATS, covering all conformance requirements expressed in the relevant base standard, only exists if the set of profile standards for the protocol covers the whole functionality of the base protocol. This means that the development of a real base standard ATS is a stepwise approach.

If a new profile standard is defined, then it may contain capabilities and options which are also included in some already available profile standards. If there is an ATS available covering these profile standards, then the test cases covering test purposes concerning the capabilities and options common to both the new profile standard and the already available set of profile standards can also be used for the new profile standard. This means that only test cases for capabilities and options which are not common, need to be added to the ATS.

For profile testing, only parts of an ATS developed in this way may be useful, i.e. test cases applicable to the specific profile have to be selected from the set of all test cases contained in the ATS. In TTCN, test case selection expressions have been foreseen for this purpose. Such selection expressions are mainly based on test suite parameters and test suite constants.

It is difficult to provide more specific rules, but as a minimum, a set of test suite parameters which are directly related to profile standards should be considered. Such parameters should be used carefully and kept as test suite parameters, together with some notes on restrictions, outlined in the ATS development document or in the exception report. In addition, if possible, sets of these parameters should be created, which may then be exchanged from one profile standard to another.

List of possible candidates:

- functional units;
- Quality of Service (QoS) criteria (e.g. transport QoS);
- parameter for classification of service (e.g. in FTAM transfer or access class);
- type of bearer service provided (ISDN);
- defined objects (e.g. in Network Management (NM) managed objects).

5.4 Application parameterization for embedded ATS

As application layer testing always includes the testing of the embedded layer protocols (not always fully exhaustive) it is desirable to have common embedded layer ATS specifications (the so-called common part) which are, as far as possible, application layer independent. Such common embedded layer ATS specifications then can be complemented by ATS specifications covering the application protocol specific aspects (the so-called specific part). Since the common part and the specific part cannot be absolutely independent, both parts need to be aligned. One way to achieve this is via test suite parameters. Test suite parameters used for aligning common and specific parts should be described in detail in the ATS development document. EWOS has produced a technical report and a technical guide (see EWOS ETG 022 [26]) in which various parameterization techniques are described.

6 Guidelines on naming conventions

6.1 Introduction

The problem of agreeing naming conventions is known from many types of activities, and the reasons for such conventions are similar for authors of TTCN ATSs. Normally, there are multiple authors involved, sometimes these authors are from different organizations, the maintenance is usually performed by different people than the creation, the implementation can be expected to be in different environments and on different test tools, etc.

Each of the quality factors of table 1 (except "Correctness (F1)") may be affected by naming conventions. It is known, however, that there are nearly as many naming conventions or preferences as there are authors. This ETR provides some general principles and give some generic examples to use the limited character set of TTCN identifiers within useful naming schemes to contribute to the quality factors.

NOTE: The uniqueness of names for different TTCN objects is assumed to be understood by the reader of this ETR, since it is a requirement from ISO/IEC 9646-3 [3].

6.2 General rules for naming conventions

Rules 1, 2 and 3 contain general requirements that naming conventions in an ATS should fulfil.

RULE 1: Statement of naming conventions
Naming conventions should be explicitly stated. Naming conventions should not exist only for a single ATS, and the reader of an ATS should not be forced to "derive" the rules implicitly They naming conventions should be part of the ATS conventions contained in the ATS specification document.

RULE 2: Coverage of naming conventions
Naming conventions stated should, as a minimum, cover the following TTCN objects: - test suite parameters/constants/variables; - test case variables; - formal parameters; - timers; - PDU/ASP/structured types; - PDU/ASP/structured types constraints; - test suite operations; - aliases; - test case/test step identifiers.

There may be TTCN object types for which there are so few instances defined in a particular ATS, that an explicit naming convention does not appear to be necessary, e.g. for Points of Control and Observation (PCOs). But in ATSs of "normal" size, there are so many instances of the objects given in rule 2, that naming conventions for these objects seem to be appropriate.

RULE 3: General properties of naming conventions

a) **Protocol standard aligned**

When there is a relationship between objects defined in the ATS and objects defined in the protocol standard, e.g. PDU types, the same names should be used in the ATS if this does not conflict with the character set for TTCN identifiers or with other rules. In case of a conflict, similar names should be used.

b) **Distinguishing**

The naming conventions should be defined in such a way, that objects of different types appearing in the same context, e.g. as constraint values, can be easily distinguished.

c) **Structured**

When objects of a given type allow a grouping or structuring into different classes, the names of these objects should reflect the structuring, i.e. the names should be composed of 2 or more parts, indicating the particular structure elements.

d) **Self-explaining**

The names should be such that the reader can understand the meaning (type/value/contents) of an object in a given context. When suffixes composed of digits are used, it is normally useful to have some rule expressed explaining the meaning of the digits.

e) **Consistent**

The rules stated should be used consistently throughout the document, there should be no exceptions.

f) **Appropriate name length**

Following the above rules extensively may occasionally lead to very long names, especially when structuring is used. The names should still be easily readable. When TTCN graphical form (TTCN.GR) is used, very long names are very inconvenient.

NOTE: Also, test tools may not be able to implement very long identifier names, which is an important aspect in this context.

EXAMPLE 1: Sample of simple naming schemes applicable to individual TTCN object classes.

The items of this example are used to show some possibilities to define distinct naming classes, using the restricted character set allowed for TTCN identifiers. Indication of preference to use a specific naming scheme for a given class of TTCN objects is not intended. The naming convention chosen for a particular TTCN object class should be unique and should be used consistently for that class.

a) Use only capital letters, separating name parts with "_" when appropriate. This convention is used for test suite parameters or test suite constants in some test suites. Specific examples are:

LCI	(logical channel identifier);
TVC_OR_OVC	(two-way or outgoing logical channel);
CREF	(call reference).

b) Use prefixes like "TSP_" (for test suite parameter) or "TSC_" (for test suite constant), following a) otherwise. An example is:

TSP_fname1	(name of first file);
TSP_SP_A	(signalling point code A).

c) Use capital letter for first character of a name part, lower case or digits for the other characters; separate name parts by "_" when appropriate. This is used e.g. for test suite variables in some test suites. An example is:

Ready_To_Receive.

d) Like c), but use only lower case characters and possibly digits. This is used as a convention for test case variables or formal parameters in some test suites. An example is:

sls_code	(signalling link selection code).

e) Like c), but without usage of "_" as separator. This is the ASN.1-manner of naming. An example is:

FileIdentifier	(file identifier).

Variations of c), d) and e) are possible, where only the first part of the name starts with lower case character, e.g.:

- fileIdentifier; or
- file_Identifier.

NOTE: See Annex E of ISO/IEC 8824 [11] for examples and hints to naming of ASN.1 objects.

EXAMPLE 2: Suffixes composed of digits.

Use digits to indicate starting and ending state of a preamble:

PREAMBLE_<nn>_<mm>;

<nn>:	1 or 2 digits to indicate the protocol state in which the preamble starts;
<mm>:	1 or 2 digits to indicate the protocol state in which the preamble ends.

EXAMPLE 3: Structured naming scheme for ISDN data link ASPs.

<ASP type>:	DL_<ASP name><ASP suffix>	
<ASP name>:	EST	(ESTABLISH); or
	REL	(RELEASE); or
	DAT	(DATA); or
	UDAT	(UNIT DATA)
<ASP suffix>	RQ	(REQUEST); or
	IN	(INDICATION); or
	CO	(CONFIRMATION).

EXAMPLE 4: Structured naming scheme for FTAM and Association Control Service Element (ACSE) ASPs.

<ASP type>:	F_<FASP name><ASP suffix>; or	
	A_<AASP name><ASP suffix>; or	
	<others>.	
<FASP name>:	<FPDU name>;	
<FPDU name>:	INI	(INITIALIZE); or
	TER	(TERMINATE); or
	<others>.	
<AASP name>:	ASC	(ASSOCIATE); or
	RLS	(RELEASE); or
	<others>.	
<ASP suffix> req	(REQUEST); or	
	ind	(INDICATION); or
	rsp	(RESPONSE); or
	cnf	(CONFIRMATION).

6.3 Specific rules for naming conventions

The following rules contain some specific recommendations for particular TTCN object types.

6.3.1 Naming test suite parameters/constants/variables test case variables and formal parameters

	RULE 4: Specific naming rules for test suite parameters/constants/variables test case variables and formal parameters
a)	The name should reflect the purpose/objective the object is used for.
b)	If the type is not a predefined one, it is useful that the name reflects the type, too.
c)	It could be useful, that the individual naming conventions are not the same for all object classes this rule applies to. e.g. use upper case letters for test suite parameters/constants, and use one of the other possibilities presented in example 1 for other object classes.

EXAMPLE 5: Names for parameters and variables.

a) MOD8
Test suite constant indicating "modulo 8" counting;

b) IUT_ACTS_DTE
Test suite parameter indicating that the IUT acts as a DTE;

c) Loop_Count
Test case variable used for counting repetitions in a test case;

d) a_ascreq
Formal parameter of type "A_ASCreq".

6.3.2 Naming timers

RULE 5: Specific naming rule for timers
If the timer is not defined in the protocol to be tested, the name should reflect the objective of the timer used for testing. NOTE: There is no need to indicate the object type "timer" in the name, since timers only occur together with timer operations.

EXAMPLE 6: Timer names:

a) START TRESP
Start waiting for response from the IUT;

b) CANCEL INACT
Cancel inactivity timer.

6.3.3 Naming PDU/ASP/structured types

RULE 6: Specific naming rule for PDU/ASP/structured types
As far as applicable, derivation rules or mapping tables should be used to relate the names of the types to the corresponding objects in the protocol or service definition. NOTE: There may be types, e.g. erroneous PDU types, that do not relate to an object in the protocol or service definition.

EXAMPLE 7: PDU/ASP/structured types.

	Used in ATS	**Used in protocol/service description**
a)	DL_EST_RQ	DL-ESTABLISH-REQUEST;
b)	SETUP	Setup;
c)	Clear_Confirmation	CLEAR CONFIRMATION;
d)	ERROR	-;
e)	CDPN	Called party number;
f)	FINIRQ	F-INITIALIZE request;

6.3.4 Naming PDU/ASP/structured types constraints

RULE 7: Specific naming rule for PDU/ASP/structured types constraints
Rules should be stated to derive the names from the names of the corresponding type definitions. It is often possible to use the type name plus an appropriate suffix reflecting the specific constraint value. In case of lengthy names, useful abbreviations or a defined numbering scheme can be chosen.

EXAMPLE 8: PDU/ASP/structured types constraints.

	Type Name	**Constraint name**	**Comment**
a)	result	Result_success	(result is "success");
b)	Clear_Confirmation	CLRC_0	(basic clear confirmation);
c)	Packet_Header	HDR_ANY_QD	(header with any Q/D-bit);
d)	HEADER_8073	HD_8073_CR_INVCLASS	(8073 CR header with invalid class);
e)	FINIRQ	FINIRQbase	(F-INITIALIZE request base constraint).

6.3.5 Naming test suite operations

RULE 8: Specific naming rule for test suite operations
The name should reflect the operation being performed. i.e. the name should indicate an activity, not a status. This can be achieved e.g. by using appropriate prefixes like "check", "verify", etc.

EXAMPLE 9: Test suite operation names.

a)	set_result	(set result value);
b)	VER_VALID_HDR	(verify valid header format);
c)	CONCAT_FIELDS	(concatenate 2 bit fields to form a single one);
d)	ck_svc_fn	(check service class against functional unit).

6.3.6 Naming aliases

RULE 9: Specific naming rule for aliases
The name should reflect that aspect of its expansion, that is important in the situation where the alias is used. Derivation rules should be provided to derive the alias name from its macro expansion or from the name of an embedded ASP/PDU. See also Clause 9.

EXAMPLE 10: Alias names.

	ASP/PDU identifier	**Alias Identifier**
a)	DL_ESTABLISH_REQ	EST_REQ

The important aspect is the ASP as a whole, and the alias is only a short form of the name;

	ASP/PDU identifier	**Alias Identifier**
b)	DL_DATA_IND	CALLP

The important aspect in the situation where this alias is used is the call request packet embedded in the ASP, and the alias name reflects the embedded PDU type.

NOTE: The alias identifier needs to be different from the PDU type identifier.

6.3.7 Naming test cases

In this context, the naming of test cases has 2 parts:

- test case identifier;
- test group reference.

