



INTERNATIONAL TELECOMMUNICATION UNION

CCITT

THE INTERNATIONAL
TELEGRAPH AND TELEPHONE
CONSULTATIVE COMMITTEE

X.290

(11/1988)

SERIES X: DATA COMMUNICATION NETWORKS
OPEN SYSTEMS INTERCONNECTION (OSI)
PROTOCOL SPECIFICATIONS, CONFORMANCE
TESTING

Conformance testing

**OSI CONFORMANCE TESTING METHODOLOGY
AND FRAMEWORK FOR PROTOCOL
RECOMMENDATIONS FOR CCITT
APPLICATIONS**

Reedition of CCITT Recommendation X.290 published in
the Blue Book, Fascicle VIII.5 (1988)

NOTES

- 1 CCITT Recommendation X.290 was published in Fascicle VIII.5 of the *Blue Book*. This file is an extract from the *Blue Book*. While the presentation and layout of the text might be slightly different from the *Blue Book* version, the contents of the file are identical to the *Blue Book* version and copyright conditions remain unchanged (see below).
- 2 In this Recommendation, the expression “Administration” is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Recommendation X.290

**OSI CONFORMANCE TESTING METHODOLOGY AND FRAMEWORK FOR
PROTOCOL RECOMMENDATIONS FOR CCITT APPLICATIONS¹⁾**

(Melbourne, 1988)

The CCITT,

considering

- (a) that Recommendation X.200 defines the Reference Model of Open Systems for CCITT Applications;
- (b) that the objective of OSI will not be completely achieved until systems dedicated to CCITT applications can be tested to determine whether they conform to the relevant OSI protocol Recommendations;
- (c) that standardized test suites should be developed for each OSI protocol Recommendation as a means to:
 - obtain wide acceptance and confidence in conformance test results produced by different testers,
 - provide confidence in the interoperability of equipments which passed the standardized conformance tests;
- (d) the need for defining an international Recommendation to specify the framework and general principles for the specification of conformance test suites and the testing of protocol implementations,

unanimously recommends

- (1) that the general principles, definition of terms and concepts of OSI protocol conformance testing shall be in accordance with Part 1 of this Recommendation;
- (2) that the test methods, test suites and test notation shall be in accordance with Part 2 of this Recommendation.

¹⁾ Recommendation X.290 was developed in close collaboration with the ISO/IEC effort on OSI Conformance Testing Methodology and Framework. At the time of publication, Recommendation X.290 was aligned with the texts of DP 9646/1 and DP 9646/2. Since this work was at an early stage of development, changes should be expected. Consequently, users should exercise caution in applying this Recommendation.

CONTENTS

Part 1 – General concepts

- 0 Introduction
- 1 Scope and field of application
- 2 References

SECTION 1 – *Terminology*

- 3 Definitions
- 4 Abbreviations

SECTION 2 – *Overview*

- 5 The meaning of conformance in OSI*
- 6 Conformance and testing
- 7 Test methods
- 8 Test suites
- 9 Relationships between, concepts and roles
- 10 Compliance

Part 2 – Abstract test suite specification

- 0 Introduction
- 1 Scope and field of application
- 2 References
- 3 Definitions
- 4 Abbreviations
- 5 Compliance

SECTION 1 – *Requirements on protocol specifiers*

- 6 Conformance requirements in OSI* Recommendations*
- 7 PICS proformas

SECTION 2 – *Requirements on abstract test suite specifiers*

- 8 Test suite production process
- 9 Determining the conformance requirements and PICS
- 10 Test suite structure
- 11 Generic test case specification
- 12 Abstract test methods
- 13 Specification of abstract test suites
- 14 Use of an abstract test suite specification
- 15 Test suite maintenance

Annex A – Options

Annex B – Guidance for protocol Recommendations* writers

Annex C – Incomplete static conformance requirements

Annex D – Tree and tabular combined notation

Appendix I – Applicability of the test methods to OSI* protocols

Appendix II – Index to definitions of terms

Appendix III – Examples for guidance for PICS proforma

Appendix IV – Example of choice of abstract test methods

Part 1 – General concepts

0 Introduction

The objective of OSI will not be completely achieved until systems can be tested to determine whether they conform to the relevant “OSI or related CCITT X-series or T-series” (hereafter abbreviated to “OSI*”) protocol “standard(s) or Recommendation(s)” (hereafter abbreviated to “Recommendation(s