The grouping/structuring of test purposes is performed in the TSS&TP document. It is a requirement from ISO/IEC 9646-2 [2], that the naming scheme of the test cases follow the naming scheme in the TSS&TP document. With respect to the naming of the constituents of the test group reference, no guidance is given here, except that it is recommended that the suite identifier is included in the test group reference, therefore, also implicitly identifying the protocol tested in the ATS.

With respect to the naming of the test case identifier and its relation to the test group reference, no explicit rule is expressed, but 2 basic examples are presented.

EXAMPLE 11: Test case names.

	Test group reference	**Test case identifier**
a)	FTAM/R/CA/WR/	FTAM_R_CA_WR_1

The test case identifier reflects the complete path of groups in which it is contained. A suffix distinguishes the test cases (and the test purposes) of the lowest level subgroup;

	Test group reference	**Test case identifier**
b)	R1/R1_Proper/	P1_101

In this example coming from the X.25 test suite ISO/IEC 8882-3 [10], the test suite identifier is unfortunately not included in the test group reference, but the "P" in the test case identifier identifies the "X.25 packet level" at least. The groups in the suite are mainly related to X.25 states like "R1", and these states are numbered starting from 1. The "1" after the "P" in the identifier corresponds directly to the group associated to state R1, which in fact is group 1. The groups are divided into "proper", "improper" and "inopportune" subgroups. The "1" after the "_" in the identifier corresponds to the subgroup "proper". The remaining 2 digits in the identifier are used to number the test cases within the subgroup. Information about the groups and their numbering is contained in the "Test suite information" clause of the ATS ISO/IEC 8882-3 [10].

6.3.8 Naming test steps

RULE 10: Specific naming rule for test steps
The name should reflect the objective of the test step.

EXAMPLE 12: Test step names.

a) LC_VERIFY

Verify that the received logical channel number is not yet in use;

b) D1_PVC_PREAMBLE

Put the IUT in state D1, only used for PVC logical channels;

c) UTIL_Rec_data(num,size:INTEGER)

Lower tester is expecting to receive "num" TSDUs of "size" octets;

d) Resconnect

Lower tester establishes an FTAM association.

7 Guidelines on type definitions

7.1 Use of ASN.1 or tabular format

One of the basic decisions to be taken when writing an ATS is whether or not to use the ASN.1 format for type definitions. ASN.1 was created mainly for the needs of user-oriented data communications, and is therefore more common in higher layer protocols than in lower layer protocols. However, some of its features also makes ASN.1 useful for the lower layer protocols. Except for the meta-type **PDU**, every type and constraint value expressed in tabular TTCN format can also be expressed in ASN.1. The lack of **PDU** can be easily overcome in ASN.1 by defining a particular type containing all PDU types of the ATS as alternatives, using the CHOICE feature, as shown in example 13.

EXAMPLE 13: Use of CHOICE to simulate the meta-type PDU**.**

ASN.1 ASP type definition		
ASP Name	:	ANY_PACKET
PCO Type	:	
Comments	:	Choice of all possible PDU types in the X.25 packet layer
Type definition		

```
CHOICE {
        callRequest,    -- call request packet
        callConfirm,    -- call confirmation packet
        clearRequest    -- clear request packet
        clearConfirm,   -- clear confirmation packet
        data            -- data packet
}
```

ANY_PACKET can be used instead of **PDU** in a type declaration, when appropriate.

NOTE: The application of **ANY_PACKET** and **PDU** is still not completely equivalent: in TTCN it is not allowed to refer to subfields of a field having type **PDU**, while this is possible for subfields of a field having type **ANY_PACKET**.

Except for the use of CHOICE, there are other features in ASN.1 going beyond the possibilities of tabular TTCN, e.g.:

- OPTIONAL
  to indicate optional elements (while in tabular TTCN, all structured type elements, PDU fields and ASP parameters are optional by definition);

- DEFAULT
  to give a default value to an element inside a type definition;

- SET or SET_OF
  to specify sets of values without giving a requirement for the order in which they appear;

- sub-typing
  to restrict possible values of a type with respect to a parent-type.

Because of the more powerful methods of specifying types and values, some experts recommend the use of **only** ASN.1 for these purposes in TTCN ATSs. This ETR does not make this recommendation, but highlights some advantages, and also some problem areas for the use of ASN.1.

There is one aspect that restricts the generality of the use of ASN.1 in TTCN ATSs compared to the tabular method: types referred to in an ASN.1 type declaration may only be other ASN.1 types (e.g. see production 57 in Annex A of ISO/IEC 9646-3 [3]), while types referred to in a tabular TTCN type definition may be any other type, defined in ASN.1 or in tabular format. This leads to a problem of compatibility in assignments and expressions, shown in example 14.

EXAMPLE 14: Compatibility of ASN.1 and tabular types.

(TCVAR1:= ASP1.par2).

What happens if the test case variable TCVAR1 has been defined as an OCTET STRING type using ASN.1, while parameter par2 has type OCTETSTRING, but the ASP ASP1 is defined using the tabular format?

Another problem could arise with the global meaning of named numbers defined in ASN.1. Example 15 shows the declaration of an INTEGER type including a named number list.

EXAMPLE 15: Type with named number list.

ASN.1 type definition
Type name : INT **Comments** :
Type definition

INTEGER {
    on(0),
    off(1)
}

Unlike e.g. the names of the elements of a **SEQUENCE**, the identifiers **on** and **off** have a global meaning in the ATS (see ISO/IEC 9646-3 [3], A.4.2.6). Since these named numbers have no meaning for the type, but only for value references of that type, the implicit use of those global elements could be circumvented by defining INT as an **INTEGER** without named values, but also defining **on** and **off** as test suite constants, and using these constants as constraint values for INT, when appropriate.

For the **ENUMERATED** type, named numbers are also used, but in this case they are a mandatory part of the type declaration. So one should either accept the enclosed global names, or replace the **ENUMERATED** type by an **INTEGER** type (when applicable).

The ASN.1 feature of **sub-typing** allows to specify a type by giving a subset of values of a parent-type. This is similar to the type & restriction for tabular TTCN simple types, but is more general and more powerful. The restriction for a simple type can be a length restriction (for strings), an integer range, and a simple value list, where the latter can only contain literal values (as opposed to values with a name, like test suite constants). Sub-typing in ASN.1 can also be applied to types with a substructure, and the values of a value list need not be literal values. See example 16.

EXAMPLE 16: Use of sub-typing.

ASN.1 type definition		
Type name	:	iut_addr
Comments	:	Address of IUT
Type definition		
address(ShortIUTAddr, LongIUTAddr) -- where address is a structured address format		

NOTE: Giving a value list inside a type acts like an **OR** condition, when a field of this type is contained in a PDU (and the PDU constraint does not further restrict the allowed value to a single one). In this way, a single line could be used in some dynamic behaviour to replace several received event lines, each one specific to a single value of the value list.

Rule 11 now gives some criteria related to the use of ASN.1:

	RULE 11: Selecting the ASN.1 format for type definitions
a)	If the protocol standard uses ASN.1 to specify the PDUs, the ATS specifier should also use ASN.1.
b)	If the protocol standard does not use ASN.1, check carefully whether features of ASN.1 that the tabular format of type definition does not present are necessary in the ATS, or could ease the design and understanding of the definitions as a whole. Check especially whether fields or parameters have to be specified, the order of appearance of which, in a received ASP/PDU, cannot be predicted. If any of these conditions apply, use ASN.1 for type and ASP/PDU type declarations.
c)	Use the option of "ASN.1 ASP/PDU type Definitions by Reference" whenever applicable.
d)	Example 14 shows a compatibility problem that could occur, when ASN.1 type declarations as well as tabular type declarations are used in an ATS. Use the ATS Conventions to describe how this compatibility problem is handled in the ATS, i.e. whether in expressions and assignments entities defined in ASN.1 are only related to entities defined in ASN.1 or not.

EXAMPLE 17: Selection of ASN.1 type definitions.

The conformance test suites ISO/IEC 8882-2 [9] and ISO/IEC 8882-3 [10] for the X.25 layers 2 and 3 use the ASN.1 format of structured type and PDU type definitions, although the protocol is not specified using ASN.1.

7.2 General guidelines on type definitions

Rule 12 gives some more guidelines on type definitions, independently of whether ASN.1 is used or not.

RULE 12: Further guidelines on type definitions

a) Use simple type or ASN.1 type definitions whenever an object of a base type with given characteristics (length, range, etc.) will be referenced more often than once.

b) Use the optional length indication in the field type or parameter type column of structured type and ASP/PDU type definitions whenever the base standard/profile restricts the length.

NOTE 1: This can often be achieved by references to simple types.

c) Map the applicable ASPs/PDUs from the service/protocol standard to corresponding ASP/PDU type definitions in the ATS.

NOTE 2: It may happen that not all ASPs/PDUs of a service/protocol standard are applicable to a particular ATS for the related protocol. It may also happen that additional ASP/PDU type declarations are necessary, e.g. to create syntactical errors.

d) Map the structure of ASPs/PDUs in the service/protocol standard to a corresponding structure in the ATS.

NOTE 3: This mapping is not always one-to-one, e.g. because a field in the PDU definition of the protocol standard is always absent under the specific conditions of an ATS. But it should normally not happen, that a structured element in the protocol standard is expanded using the "<-" macro expansion, so that the individual fields are still referenced, but the structure is lost in the ATS.

EXAMPLE 18: Use of simple/ASN.1 type definitions.

The conformance test suite ISO/IEC 8882-3 [10] defines the ASN.1 type "Address" as HEX STRING (SIZE 1 .. 15). All ASN.1 type/ASN.1 PDU type declarations containing an address, refer to this type and the allowed length range of the address is implicitly covered for each reference.

EXAMPLE 19: Mapping of PDU types in the protocol specification.

The conformance test suite ISO/IEC 8882-2 [9] maps all frames defined in the protocol standard ISO 7776 to corresponding ASN.1 PDU type definitions, keeping the structure with respect to subfields (I_Frame, RR, RNR, REJ, SABM, SABME, DISC, DM, UA, FRMR). Four additional ASN.1 PDU types are declared to create erroneous frames (Unformated_Frame_Type, RR_L, RNR_L and REJ_L).

EXAMPLE 20: Specific type definition samples.

The following 5 tables give samples of specific type definition.

Structured type definition		
Type name : CAUSE_TYPE		
Comments : CAUse information element		
Element name	**Type definition**	**Comments**
CAU_I	OCTETSTRING[1]	Identifier
CAU_L	OCTETSTRING[1]	Length
CAU_E3_LOC	OCTETSTRING[1]	Location
CAU_CV	OCTETSTRING[1]	Cause value
CAU_DI	OCTETSTRING[1]	Diagnostics

PDU type definition		
PDU name : REL		
PCO type :		
Comments : RELEASE message		
Field name	**Field type**	**Comments**
PD	OCTETSTRING[1]	Protocol discriminator
CR	CREF_TYPE	Call reference
MT	OCTETSTRING[1]	Message type
CAU	CAUSE_TYPE	Cause
FAC	OCTETSTRING[2..252]	Facility
DSP	DSP_TYPE	Display
UUI	OCTETSTRING[2..131]	User-user information
Detailed comments: This PDU has global significance when used as the first clearing PDU. The CAU field is mandatory if the REL PDU is the first clearing PDU or if the REL PDU is sent as result of Timer T305 expiry. The fields FAC, DSP and UUI are optional.		

ASN.1 type definition
Type name : I_Field
Comments : Information Field of an I Frame, containing unformatted data or an X.25 packet
Type definition

CHOICE {
    unformatted OCTETSTRING(SIZE(1..4100)),
    packet PDU_LIST -- List of allowed PDU types
}

ASN.1 PDU type definition
PDU name : I_Frame
PCO type :
Comments :
Type definition

[0] SEQUENCE {
    addressFrame_address
    control I_Control
    user_data I_Field
}

ASP type definition		
ASP name : N_UNITDATA_REQ		
PCO type : NET		
Comments : (Connectionless) network service data request		
Parameter name	**Parameter type**	**Comments**
NS_SRC_ADDR	NETADDR	Source address
NS_DEST_ADDR	NETADDR	Source address
NS_USERDATA	PDU	NS user data containing Transport PDU (TPDU)

8 Guidelines on test suite operations

Within the currently available IS version of TTCN, test suite operations may be described simply by using natural language. As natural language is known to lead to definitions which are:

- non-precise;
- ambiguous.

The use of natural language within ATSs should be avoided. As long as the standardization bodies do not provide and require a precise specification language, the developer of an ATS is free to use any specification language in test suite operations, but the ATS developer should always keep in mind the possible deficiencies mentioned above**[3)]**.

Nevertheless, so far, the developer of a test suite operation should consider the following rules:

	RULE 13: Specification of test suite operations
a)	Use a test suite operation only if it cannot be substituted by other TTCN constructs.
b)	Write down the rationale/objective of the test suite operation. Reference standards if applicable.
c)	Classify and simplify algorithm. Split test suite operation if too complex.
d)	Choose an appropriate specification language depending on the rationale/objective: - predicates for Boolean tests; - abstract data types for manipulation of ASN.1 objects; - programming languages for simple calculation.
e)	Check/proof the test suite operation: - is the notation used known/explained; - are all alternative paths fully specified; - is the test suite operation returning a value in all circumstances; - are error situations covered (empty input variables, etc.).
f)	State some evident examples.

NOTE: Test suite operations are not allowed to have any side effects (see also ISO/IEC 9646-3 [3], section 10.3.4), which particularly means that the values of test suite/test case variables may not be altered in a test suite operation.

3) In the ISO/IEC 9646-3 [3] TTCN PDAM2 editing meeting, it was decided that the proposed BNF as well as the use of free text is allowed to specify test suite operations. PDAM2 was progressed to the DAM status.

EXAMPLE 21: Test suite operations.

Test suite operation definition

Operation name	:	VP_LENGTH(hd:HEADER_8073)
Result type	:	INTEGER
Comments	:	Result is the length of the variable part in a TPDU

Description

```
VP_LENGTH(hd:HEADER_8073)
{
 return (HEX_TO_INT(hd.LI) - LENGTH_OF(hd.FIXED_PART);
}
```

Detailed comments:

The test suite operation is only applicable if hd has a structure that contains at least the 2 string components LI and FIXED_PART, and if LI is non-empty.

The LI field in the header of the 8073 TPDU contains the total length of the header. The test suite operation extracts the contents of this field, converts it into a number and subtracts the length of the subfield FIXED_PART of the header (see ISO/IEC 8073 13.2).

Example:

VP_LENGTH(hd) = 0 if the header contains no variable part.

NOTE: The user data of the TPDU (if existing) are not contained in the variable part.

Test suite operation definition

Operation name	:	is_element_in_seqof (list: seqoftype ; el: eltype)
Result type	:	BOOLEAN
Comments	:	returns TRUE if el exists in list; FALSE otherwise

Description

```
is_element_in_seqof (list: seqoftype ; el: eltype)
BEGIN
        IF (list == {}) /* {} stands for empty sequence of */
                THEN return(FALSE)
        IF (el == head(list))
                THEN return(TRUE)
                ELSE return(is_element_in_seqof(tail(list), el))
END
```

Examples:

```
        head({A})       = A
        head({A,B})     = A
        tail({A})       = {}
        tail({A,B})     = {B}

        is_element_in_seqof({} , C)      =      FALSE
        is_element_in_seqof({A} , A)     =      TRUE
        is_element_in_seqof({A} , C)     =      FALSE
        is_element_in_seqof({A,B} , A)   =      TRUE
        is_element_in_seqof({A,B} , B)   =      TRUE
        is_element_in_seqof({A,B} , C)   =      FALSE
```

9 Guidelines on aliases

TTCN is provided in the graphical format mainly to support an easy understanding of the test descriptions, particularly the dynamic behaviour descriptions, by human readers. When a service description is defined for a protocol to be tested in an ATS, the SEND and RECEIVE events in the behaviour description will be associated to service primitives rather than PDUs, which are the more important aspect in protocol testing. When, additionally, static chaining is used in the constraint of such a statement line, the reader will not have a good understanding of that statement without checking the constraint declaration.

The concept of aliases means essentially that ASPs (and PDUs, if appropriate) may be renamed, such that the main aspect of an ASP (or a PDU) in a behaviour line is apparent from the new name, i.e. the alias. In many cases the "main aspect" is an embedded PDU the type of which is reflected by the alias name.

NOTE: An ASP (or PDU) may have more than one alias, since normally PDUs of several types can be embedded in a single ASP (or PDU).

EXAMPLE 22: Declaration and usage of aliases.

Alias definitions		
Alias name	**Expansion**	**Comments**
I_IAM	MTP_TRANSFER_INDICATION	MTP_TRANSFER_INDICATION service primitive used to carry an ISUP IAM message.
I_ACM	MTP_TRANSFER_REQUEST	MTP_TRANSFER_REQUEST service primitive used to carry an ISUP ACM message.
I_ANM	MTP_TRANSFER_REQUEST	MTP_TRANSFER_REQUEST service primitive used to carry an ISUP ANM message.
I_REL	MTP_TRANSFER_REQUEST	MTP_TRANSFER_REQUEST service primitive used to carry an ISUP REL message.
I_RELC	MTP_TRANSFER_INDICATION	MTP_TRANSFER_INDICATION service primitive used to carry an ISUP RELC message.

Usage of Aliases in the dynamic behaviour description:

Test step dynamic behaviour					
Test step name : ISUP_ORI_CALL_1					
Group : ISUP/ORI_CALL/					
Objective : To verify that a call can be successfully completed and cleared					
Default : AnyOtherEventExpected					
Comments : Set up successfully an ISUP connection and release it after verification of the connection					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		? I_IAM	MTRANSI(IAM_ SPEECH)		1)
2		! I_ACM	MTRANSR(ACM_ Basic)		2)
3		! I_ANM	MTRANSR(ANM_B asic)		3)
4		+CHECK_CONNECTIVITY			4)
5		! I_REL	MTRANSR(REL_ cause16)		5)
6		? I_RELC	MTRANSI(RELC_B asic)	PASS	6)
Detailed comments:					
1) Receive an MTP transfer indication ASP containing the IAM message with "speech" transfer capability.					
2) Send an MTP transfer request ASP containing the Basic ANS message.					
3) Send an MTP transfer request ASP containing the Basic ACM message.					
4) Use test step CHECK_CONNECTIVITY to verify that speech is possible.					
5) Send an MTP transfer request ASP containing the REL message with cause code 16.					
6) Receive an MTP transfer indication ASP containing the RELC message.					

10 Guidelines on constraint definitions

10.1 Introduction

Constraint definitions within TTCN form an important and typically voluminous part of an ATS. Due to the amount of constraints which are to be expected within an ATS, it is very important that this information is:

- readable; and
- structured.

Both aspects are very important to allow a developer of an executable test suite or a person maintaining the ATS to easily obtain the relevant information. More than for the other TTCN object types already dealt with in this ETR, the development of a general concept, in the sense of software engineering, is required for the constraints part. Some features allowed in constraint specification are "contradicting" or "opposing", like:

a) whether or not to use ASN.1;
b) use of base/modified constraints;
c) static/dynamic chaining;
d) parameterized/"constant" constraints.

Except in a), the conflict results to a large extent in more/less readable constraint references in the dynamic behaviour description on the one side, and more/less numerous and voluminous constraint declarations on the other side. This conflict cannot be solved generally, but it needs an individual strategic treatment in the development of an ATS, i.e. a design like a module in software development.

In the following subclauses, the individual aspects of a) to d) are treated by giving some rules and examples. In addition, some examples for specifying individual constraint values, especially wildcards, are given.

The general result of the discussion in this subclause is expressed in rule 14:

	RULE 14: General aspects of specifying constraints
a)	Develop a design concept for the complete constraints part, particularly with respect to the "conflicting" features as indicated in items i) to iv) and including naming conventions (see Clause 6).
b)	Make extensive use of the different optional "Comment" fields in the constraint declaration tables to highlight the peculiarity of each constraint.

NOTE: TTCN allows compact table formats for specifying constraints, particularly for non-ASN.1 constraint declarations. These compact tables normally allow to find a referenced constraint more quickly, but the editor has to provide many different table formats. The quick reference can also be achieved, when a cross reference document is created (see subclause 4.3). No preference is expressed in this ETR for a particular table format.

10.2 Using ASN.1 to specify constraints

ASN.1 as specified in ISO/IEC 8824 [11] is a dual concept of specifying types on the one side and values of these types on the other side. Therefore all arguments given in Clause 7 apply for constraints too, and no separate rules are given here.

10.3 Base constraints and modified constraints

This subclause applies to ASP/PDU/structured Type constraints independently of whether ASN.1 is used or not.

Modified constraints refer to a base constraint in their derivation path, so base constraints exist but modified constraints may not. The basic question is whether modified constraints should be defined in an ATS and what the relation between base constraints and modified constraints is, if the latter are used.

Modified constraints are particularly useful when the value of a single field or of a small number of fields are different with respect to a given constraint. When a base constraint focuses on the minimum requirements, the modified constraint focuses the view on the changed parameter/field and avoids redundant information. A disadvantage can occur when a base constraint has to be changed during development or maintenance and the modified constraints basing on this constraint are automatically changed, too. Side effects on test case behaviour and test case selection may occur in this case.

Modified constraints do not provide a new technical means to specify constraints values, but only a means for a shorter presentation.

NOTE: Some Conformance Testing Services (CTS) projects (e.g. CTS 4 Signalling System No.7 (SS7)) exclude the use of modified constraints in their ATS design document.

This discussion gives rise to rule 15:

RULE 15: Relation between base constraints and modified constraints

a) Define different base constraints for the send- and receive direction of a PDU (when applicable).

b) Use modified constraints preferably when only a small number of fields or parameter values are altered with respect to a given base.

NOTE 1: For SEND events the creation of a further modified constraint can sometimes be avoided, if an assignment is made in the SEND statement line, thus overwriting a particular constraint value.

c) Design the relation between base constraints and modified constraints always in connection with parameterization of constraints (see the two subsequent subclauses).

NOTE 2: Additional parameters in a constraint, introduced to avoid the declaration of further base/modified constraints can reduce the amount of constraints needed in an ATS, but then the constraint reference is getting more and more unreadable.

d) When modified constraints are used, keep the length of the derivation path small.
The length of the derivation path (resulting from the number of dots in it) is a kind of nesting level, and it is known from experience that a length greater than 2 is normally difficult to overview and maintain.

EXAMPLE 23: Definition of a base and modified PDU constraint.

PDU constraint declaration		
Constraint name : SCNbase (pcn: PDU)		
PDU type : SCN		
Derivation path :		
Comments : Base session connect constraint		
Field name	**Field value**	**Comments**
Connection_Identifier	-	
Connect_Accept_Item	Connect_Accept_Itembase	
Session_User_Requirements	FU_duplex_only	
Calling_Session_Selector	TSP_Clg_Sess_SEL	
Called_Session_Selector	TSP_cld_Sess_SEL	
Enclosure_Item	-	
User_Data	-	
Extended_User_Data	pcn	

Definition of a PDU modified constraint:

PDU constraint declaration		
Constraint name : SCN001 (pcn: PDU)		
PDU type : SCN		
Derivation path : SCNbase.		
Comments : Derived session connect constraint for use of session sync minor FU		
Field name	**Field value**	**Comments**
Session_User_Requirements	FU_duplex_minsync	

10.4 Chaining of constraints

As already mentioned in subclause 7.1, the meta-type PDU defined in TTCN has no counterpart in ASN.1. Disregarding this small difference, this subclause is independent of whether or not ASN.1 is used.

ASPs and PDUs may refer to other PDUs as their parameter or field types, which in turn may refer to other PDUs and so on. This is the principle of chaining. Theoretically the length of such a chain is not restricted in TTCN, but practically it is restricted by the protocol and PCOs employed.

In this subclause the typical situation is considered, where a particular ASP type contains a parameter (often called "user data") of type **PDU.** This means that the ASP can carry different PDU types. The conclusions in this subclause are also valid for longer chains or when **PDU** is replaced by a specific PDU type.

The target point here is the situation, when the ASP-PDU chain is getting specified a value, particularly a value for the parameter of type **PDU**. ISO/IEC 9646-3 [3] shows the 2 possibilities in subclause 11.4:

1) the value given to the parameter of type **PDU** is the name of a particular PDU constraint.
This is called **static chaining**, because the value for the embedded PDU is given statically in the constraint declaration, and is the same for each constraint reference in the dynamic behaviour description.

    NOTE: The embedded PDU can still be parameterized with respect to one or more of its subfields, yielding a "dynamic" effect for these fields at least.

2) the value given to the parameter of type **PDU** is the name of a formal parameter of type **PDU**, declared in the ASP constraint declaration.
This is called **dynamic chaining**, because the embedded PDU is given a value, when a PDU constraint is passed to the ASP constraint as actual parameter in the dynamic behaviour description.

Basically one can say that dynamic chaining is the parameterization of ASPs or PDUs with respect to embedded PDUs, and all the advantages and disadvantages of parameterized constraints apply (see also subsequent subclause). Dynamic chaining can reduce the number of constraint declarations considerably, but as additional parameters are introduced, the constraint reference can get less readable.

Rule 16 gives some guidance how to use of static and dynamic chaining, followed by an example.

	RULE 16: Static and dynamic chaining
a)	Make a careful evaluation of which embedded PDUs are needed in ASPs/PDUs, in which (profile) environment the ATS may operate and which kind of parameterization for other parameters/fields is needed, to find an appropriate balance between the use of static and/or dynamic chaining in a particular ATS.
b)	When the ATS is used in different profile environments and the types and values of embedded PDUs cannot be predicted, dynamic chaining is normally the better choice.
c)	When static chaining is used, chose the name of the ASP/PDU constraint such that it reflects the peculiar value of the embedded PDU (see also the clause on naming conventions).

EXAMPLE 24: Static chaining of constraints.

ASP constraint declaration		
Constraint name :	n_datreq_cr	
ASP type :	N_UNITDATA_REQ	
Derivation path :		
Comments :	(Connectionless) network service data request containing a CR TPDU constraint as constraint value of the NS_USERDATA parameter.	
Parameter name	**Parameter type**	**Comments**
NS_SRC_ADDR	LOCAL_NET_ADDR(LOCAL_NET_ADDR_LEN)	Network address of tester
NS_DEST_ADDR	REMOTE_NET_ADDR(REMOTE_NET_ADDR_LEN)	Network address of IUT
NS_USERDATA	CR_8073	Base constraint for the CR TPDU

EXAMPLE 25: Dynamic chaining of constraints.

ASP constraint declaration		
Constraint name :	n_datreq(tpdu: PDU)	
ASP type :	N_UNITDATA_REQ	
Derivation path :		
Comments :	(Connectionless) Network Service Data Request containing an TPDU passed by the formal parameter tpdu.	
Parameter name	**Parameter type**	**Comments**
NS_SRC_ADDR	LOCAL_NET_ADDR(LOCAL_NET_ADDR_LEN)	Network address of tester
NS_DEST_ADDR	REMOTE_NET_ADDR(REMOTE_NET_ADDR_LEN)	Network address of IUT
NS_USERDATA	tpdu	Formal parameter containing TPDU

10.5 Parameterization of constraints

Parameterization of constraints in the sense of subclause 11.3 in ISO/IEC 9646-3 [3] means the inclusion of formal parameters in the constraint declaration, which get concrete values by passing actual parameters in the constraint reference in the dynamic behaviour description (the dynamic chaining is a kind of special case for parameterization).

NOTE: The use of test suite parameters in the "value" column of a constraint declaration is not understood as "parameterization" of the constraint, as it is seen as "parameterization of the ATS" (see Clause 5).

Some constraints fields (e.g. sequence numbers) need to be parameterized, because their value is determined dynamically. Apart from these "necessary" parameterizations, the typical antagonism that has to be considered is, that the total number of declared constraints can be reduced drastically by using parameterization, but that a big number of parameters can make the constraint reference unreadable.

In some ATSs, an "indirect parameterization" is used: some fields are given the value "?" or "*", and an assignment or a Boolean expression containing the reference to this field is applied in the behaviour description. This shifts the parameterization away from the constraints reference. Systematic use of this possibility is made e.g. in the X.25 data link layer ATS ISO/IEC 8882-2 [9] (see also example 28).

Rule 17 gives some hints to the problem of parameterization, followed by several examples.

RULE 17: Parameterization of constraints	
a)	Make a careful overall evaluation of which field/parameter values are needed in ASPs and PDUs to find an appropriate balance between the aim of a comparably small number of constraint declarations and readable and understandable constraint references.
b)	Keep the number of formal parameters small. Keep in mind, that the number of formal parameters in structured/ASN.1 types Constraints will add up to the total number of ASP/PDU constraints. A clear border for the number of formal parameters cannot be stated, but it is known from experience that a number bigger than 5 normally cannot be handled very well.

EXAMPLE 26: Parameterized constraint (tabular format).

PDU constraint declaration		
Constraint name	:	RL3(cref, cval:BITSTRING)
PDU type	:	REL
Derivation path	:	
Comments	:	RELEASE message constraint with call reference and cause value passed as parameters

Field name	**Field value**	**Comments**
PD	'00001000'B	Protocol discriminator
CR	cref	Call reference parameterized by cref
MT	'01001101'B	Message type RELEASE
CAU	CAU1(CVAL)	Cause information element (structured type) parameterized by cval = cause value
FAC	-	Facility absent
DSP	-	Display absent
UUI	-	User-user information absent

EXAMPLE 27: Parameterized constraint (ASN.1 format).

ASN.1 PDU constraint declaration		
Constraint name	:	INT_L1(lcif:INTEGER)
PDU type	:	Interrupt
Derivation path	:	
Comments	:	Long interrupt CCITT 1980

Constraint value

```
{
        header  HDR_G(lcif)
        --      header parameterized with current logical channel

        user_data       UD_INTERR
        --      user_data given by test suite parameter
}
```

EXAMPLE 28: Different kinds of constraint parameterization.

This example is basically taken from the X.25 Data Link Layer ATS ISO/IEC 8882-2 [9]. The Receive Ready (RR) PDU is declared in the following way:

ASN.1 PDU type definition		
PDU name	:	RR
PCO type	:	
Comments	:	Receive ready
Type definition		

```
SEQUENCE {
        address OCTET STRING(SIZE(1))          -- address field
        nr      BIT STRINGT(SIZE(3))           -- sequence number field
        pf      BIT STRINGT(SIZE(1))           -- poll/final Bit
        control BIT STRINGT(SIZE(4))           -- control field identifying the PDU
}
```

A constraint for the PDU is defined as follows:

ASN.1 PDU constraint declaration		
Constraint name	:	RR_31
PDU type	:	RR
Derivation path	:	
Comments	:	Send constraint for RR command with P=1
Constraint value		

```
{
        address ADDR_A         -- TS constant, value = '03'O
        nr          -          -- sequence number is set using assignments
        pf          P1         -- TS constant, value = '1'B
        control     '0001'B    -- fixed value
}
```

Detailed comments:
In the original X.25 ATS the field value of the nr field is "?" which appears to be a bad style for a send constraint, because it is a potential candidate for a test case error if the "?" is not replaced by a specific value before the constraint is sent.

The address field can have two possible values: ADDR_a and ADDR_B. The pf field also can have two possible values: P0 and P1. The nr field will contain a sequence number, dynamically determined from the Test Case Variable VR. Three further constraints have been defined to cope for the possible combinations (only used for sending). The use of the PDU/constraint in some dynamic behaviour is like the following:

No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
n		L!RR (RR.nr:= VR)	RR_31		

If parameterization of all of the 3 variable fields was used, only one constraint could replace the four constraints mentioned above:

ASN.1 PDU constraint declaration		
Constraint name	:	RR_1(addr:OCTET STRING; n_r, p_f: BIT STRING)
PDU type	:	RR
Derivation path	:	
Comments	:	
Constraint value		

```
{
        addressaddr
        nr              n_r
        pf              p_f
        control         '0001'B -- fixed value
}
```

and the use of this PDU/constraint would be:

No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
n		L!RR	RR_1(ADDR_A,VR,P1)		

10.6 Constraint values and matching mechanisms

While the previous subclauses treated the building up and structuring of constraints, this subclause focuses on values given to individual fields or parameters inside constraints.

The constraint values used in a concrete ATS are dictated by the particular circumstances, and no general rule can be given whether a field is absent or not, etc. Rule 18, therefore, concentrates on the optional elements of the constraint value specification:

RULE 18: Constraint values	
a)	Use comments to highlight the peculiarity of the value, especially when the value is a literal, whose meaning is not apparent.
b)	Use test suite constants instead of literals, when appropriate. Normally not all literals can be defined as Test Suite Constants, but a rule by thumb is: if a literal value of a given type occurs more than once (as a constraint value or more generally in an expression), then it is useful to define it as a Test Suite Constant, letting the name reflect the value.
c)	Use the length attribute when possible and when the length is not implicit in the value itself or given by the type definition (e.g. for strings containing "*").

Values in constraints for the SEND event need to evaluate to a specific value (see subclause 11.5 of ISO/IEC 9646-3 [3]). Specific values are expressions containing:

- literal values;
- test suite constants;
- test suite parameters;
- formal parameters; and
- test suite operations.

For values in RECEIVE events additional matching mechanisms are defined (see table 5, in subclause 11.6.2 of ISO/IEC 9646-3 [3]). The following list of examples is focusing on these additional matching mechanisms. They are used:

- instead of values;
- inside values; and
- with attributes.

The examples as a whole are not intended to form consistent constraints, though only two tables are used, one for the ASN.1 format and one for the tabular format. PDU constraints are chosen, but the individual values/matching mechanisms are also applicable to ASP constraints and structured/ASN.1 types Constraints respectively. Each example is commented inside the table, including the indication of the related field/parameter type.

EXAMPLE 29: Constraint values and matching mechanisms using the tabular format.

PDU constraint declaration		
Constraint name :		
PDU type :		
Derivation path :		
Comments :		
Field name	**Field value**	**Comments**
lc_id	COMPLEMENT(LCN_ACTIVE)	Type: INTEGER. All logical channel numbers except that of the currently active logical channel (LCN_ACTIVE) are accepted.
User_Data	-	Type OCTETSTRING. The user data field is expected to be absent.
d_bit	?	Type: BITSTRING[1]. The D-bit shall be present as part of the general format identifier, but its value may be any value ('0'B or '1'B in this case).
CallingAddr	CLG_IUT IF_PRESENT	Type: callingAddr. The "calling address" information element may be present or not. If it is present, its contents need to be the calling address of the IUT.
S22_VAL	'*'O[TMP_S22_LEN]	The TMP parameter S22_VAL may contain any octets, but the length should be equal to the value of test suite parameter TMP_S22_LEN.
cause_val	(NORMAL_CLR,CALL_REJ, NORMAL_UNSPEC)	Type: BITSTRING. Each of the 3 cause values given by the referenced test suite constants is accepted.
lc_id	(LTC .. HTC)	Type: INTEGER. All logical channel numbers in the range between test suite parameter LTC and test suite parameter HTC are accepted.
control_field	'001?1111'B	Type: BITSTRING. The P/F bit (position 4) in the control field may be set to 0 or 1.
ud	'02*'O[1 .. 128]	The user data field is expected to contain the value '02'O (subsequent protocol identifier = "syntax-based videotex") in the first octet, followed by any other octets up to a maximum total length of 128 octets.
LI	INT_TO_HEX(LENGTH_OF (VARIABLE_PART)+LENGTH_OF (FIXED_PART) + 1, 2)	TYPE of LI: HEXSTRING. The formal parameters VARIABLE_PART and FIXED_PART are of type OCTETSTRING. The contents of the length indicator field LI is calculated from the values of the formal parameters.

EXAMPLE 30: Constraint values and matching mechanisms using the ASN.1 format.

Example 29 also applies to the ASN.1 format. Therefore, only features specific to ASN.1 are presented in this example.

ASN.1 PDU constraint declaration
Constraint name :   **PDU type** :   **Derivation path** :   **Comments** :
Constraint value

{
 facilities FAC_FSRC IF_PRESENT -- Fast Select Reverse Charging)
 -- Type: Facility_Field (SET). If the facility field is present in the PDU, it should have the
 -- indicated value
 }

or

Constraint value

{
 facilities SUPERSET(FAC_AE,FAC_BCUG)
 -- Type: Facility_Field (SET). At least the facilities "Address Extension" and
 -- "Basic Closed User Group" are expected in the facilities field.
 }

or

Constraint value

{
 facilities SUBSET(FAC_AE,FAC_BCUG)
 -- Type: Facility_Field (SET). No facility or one or both of the indicated facilities are expected.
 -- No other facility than "address extension" and "basic closed user group" may be present in
 -- the facilities field.
 }

11 Guidelines on test cases

11.1 Introduction

As test cases are defined to specify all sequences of foreseen test events necessary in order to achieve the test purpose, they form the main part of an ATS. A test case is structured into three parts:

- preamble,
  the sequence of test events to put (if necessary) the IUT into the desired stable testing state in which the test body starts;
- test body,
  the sequence of test events (starting in a stable testing state) for achieving the test purpose;
- postamble,
  the sequence of test events to put the IUT into the desired stable testing state if the test body ends without being there [4].

Different test cases may use the very same partial sequence of test events. Such partial sequences of test events may be defined within named subdivisions called test steps. A test step is similar to a procedure or subroutines in higher programming languages. The test step concept enables test cases being modularized. In the extreme case a test case may only consist of a sequence of test step names together with some control flow information, built from conditions on test body and state variables and finally the assignment of a verdict.

---

[4] In the most standardized protocols the postamble will simply contain a abort/disconnect event to return to the idle state of the protocol machine.

This clause will therefore focus on the following aspects:

- assignment of verdicts;
- use of a test body entry marker;
- use of a state variable;
- use of test steps,

whilst the test step specific aspects are handled in a separate Clause.

11.2 Assignment of verdicts

The objective of a test case is to achieve a given test purpose. Therefore, after having executed a test case there should be a statement of successful/unsuccessful completion of the test purpose, i.e. a final verdict has to be assigned within the test case. The sequence of test events achieving the test purpose builds the test body, i.e. the final verdict should not be assigned before the end of the test body (otherwise the test case will be terminated without achieving the test purpose). From this point of view a final verdict should not be assigned within the preamble. On the other side the final verdict may depend upon some events included in the test postamble, but should not depend only upon test events within the test postamble (in this case the sequence of test events achieving the test purpose is totally included in the test postamble, in contradiction to ISO/IEC 9646-3 [3]). In such a situation a final verdict cannot be assigned within the test body, but a preliminary verdict should be assigned.

Preliminary results are recorded in the conformance log. Thus they may be used as support for analyzing the test results or as diagnostic support for test operators during test campaigns identifying that a certain point has already been achieved. For this it may be helpful to assign the preliminary result "(PASS)" after a sequence of specific events within the test case. Thus by analyzing the conformance log of a "non-passing" test case it can be seen which sub-sequence of test events was responsible for changing the original preliminary result "(PASS)". To see if the original preliminary result was still valid when entering the test body the entry point of the test body should be recorded in the conformance log.

Care should be taken by setting preliminary results. If the sequence of events between two succeeding preliminary results contains a large amount of events, then the preliminary results recorded in the conformance log may be of minor help for analyzing the test result. On the other side, if for each event a preliminary result is assigned, the big amount of recorded preliminary results may make the analysis of the conformance log more difficult and does not help.

EXAMPLE 31: Setting of preliminary results within test cases.

Test case dynamic behaviour					
Test case name: P6_306					
Group : PACKET/P3/Inopportune/					
Purpose : Verify the IUT clears an Interrupt packet received in state P3					
Default :					
Comments : Test case for the X.25 packet level protocol					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		+P1_PREAMBLE(TVC_OR_IVC)			1
2		!CALL START TD_RESP	CALL_0(LCI)		
3		?TIMEOUT TD_RESP			
4	Body	!INT START TD_RESP	INT_0(LCI)	(P)	2
5		+P3_INOPP			3
6		+R1_POSTAMBLE			4
		...			
Detailed comments: 1) put the IUT in state P1; 2) send interrupt packet; 3) handle the response to inopportune packets sent from the IUT while in state P3; 4) verify the IUT is in state R1 for VCs and D1 for PVCs.					

In example 31 the setting of the preliminary verdict "(PASS)" is associated with the end of the preamble and the beginning of the test body.

When assigning verdicts special attention should be given to the fact that the default tree of a test case will have to cover all abnormal situation as unexpected alternative events and timeouts. Such events may result in different verdicts depending on the fact if the test body has already been reached or not when the default tree is entered (see also later clause about default trees).

Example 32 illustrates the combination of events and verdicts in relation to the test body. Special attention should be given to the column entitled Default which represents default behaviour on unsuccessful events. According to the position of the corresponding successful events in either the preamble or the test body the preliminary result "(INCONC)" or "(FAIL)" is assigned in the default part.

EXAMPLE 32: Relation between verdict assignment and test body.

Test Purpose	**Preamble**	**Verdict**	**Test Body**	**Verdict**	**Default**	**Verdict**
Accept FTAM Connection	(empty)		F-INIT(+)	PASS	F-INIT(-) TIMEOUT	(FAIL) (INCONC)

Select File	F-INIT(+)	(PASS)	F-SELECT(+)	PASS	F-INIT(-)
					F-SELECT(-)
					TIMEOUT
						(INCONC)
						(FAIL)
						(INCONC)

Open File	F-INIT(+)
	F-SELECT(+)
		(PASS)
		(PASS)
			F-OPEN(+)	PASS	F-INIT(-)
					F-SELECT(-)
					F_OPEN(-)
					TIMEOUT
						(INCONC)
						(INCONC)
						(FAIL)
						(INCONC)

(+)	corresponds to the successful event.
(-)	corresponds to the unsuccessful event.

Example 32 shows that the test body of the first test purpose forms the preamble of the second test purpose and that the test body and preamble of the second test purpose builds the preamble for the third test purpose. Furthermore it can be seen that the verdict assigned to a certain event depends on the position of the event within the test case (here preamble/test body/default). This can be seen, e.g., in the second test case assigning a preliminary result "(FAIL)" in case of a F-SELECT(-) (failure in test body) whereas the third test case will assign a preliminary result "(INCONC)" on event F-SELECT(-) (unsuccessful select in preamble).

RULE 19: Verdict assignment in relation to the test body
Make sure that verdict assignment within a default tree is in relation to the test body. If an unsuccessful event arising in the test body is handled by the default tree, then assign a preliminary result "(FAIL)" within the corresponding behaviour line of the default tree. If the position of the unsuccessful event is not in the test body, assign a preliminary result "(INCONCLUSIVE)". If the behaviour line handling the unsuccessful event is a leaf of the default tree, assign a final verdict instead.

11.3 Test body marker

As already mentioned, verdict assignment within a default tree may be dependent on the fact if the test body has been reached before the default tree is entered. Therefore, information should be transferred to the default tree whether the test body has been entered or not. For transferring this information to the default tree a label marking the test body entry (as recommended by Annex E of ISO/IEC 9646-3 [3]) cannot be used.

A very convenient way to make default trees know whether the test body has been entered or not is using a test body variable. This may be a Boolean test case variable being set to TRUE when the test body is entered and having the value FALSE otherwise.

It is the decision of the test suite developer on which way to mark the entry of the test body. This may be via a test suite/case variable, via a label or by simply adding special comments.

RULE 20: Test body entry marker
The entry of the test body should be marked.

11.4 Use of a variable reflecting the protocol machine state

It might be very helpful to use a variable reflecting the current state of the Protocol Machine (PM). Such a variable can be used in order to decide which specific action may be chosen next. There are protocols which allow optional behaviour states. Behaviour states can be easily treated by using a variable within TTCN. In addition, at the end of the test this mechanism can be used to decide which postamble needs to be used in order to reach the testing idle state.

RULE 21: State variable
For realizing test purposes dependent on protocol states, use a variable to reflect the current state of the IUT.

A test suite variable instead of a test case variable being used for reflecting the current state of a state-oriented protocol machine could provide an additional control flow mechanism in the following way:

- each time a test case is left, a variable is set to a value reflecting the assumed state when leaving the test case;

- each time a test case is entered, the variable will be checked to verify whether a starting state for this test case is given.

By using this mechanism the consistency of the test case initializing sequence can be assured.

11.5 Use of test steps

Test steps are seen to be more and more important in the development of ATSs as they ease the way of writing ATCs by just attaching a certain already defined test step. Besides easing the writing of ATCs the readability is also increasing, as long as the test step names are created following certain understandable naming conventions.

As already mentioned in the introduction of this Clause, a test step comprises a sequence of test events which may be used by different test cases. Thus, it is obvious to use test steps wherever the same sequence of test events is used by different test cases. By doing this, the readability of the test cases is improved. For improving the structure and the readability of test cases it may even be useful to define a test step for a sequence of test events which is used only once within a single test case, e.g. by expressing preambles and postambles via test steps, or in state oriented protocols when test purposes "... check that the IUT has reached state X" are defined. For realizing such test purposes it makes sense to combine all test events being used for checking if state X has been reached within a test step and to attach this test step within the respective test cases instead of repeating the corresponding event sequence.

RULE 22: State checking event sequences
Combine event sequences used for checking a state of the IUT within test steps.

12 Guidelines on test steps

12.1 Introduction

Similar to subroutines in programming languages, conventions should be stated to allow easy combination of test steps and to guarantee that no errors are introduced by combining test steps.

This Clause will try to state guidelines how to write test steps. Major aspects which are dealt with are:

- multiple usage/construction of test step libraries;
- level of complexity/nesting level;
- assignment of verdicts;
- returning values from test steps;
- exit from test steps;
- parameterization of test steps;
- restrictions on behaviour description.

12.2 Construction of test step libraries

During the development of ATSs it became more and more important that available test steps were written in such a way that they might be used in multiple instances (re-use of test steps), i.e.:

- called more then once;
- used as preamble/postamble (or part of it) in ATCs;
- used in other ATSs, resulting in the re-usage of a test step library in other ATSs.

In order to allow such usage, general rules applying to all test steps which should form such a test step library should be given. These rules apply to the following aspects:

- treatment of verdict assignment;
- easy adaptation to test case needs;
- straightforward, no error case treatment.

The "treatment of verdict assignment" aspect is very important. For details see later subclause.

For "easy adaptation" of test steps to the needs of a certain instance it is essential that constraints used are parameterized in a suitable form. Other relevant aspects, e.g. the repetition argument within an existing loop, should be considered as possible candidates for test step parameterization to reach easy adaptation.

RULE 23: Easy adaptation of test steps to test cases
For easy adaptation of a test step to test case needs, parameterize the constraints used within a test step.

"Straightforward, no error case treatment" supports the sequencing of several test steps. Consider the fact that if a test step treats more than one alternative, events resulting in different states could increase the complexity and therefore the actions/alternatives of the following test steps. If it is of no need for the test to continue after receipt of a specific event (e.g. if the test step objective was to establish the connection, the failure of the connection establishment is not of interest) the event should be treated in the default tree resulting in INCONCLUSIVE or FAIL depending on the fact if the test body was reached or not (see subclause above).

Example 33 illustrates a test step which might be used in various instances. The way it is written allows the combination with other similar test steps as it covers parameterization as well as the treatment of only relevant events.

EXAMPLE 33: Reusable test step - resconnect.

Test step dynamic behaviour					
Test step name : Resconnect (a_ascreq: A_ASCreq; a_asccnf: A_ASCcnf)					
Group : FTAM/Step-LIB/					
Objective : Establish an FTAM association					
Default : DEF(L)					
Comments :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		L!A_ASCreq	a_ascreq		1
2		START A			2
3		L?A_ASCcnf	a_asccnf		3
4		CANCEL A			4
5		(F_STATE:=CON)			5
Detailed comments: 1) send constraint to establish connection; 2) start a timer in order to treat the situation where IUT does not respond; 3) await confirmation that connection is established; 4) cancel timer after receiving expected event; 5) set state variable to corresponding state.					

The following aspects allow the above test step to be used in a test step library:

- parameterized constraints,
  variants are possible;
- timer is started,
  timeout will occur if IUT does not respond;
- straightforward,
  no other events except those related to the test step objective are treated;
- setting of state variable,
  may allow certain action in later behaviour (e.g. in default to reach idle state to finish test);
- no setting of any result.

But, the events not covered:

- TIMEOUT;
- alternative events (connection not established, aborts, etc.),

need to be treated in the default tree (see subclause 13.3).

In contrast to example 33 the following test step could not be used for building such a test step library.

EXAMPLE 34: Non-reusable test step - resconnect.

Test step dynamic behaviour					
Test step name : Resconnect					
Group : FTAM/Step-LIB/					
Objective : Establish an FTAM association					
Default : DEF(L)					
Comments :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		L!A_ASCreq	A_ASCreqbas (PCP001 (FINIRQ005))		1
2		L?A_ASCcnf	A_ASCcnf501 (PCPA501 (FINIRP505))	(P)	2
3		L?A_ASCcnf	A_ASCcnf501 (PCPRbase (FINIRPbase))	(F)	3
Detailed comments:					
1) send constraint to establish connection;					
2) await confirmation that connection is established;					
3) await negative confirmation that connection is not established.					

The following aspects are not covered in example 34:

- constraints are not parameterized via test step parameters,
  no different instances, variations not possible;
- the calling test case itself will not have any knowledge about the alternative chosen (line 2 or 3):
  - no state variable set;
  - not straight forward;
- no timer is started,
  test will never stop if IUT does not respond;
- setting of results:
  - unclear result, the calling test case may need analysis of result to determine next action;
  - test step can not be used as preamble as preliminary result may be set to FAIL (should only be set to INCONCLUSIVE or PASS).

12.3 Level of complexity / nesting level

In order to ensure that test steps are readable and easy to understand, complexity of test steps should be kept at a level as low as practical. Structuring aspects to reach low complexity could be:

- atomic confirmed service primitives based on one send and one receive action:
  - sequence of a service request and a service indication primitive;
  - sequence of a service response and a service confirmation primitive;
- test event sequences building a "logical" unit:
  - loops of test events;
  - sequence of test events for checking a particular state of the IUT;
  - etc.

RULE 24: Minimizing complexity of test steps
Minimize the complexity of test steps either by restricting the objective of a test step to atomic confirmed service primitives or by separating event sequences which build different "logical" units into different test steps.

Besides minimizing the complexity of test steps the level of nesting of test steps should be kept small. Experience has shown that a nesting level greater than 3 results in dependencies and complexities which should be avoided [5].

Only some circumstances (i.e. the use of a test management protocol or the substitution of a multiply used complex instruction) may justify the use of a greater nesting level.

RULE 25: Nesting level of test steps
Keep the nesting level of test steps to a minimum.

Restricting the nesting level of test steps to a certain level has the additional advantage that recursive tree attachments are not possible. Recursive tree attachment are allowed by TTCN. The use of recursive tree attachments result in expanded behaviour trees with one infinite sequence of events. On the other hand, the execution of a test case should be finite in each case. Therefore, special care has to be taken that a test case including recursive tree attachment results in finite test execution by using counters or Boolean expressions. But using recursive tree attachments in conjunction with counters or Boolean expressions makes a test step much more complex. Therefore, the use of recursive tree attachments should be avoided. The same test behaviour can be reached by defining loops which are much easier to control.

RULE 26: Recursive tree attachment
Avoid recursive tree attachment. Where possible, use loops instead of recursive tree attachments.

Instead of using a test step of high complexity within a test case it would be more convenient to use a corresponding sequence of test steps reflecting specific atomic behaviour or building "logical" units. In examples 35 and 36, an example for both correct and incorrect structuring of test steps with respect to the objective "specific atomic behaviour" is given.

[5] Within the FTAM responder ATSs developed in CTS2 a nesting level of 1 was kept. The only test step used within others was the one for sending an ABORT FPDU.

EXAMPLE 35: Well structured test steps.

```
Test case: SAMPLE_I
+STEP_A
 +STEP_B
  +STEP_C
```

```
Test step: STEP_A
!Send_AA
 ?Receive_AB
```

```
Test step: STEP_B
!Send_BA
 ?Receive_BB
```

```
Test step: STEP_C
!Send_CA
 ?Receive_CB
```

The structure chosen:

- is based on test steps structured according the atomic behaviour aspect;
- allows easy production of other test cases based on these test steps;
- is easy to read and to understand as the level of complexity is kept low.

EXAMPLE 36: Not well structured test steps.

```
Test case: SAMPLE_I
+STEP_a
```

```
Test step: STEP_a
!Send_AA
 ?Receive_AB
  +STEP_b
```

```
Test step: STEP_b
!Send_BA
 ?Receive_BB
  !Send_CA
   ?Receive_CB
```

The test steps in example 36 are not well structured in the following sense:

- structuring is not sequential,
  test step STEP_a calls STEP_b;
- complexity is not atomic,
  test step STEP_b is too complex;
- structuring does not allow construction of other test cases using the first part of test step STEP_b.

12.4 Assignment of verdicts

As already stated in previous subclauses, it is very important that the assignment of verdicts within test steps is treated very carefully in order to allow multiple usage and construction of test step libraries.

It is obvious that no final verdict should be set within a test step as this would automatically terminate the ATC and would contradict the possibility of combining test steps.

If used, the setting of preliminary results should be treated very carefully in order to allow an easy combination of test steps. Multiple settings of preliminary results, e.g. in a first step to INCONCLUSIVE and then to PASS would be more confusing than helping, as the second assignment does not affect the result variable (see table 6 in ISO/IEC 9646-3 [3]).

Setting of a preliminary FAIL result would contradict the re-usage of this test step at a different position, e.g. it could not be used as part of a preamble for another test case (this could falsify the test report as a FAIL verdict was assigned without reaching the test body).

Setting of preliminary PASS results within the test step can be useful for analyzing the conformance log. Therefore, if it is the intention of the test suite specifier to include preliminary results in the test suite for supporting the analysis of the conformance log, at least one preliminary PASS result should be set at the leaf of the "passing" event sequence of the test step.

When specifying test steps without any assignments of verdicts, the problems concerning multiple usage, falsified test reports, etc. can be avoided. But there may be situations in which the assignment of verdicts within a test step is of advantage:

- improvement of readability;
- support for test operator;
- additional information for supporting the analysis of conformance test reports.

There may also be situations where it is meaningful to combine a sequence of test events into a test step which mainly contribute to the building of the final verdict. For instance, by realizing test purposes **"... and check that IUT has reached state X"**, it may be reasonable to combine the event sequence for checking if state X has been reached in a test step. For such test steps it cannot be avoided that at least a preliminary FAIL is set. In this case the test suite specifier should take care not to use the respective test step within a preamble. Example 37 shows such a test step.

EXAMPLE 37: Test step for verifying the IUT has reached a given state.

Test step dynamic behaviour					
Test step name	: P6_POSTAMBLE				
Group	: PACKET/P6_POSTAMBLE/				
Objective	: Verify the IUT is in state P6 after the test body has successfully completed				
Default	:				
Comments	: Sample from the ATD for the X.25 Packet Level Protocol				
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		!CLEARC START TD	CLRC_0(LCI)		
2		REPEAT LTS_TIMEOUT_TD UNTIL [FLAG]			1)
		LTS_TIMEOUT_TD			
3		?TIMEOUT TD (FLAG:= TRUE)			2)
4		?CLEAR	CLR_1(LCI)		3)
5		?OTHERWISE CANCEL (FLAG:= TRUE)		(F)	4)

Detailed comments:
1) wait some time and check the events; test case variable FLAG initialized to FALSE;
2) stop waiting;
3) CLEAR is accepted (due to the possible expiry of the T21 timer in the IUT);
4) other events are not accepted.

RULE 27: Verdict assignment within test steps
If verdicts are assigned within a test step, guarantee at least the partial (i.e. not general) re-use of the test step.

12.5 Returning values from test steps

The semantic given to test steps does not consider the return of any value from test steps. Nevertheless it may be necessary for a test step to pass some information to the calling test case (e.g. about a session point synchronization number, or the number of data blocks read) by setting a test suite/case variable. Such conventions should be stated in detail within the test step.

EXAMPLE 38: Test step returning values.

Test step dynamic behaviour					
Test step name : UTIL_IUT_tpdu_size					
Group : Utilities/					
Objective : TMP_Set up a connection from IUT to LT, negotiate TPDU size and send one TPDU with the negotiated size. The test suite variables LOCAL_TPDU_NR and REMOTE_TPDU_NR will be incremented according to the number of sent and received TPDUs					
Default : DefBody					
Comments : Simplified sample from ATS for transport class 4 showing usage of return values					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		+PRE_IUT_con_vsize			1
2		+UTIL_IUT_data_vsize			2
3		(LOCAL_TPDU_NR:= LOCAL_TPDU_NR+1)			3
		.....			
Detailed comments: 1) establish a transport connection from the IUT to the LT and negotiate TPDU size; 2) send a TMP PDU to IUT, to let IUT send one TPDU with the negotiated TPDU size; 3) increment LOCAL_TPDU_NR by one.					

12.6 Exit from test steps

Real exits as found in programming languages cannot be produced within TTCN. Nevertheless, the cutting (i.e. no-execution) of certain alternative test events may be seen as such and should be avoided. Generally a test step should only result in a stable well defined state (indicated by a state variable). This allows the test case to perform the corresponding required action afterwards.

EXAMPLE 39: Test step resulting in different states - Iniope_rat_cha.

Test step dynamic behaviour		
Test step name	:	Iniope_rat_cha(foperq:FOPERQ; foperp:FOPERP; fratrq:FRATRQ; fratrp:FRATRP; fcharq:FCHARQ; fcharp:FCHARP)
Group	:	FTAM/Step-LIB/
Objective	:	Receive either F-OPEN-Request, F-READ-ATTRIBUTE-request or F-CHANGE-ATTRIBUTE-request in state SELECT; send corresponding response State is changed from SELECT to OPEN if F-OPEN is received and unchanged otherwise
Default	:	DEF(L)
Comments	:	Sample showing a test step resulting in different states

No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		START A			1
2		L?P_DTind	P_DTind510 (foperq)		2
3		L!P_DTreq	P_DTreq010 (foperp)		2.1
4		(F_STATE:=OPEN)			2.2
5		L?P_DTind	P_DTind511 (fratrq)		3
6		L!P_DTreq	P_DTreq011 (fratrp)		3.1
7		L?P_DTind	P_DTind512 (fcharq)		4
8		L!P_DTreq	P_DTreq012 (fcharp)		4.1

Detailed comments:
1) start inactivity timer;
2) receive F-OPEN-Request:
   2.1) send response FPDU;
   2.2) set variable to OPEN state;
3) receive F-READ-ATTRIBUTE-Request:
   3.1) send response FPDU;
4) receive F-CHANGE-ATTRIBUTE-Request:
   4.1) send response FPDU.

Example 39 shows a test step resulting in two different states:

- the new OPEN state (if expected OPEN FPDU was received);
- the previous (SELECT) state.

The test case should perform the corresponding required action depending on the value of the state variable.

12.7 Parameterization of test steps

Test steps may be parameterized. By selecting appropriate actual parameter values the test step can be adapted to the special needs of a test case.

In conjunction with parameterized constraints being used as actual parameter values of a test step (passing parameterized constraints) the parameterization of test steps is a very powerful mechanism. For testing protocols of the lower layers, many test purposes deal with testing the behaviour of the IUT on receipt of invalid PDUs being of the same PDU-type but differing in PDU field values. By defining a respective PDU constraint having parameters for each PDU field which may differ during the test campaign the constraint may be used as a formal parameter of a test step which takes some specific actions. The very same test step can be used in different test cases by simply adapting the actual parameters of the constraint to the needs of the test case. This is illustrated in example 40.

EXAMPLE 40: Using parameterized constraints as test step parameters.

PDU constraint declaration		
Constraint name	:	tpdu_cc(code:CODETYPE; checksum:CHECKSUMTYPE)
PDU type	:	TPDU_CC
Derivation path	:	
Comments	:	PDU constraint for checking reaction of IUT on receiving a CC PDU with an invalid value for field code or checksum

Field name	**Field value**	**Comments**
Li	'00001010'B	
Code	code	constraint parameter
Cdt	'0100'B	
Dst_ref	'ABCD'O	
Src_ref	'ABCE'O	
Class	'40'H	
Checksumcode	'11000011'B	
Checksumlength	'00000010'B	
Checksum	checksum	constraint parameter

Test step dynamic behaviour		
Test step name	:	TPDU_IB_PDUCC(ivpdu:TPDU_CC)
Group	:	TRANSPORT4/Step-LIB/
Objective	:	Send invalid CC PDU to IUT and check response
Default	:	DEF(L)
Comments	:	Sample showing usage of parameterized constraints as test step parameter

No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		L!TPDU_CC	ivpdu		1
2		L?TPDU_DR	tpdu_dr	(P)	2
		.....			

Detailed comments:
1) send invalid CC PDU;
2) receive DR from IUT.

Test case dynamic behaviour					
Test case name : T4_BE_IB_WFCC_5					
Group : TRANSPORT4/BE/IB/WFCC/					
Purpose : Verify that the IUT in state WFCC discards a PDU with an invalid TPDU code					
Default : DEF(L)					
Comments : Sample for attaching a test step with a parameterized constraint as an actual parameter					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		+ PRE_IUT_WFCC			1
2		+TPDU_IB_PDUCC(tpdu_cc('0100'B, CheckSum))			2
		.....			
Detailed comments: 1) put IUT into state WFCC; 2) attach test step TPDU_IB_PDUCC; the actual parameter CheckSum is a test suite variable with a valid checksum value; '0100'B is not a valid PDU code.					

RULE 28: Parameterized test steps
Use parameterized test steps to ensure re-use of test steps within test cases for different needs.

12.8 Restrictions on behaviour description

The series ISO/IEC 9646 [1] to [8] requires that test results achieved by using the same ATS for testing a specific IUT are comparable. From this requirement three further requirements can be derived:

- test execution shall be deterministic,
  during the test campaign the upper/lower tester should never have the choice between two send events;
- test results shall be independent from time,
  results achieved during a test campaign shall not depend on performance of the test system, on performance of the underlying service provider or on performance of the TMP/Test Coordination Procedure (TCP) between upper and lower tester;
- execution of a test case shall be finite,
  a test result shall be achieved after a finite number of events has occurred.

A further requirement not outlined by ISO/IEC 9646-3 [3] but also very important in specifying ATSs is that an event in an alternative sequence should be able to be reached.

TTCN does not take into account these requirements, i.e. the TTCN semantic allows the specification of ATSs which do not fulfil these requirements. In the following, some guidance is given on how to use TTCN to fulfil the outlined requirements.

12.8.1 Sequence of alternatives

Two classes of TTCN statements can be distinguished:

a) unconditional statements (UCSs):
- send events;
- assignments to variables;
- timer operations;
- GOTO;
- REPEAT;

b) conditional statements (CSs):
- receive events;
- timeout events;
- OTHERWISE event.

If an UCS is followed by an UCS in the same sequence of alternatives, then the second UCS is never reached as can be seen in example 41.

EXAMPLE 41: Alternative sequence including two UCSs.

Test case dynamic behaviour					
Test case name:		P9_206			
Group :		PACKET/P6/IMPROPER/P06/			
Purpose :		Verify the IUT discards too long incoming call packet (user data field of 17 or 129 octets) received in state P6			
Default :					
Comments :		Modified test case for the X.25 packet level protocol			
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
		.....			
n		!CALL	CALL_U129(LCI)		
		.....			
m		!CALL	CALL_U17(LCI)		1
		.....			
Detailed comments: 1) the send event in line m will never be executed.					

From this, rule 29 can be derived.

RULE 29: Combining statements in a sequence of alternatives
If there is no Boolean expression included in an alternative sequence, a statement of type UCS should never be followed by a statement of type UCS or CS within a sequence of alternatives.

Many problems concerning the reachability of alternatives are caused by using overlapping relational expressions as alternatives in an alternative sequence.

EXAMPLE 42: Alternative sequence 1 including overlapping relational expressions.

Test step dynamic behaviour					
Test step name : RelationalExpression1 **Group** : ..... **Objective** : ..... **Default** : ..... **Comments** :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		......			
		......			
n		[a>5]			1
		......			
m		[a<=5]			1
		......			
o		[a=5]			1
		......			
Detailed comments: 1) "a" may be a test suite variable or a test suite parameter of type INTEGER.					

The first two relational expressions cover the whole range of values for "a". Therefore, the third alternative of the alternative sequence can never be reached.

EXAMPLE 43: Alternative sequence 2 including overlapping relational expressions.

Test step dynamic behaviour					
Test Step Name : RelationalExpression2 **Group** : ..... **Objective** : ..... **Default** : ..... **Comments** :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		......			
		......			
n		[a>5]			1
		......			
m		[a>6]			1
		......			
Detailed comments: 1) "a" may be a test suite variable or a test suite parameter of type INTEGER.					

In example 43, the second alternative will never be reached since the relational expression in line m is a restriction of relational expression in line n.

From examples 42 and 43, rule 30 can be derived:

	RULE 30: Using relational expressions as alternatives
a)	A relational expression should never restrict the value range of a preceding relational expression in the same alternative sequence using the same variable.
b)	The value range of a relational expression should be different from the whole value range of all preceding relational expressions in the same alternative sequence using the same variable.

12.8.2 Loops

To fulfil the requirement that the execution of a test case shall be finite, it needs to be guaranteed that the execution of a loop will terminate. If the termination condition only depends on the behaviour of the IUT, then the termination of the loop cannot be guaranteed in each case. This is shown in example 44.

EXAMPLE 44: Non-terminating loops.

Test step dynamic behaviour		
Test step name	:	SampleStep
Group	:	SAMPLE/STEP/
Objective	:	Put the logical channel into the Data Transfer State P4 (D1)
Default	:	
Comments	:	Sample illustrating a possible candidate for an endless loop

No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
		......			
		!RESET	RST_0(LCI)		
	P4D1PR	?RESETC	RSTC_1(LCI)		
		......			
		?RESET	RST_0(LCI)		
		......			
		?DATA	DAT_1A(LCI,PR)		
		GOTO P4D1PR			
		?RR	RR_1(LCI)		
		GOTO P4D1PR			
		......			

After having sent a packet of type "RESET" the tester expects an acknowledgement packet of type "RESET" or type "RESETC" from the IUT. If the IUT sends a packet of type "DATA" or "RR" (correct but not expected IUT behaviour) then this packet will be received, the tester will execute the GOTO statement and continue on waiting for an acknowledgement packet of the expected types. Therefore, if the IUT only sends packets of type "DATA" or "RR" the loop will never terminate.

From the scenario discussed above the following rule 31 can be derived.

RULE 31: Loop termination
Do not use conditions for terminating loops, which depend only on the behaviour of the IUT.

12.8.3 Avoiding deadlocks

If a tester sends a PDU to the IUT expecting a given response of the IUT, then the tester should take into consideration that the IUT may send a different PDU to the expected one, or that the IUT will react incorrectly by sending no PDU back. If this is not taken into account in specifying the ATS used, then the tester will wait for an infinite time for the expected IUT response. To avoid such deadlock situations, rule 32, outlined below, is given.

	RULE 32: Avoiding deadlocks
a)	Make sure that each alternative sequence of receive events contains an OTHERWISE statement (without any qualifier) for each PCO.
b)	Make sure that each alternative sequence of receive events contains at least one TIMEOUT event (implying that a corresponding timer was started).

NOTE: An OTHERWISE statement and a TIMEOUT event should, at least, be treated in the default.

13 Guidelines on default trees

13.1 Introduction

Default trees form a set of alternatives which are thought to be of less interest for the actual test case. Therefore, the default tree can be thought of as a tree attachment of a similar default test step attached as an additional last alternative to every set of alternatives. Because of this, the construction of the default tree can be derived by analyzing the following aspects:

- straightforward specification of test cases;
- worst case analysis of the protocol.

These two aspects are handled in the following subclauses in more detail.

13.2 Straightforward specification of test cases

Preambles should only emphasise the paths leading to the test body. All other alternatives which would result in the test body not being reached can be handled within the default tree. This may happen on receipt of ASPs/PDUs not matching any given constraint. As such events are not of interest (as they do not comply to the test purpose) they can be simply summarized by a general receive TTCN statement combining all ASP or PDU types allowed to be received on this PCO without analysing any parameters. This can be done by defining a CHOICE over all allowed ASP or PDU types, as shown in example 45.

The OTHERWISE construct cannot be used for this purpose as the OTHERWISE also matches syntactically and/or semantically invalid ASPs/PDUs received on the PCO concerned, which should in all cases result in a FAIL verdict.

EXAMPLE 45: Type and constraint definition for a general matching ACSE and presentation ASP.

ASN.1 ASP type definition		
ASP name	:	ACSE_PRES_SP
PCO type	:	APSAP
Comments	:	Choice of all possible ACSE and presentation service primitives applicable for the FTAM layer
Type definition		

```
CHOICE {
        [1]     A_ASSOCIATE_Request,
        [2]     A_ASSOCIATE_Indication,
        [3]     A_RELEASE_Request,
        [4]     A_RELEASE_Indication,
        [5]     A_ABORT_Request,
        [6]     A_ABORT_Indication
        [7]     A_P_ABORT_Indication,
        [8]     P_DATA_Request,
        [9]     P_DATA_Indication,
        [10]    P_SYNC_MINOR_Request,
        [11]    P_SYNC_MINOR_Indication,
        [12]    P_RESYNCHRONIZE_Request,
        [13]    P_RESYNCHRONIZE_Indication
}
```

ASN.1 ASP constraint declaration		
Constraint name	:	AP_MATCH_ANY
ASP type	:	ACSE_PRES_SP
Derivation path	:	
Comments	:	accept any valid incoming ASP from ACSE/Presentation PCO
Constraint value		
?		

NOTE 1: This example uses ASN.1 for the type definitions and constraint declarations. In case where an ATS is only using tabular format this would add a new level of complexity. In this case the possible solution is to define tabular constraints containing the "?" for each possible tabular PDU (or ASP) type and then to use these as set of alternatives on the same level within the default.

Some protocols can easily be transferred from each possible state of the protocol machine to the idle state of the protocol machine by just calling a particular service primitive, e.g. DISCONNECT request or an association ABORT. Therefore, the same postamble can be used for each test case referring to such a protocol. In this case it is very convenient to realize the postamble within the default tree. This default tree may simply consist of the general MATCH (as described above) followed by an ABORT/DISCONNECT request event. This is illustrated in example 46. The OTHERWISE is included afterwards to treat the remaining unexpected invalid ASPs/PDUs. Example 46 shows a first approach for a default tree for use within the FTAM ATSs:

- if the test body is reached then the result variable is set to FAIL in case of the general MATCH;

- if the test body is not reached then the result variable is set to INCONCLUSIVE in case of the general MATCH;

- in case of OTHERWISE the result variable is always set to FAIL.

NOTE 2: The correct assignment of the verdict depends on the setting of the test body variable (see rule 19).

The result variable is always set to INCONCLUSIVE if the inactivity timer (here A) has expired, when a reaction of the IUT is expected. This is due to the fact that it cannot be determined whether the IUT is in error, or would respond if the timer value is increased.

NOTE 3: Also, only in case of a connection (state variable is not equal to "no connection") the abort is sent to return to the IDLE state of the protocol machine.

It should be also recognized that no test step is used within the default tree to hide the sending of the ABORT, because it is not allowed to use test steps within the default tree (see ISO/IEC 9646-3 [3], subclause 14.18.1).

EXAMPLE 46: First approach for a FTAM default tree.

Default dynamic behaviour					
Default name : DEF (X: APSAP)					
Group : FTAM/DEFAULT-LIB/					
Objective : Illustrate first approach for default					
Comments :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		X?ACSE_PRES_SP [TEST_BODY=TRUE]	AP_MATCH_ANY	(FAIL)	1
2		[F_STATE<>NONE]			1.1
3		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	1.1.1
4		[F_STATE = NONE]		R	1.2
5		X?ACSE_PRES_SP [TEST_BODY=FALSE]	AP_MATCH_ANY	(INCONC)	2
6		[F_STATE<>NONE]			2.1
7		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	2.1.1
8		[F_STATE = NONE]		R	2.2
		X?OTHERWISE		(FAIL)	3
10		[F_STATE<>NONE]			3.1
11		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	3.1.1
12		[F_STATE = NONE]		R	3.2
13		?TIMEOUT A		(INCONC)	4
14		[F_STATE<>NONE]			4.1
15		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	4.1.1
16		[F_STATE = NONE]		R	4.2

Detailed comments:

1) if test body is reached, then verdict is FAIL if anything unexpected (but valid) is reached:
   1.1) and 1.1.1) if association is established (PM state is not NONE) abort association;
   1.2) if no association is established simply end test case;

2) if test body is not reached, then verdict is INCONCLUSIVE if anything unexpected (but valid) is reached:
   2.1) and 2.1.1) if association is established (PM state is not NONE) abort association;
   2.2) if no association is established simply end test case;

3) if anything else is received (which is not valid) the verdict is FAIL:
   3.1) and 3.1.1) if association is established (PM state is not NONE) abort association;
   3.2) if no association is established simply end test case;

4) if TIMEOUT then the inactivity timer has expired resulting in INCONCLUSIVE (i.e. IUT did not respond in time => reason unknown/uncertain):
   4.1) and 4.1.1) if association is established (PM state is not NONE) abort association;
   4.2) if no association is established simply end test case.

RULE 33: Straightforward specification of test cases

a) Use only event sequences leading to the test body within a preamble.
b) Handle all event sequences not leading to the test body within the default tree of the test case/step.
c) If the very same event sequence can be used to transfer the IUT from each possible state to the idle state, then realize this event sequence as a postamble.

13.3 Worst case analysis

In the first approach all non-matching PDU constraints were considered to be treated within the default tree. This is normally not sufficient because protocols may allow other protocol elements which need to be treated as well. These PDUs may be sent either in any state of the protocol machine (e.g. ABORTS) or in specific states (e.g. FTAM F_CANCEL during data transfer).

In addition, depending on the kind of PDU received, or even on the value of a PDU parameter, a certain corresponding postamble needs to be executed in order to terminate the test case. In case of example 46, the ABORT would require an empty postamble as no further action is required, in contrast to the F-CANCEL which would require the association to be aborted (therefore, it could be handled by the general MATCH as well).

Example 47 below completes the FTAM default tree example. It may be necessary to consider all ABORT alternatives (user and provider aborts on each level) for diagnostic reasons by specifying more detailed constraints. This aspect is not considered within example 47.

EXAMPLE 47: Complete example for a FTAM default tree.

Default dynamic behaviour					
Default name : DEF (X: APSAP)					
Group : FTAM/DEFAULT-LIB/					
Objective : Illustrate first complete default					
Comments :					
No.	**Label**	**Behaviour description**	**Constraints ref**	**Verdict**	**Comments**
1		X?A_ABRind [TEST_BODY=TRUE]	A_ABRind501	FAIL	1
2		X?A_ABRind [TEST_BODY=FALSE]	A_ABRind501	INCONC	2
3		X?ACSE_PRES_SP [TEST_BODY=TRUE]	AP_MATCH_ANY	(FAIL)	3
4		[F_STATE<>NONE]			
5		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	
6		[F_STATE = NONE]		R	
7		X?ACSE_PRES_SP [TEST_BODY=FALSE]	AP_MATCH_ANY	(INCONC)	
8		[F_STATE<>NONE]			
9		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	
10		[F_STATE = NONE]		R	
11		X?OTHERWISE		(FAIL)	
12		[F_STATE<>NONE]			
13		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	
14		[F_STATE = NONE]		R	
15		?TIMEOUT A		(INCONC)	
16		[F_STATE<>NONE]			
17		X!A_ABRreq	A_ABRreqbase (FUABRQbase)	R	
18		[F_STATE = NONE]		R	
Detailed comments: 1) an Abort is received and the test body is reached resulting in FAIL; 2) an Abort is received and the test body is not reached resulting in INCONCLUSIVE; 3) the remaining lines are as for example 46.					

14 TTCN extensions

14.1 Introduction

The applicability of the conformance testing methodology and framework defined in the series ISO/IEC 9646 [1] to [8] was originally limited to peer-to-peer communication. But the scope of OSI is not restricted to such peer-to-peer configurations, it also covers so-called multi-party configurations. Multi-party configurations exist within the OSI network management, for OSI network routeing, within OSI transaction processing and also for ISDN. There are some extensions to ISO/IEC 9646 [1] to [8] being standardized handling the specific aspects resulting from testing multi-party configurations. One of these extensions handles concurrency in TTCN. Concurrency in TTCN is not only applicable for specifying test cases in a multi-party configuration, but also for the definition of test cases handling multiplexing/de-multiplexing and splitting/recombining for both single-party and multi-party configurations.

In the following, some guidelines concerning these TTCN extensions are given. Because the standardization of the extensions is still in progress and there is little practical experience in handling the extensions, the given guidelines should be regarded as **preliminary guidelines** which may be expanded in the future, when real ATS examples are available.

14.2 Declarations

There is a series of new declaration types supported by the extensions, e.g. test component declarations, test component configuration declarations, demultiplexing declarations, coordination point declarations, coordination message constraints declarations. For all these new declarations the guidelines on naming conventions given in Clause 6 should be taken into account.

Special care should be taken to avoid recursion when declaring the test component configuration types. Such recursion may lead to deadlocks as shown in example 48.

EXAMPLE 48: Test component configuration.

Test component declarations				
Component name	**Component role**	**No. PCOs**	**No. CPs**	**Comments**
MTC1	MTC	0	3	Master test component
TC1	PTC	1	3	Parallel test component 1
TC2	PTC	1	3	Parallel test component 2
TC3	PTC	1	3	Parallel test component 3

Test Component Configuration Declaration			
Components used	**PCOs used**	**CPs used**	**Comments**
MTC1		MCP1, MCP2, MCP3	
TC1	L1	MCP1, CP1, CP3	TC1 is coordinated with TC2, TC3
TC2	L2	MCP2, CP1, CP2	TC2 is coordinated with TC1, TC3
TC3	U	MCP3, CP2, CP3	TC3 is coordinated with TC1, TC2

In the configuration given above the following deadlock situation can arise. TC1 waits on a Coordination Message (CM) from TC2 on CP1, TC2 waits on a CM from TC3 on CP2 without having sent a CM to TC1 on CP1 and TC3 waits on a CM from TC1 on CP3 without having sent a CM to TC2 on CP2. In this situation none of the parallel test components will proceed and wait forever for a CM. Such deadlock situations can only be detected by using timers within the parallel test components which delimit the waiting period or within the main test component for controlling the process of the parallel test components. Having detected a deadlock situation it can be resolved by sending an appropriate CM. To avoid such deadlock situations, recursive test component configurations should not be declared.

RULE 34: Test component configuration declaration
Avoid recursive test component configuration declarations.

14.3 Final verdicts

Final test verdicts assigned in a Parallel Test Component (PTC) have only local significance, i.e. only the PTC assigning the verdict will be terminated, all other test components will proceed. In contrast to this, the Master Test Component (MTC) and all PTCs will be terminated if a test verdict is assigned in the MTC. To ensure that a test step can be re-used in the MTC and/or the PTCs, the setting of final test verdicts within the test steps should be avoided. A test step containing a test verdict would result in the termination of only the PTC in which the test step is attached, whilst the very same test step being attached in the MTC would result in the termination of all test components. For test steps where the assignment of a final test verdict cannot be avoided the test suite specifier should take special care that the test step is either used only in PTCs or only in the MTC and not in both.

14.4 RETURN statement

The TTCN extensions define a RETURN statement, to be used only in default trees. According to the operational semantics of TTCN, a RETURN construct is handled in the following way, by expanding a default tree:

- the RETURN is replaced by an ACTIVATE followed by a GOTO construct with a label being placed at the head of the current set of alternatives.

However, this should not (but can) result in an endless loop.

RULE 35: Default trees with RETURN statement
Special care should be taken by using a RETURN statement within a default tree in order to avoid an endless loop resulting from the expansion of the default tree.

15 Miscellaneous aspects

15.1 Introduction

The following subclauses pick up miscellaneous singular aspects, which are worthwhile mentioning, but which are not so extensive that "guidelines" need to be given.

15.2 ATS structuring and contents list

ISO/IEC 9646-3 [3] requires that a conforming ATS contains the following parts in the specified order:

Part 1) suite overview;
Part 2) declarations part;
Part 3) constraints part; and
Part 4) dynamic part.

In addition, ETS 300 406 [14] requires ATS conventions to be contained in an ATS specification (see also subclauses 4.1 and 4.3). Apart from this, it appears to be very helpful for the reader, if:

a) the ATS has a comprehensive contents list, i.e. the list covers also the subclauses; and
b) the subclauses of the parts 1) to 4) above appear in the order in which they are defined in ISO/IEC 9646-3 [3].

15.3 Use of comments

It is recommended that an ATS is commented using the possibilities given in TTCN. This aspect is important in order to ensure that ATSs are readable, easy to maintain and able to be standardized. As the way of writing comments very much depends on personal style and subjective arguments no style is recommended within this ETR. Nevertheless ATS authors should always keep in mind that comments should:

- help executable test suite realizers;
- help ATS maintainer;
- contain references to base standards for clarifying decisions made.

Specifically, the last aspect gets more and more important if test laboratories have to run test campaigns and need to justify verdicts within test reports.

15.4 Timer durations and units

Timeout values are expressions. The following is recommended with respect to these expressions:

a) whenever possible use a test suite parameter as the duration;
b) when a duration derived from another duration is used, like e.g. "HALF_T1", then define a new timer and a new test suite parameter whenever possible, instead of using the same timer in the behaviour description and overwriting its value by an explicit expression;
c) when the duration of a timer is a small INTEGER in a given unit, then use in preference the next smaller unit (e.g. if a duration is one second, use the unit "**ms**" instead).

15.5 Use of labels/GOTOs

The use of **GOTO**s within specification/programming languages is usually thought to be bad style. This is because of the problem that GOTOs may violate the normal control flow and normally need a lot of restrictions and limitations within the definition of the language. This can also be seen within TTCN, which limits the definition and usage of labels.

NOTE: These restrictions are neither checked within the Extended Backus-Naur Form (EBNF) notation of Machine Processable TTCN (TTCN.MP) nor within the notation used for TTCN.GR. Such features need to be verified by using tools supporting this kind of semantic checking.

Therefore, it is recommended that the use of GOTOs is limited as much as possible and treated very carefully. Whenever possible, the **REPEAT** construction should be used instead.

15.6 PCOs and their names

ISO/IEC 9646-3 [3] uses the name "L" for the PCO declarations included as examples. This has had the effect that many ATSs use the same name "L" for their (single) PCO, e.g. ISO/IEC 8882-2 [9] (X.25 layer 2), ISO/IEC 8882-3 [10] (X.25 layer 3), I-ETS 300 313 [15] (ISDN signalling layer 2) and I-ETS 300 322 [16] (ISDN signalling layer 3). When these ATSs are used as stand-alone test specification standards, this causes no problem, but when these ATSs are used all together within a profile test specification, this naming is very inconvenient.

When e.g. an ISDN telematic terminal is tested, then the following connections on the lower layers have to be established and maintained **at the same time**:

a) data link connection on the D-channel for the ISDN signalling;

b) network connection on the D-channel for the ISDN signalling;

c) data link connection on the B-channel;

d) network connection on the B-channel for the X.25 packet layer.

This has the effect that in the profile-specific test description behaviour is specified, where different PCOs with the same names have to be referenced.

Therefore, test suite specifiers should use meaningful PCO names, which are specific for the particular service accessed.

Annex A: List of rules

Table A.1: List of all rules

No.	Rule	Page
1	Statement of naming conventions	18
2	Coverage of naming conventions	18
3	General properties of naming conventions	19
4	Specific naming rules for test suite parameters/constants/variables test case variables and formal parameters	22
5	Specific naming rule for timers	22
6	Specific naming rule for PDU/ASP/structured types	23
7	Specific naming rule for PDU/ASP/structured types constraints	23
8	Specific naming rule for test suite operations	24
9	Specific naming rule for aliases	24
10	Specific naming rule for test steps	26
11	Selecting the ASN.1 format for type definitions	28
12	Further guidelines on type definitions	29
13	Specification of test suite operations	31
14	General aspects of specifying constraints	35
15	Relation between base constraints and modified constraints	36
16	Static and dynamic chaining	37
17	Parameterization of constraints	39
18	Constraint values	41
19	Verdict assignment in relation to the test body	47
20	Test body entry marker	47
21	State variable	47
22	State checking event sequences	48
23	Easy adaptation of test steps to test cases	49
24	Minimizing complexity of test steps	51
25	Nesting level of test steps	51
26	Recursive tree attachment	51
27	Verdict assignment within test steps	53
28	Parameterized test steps	57
29	Combining statements in a sequence of alternatives	58
30	Using relational expressions as alternatives	59
31	Loop termination	60
32	Avoiding deadlocks	61
33	Straightforward specification of test cases	63
34	Test component configuration declaration	65
35	Default trees with RETURN statement	66

Annex B: List of examples

Table B.1: List of all examples

History

Document history	
October 1994	First Edition
February 1996	Converted into Adobe Acrobat Portable Document Format (PDF)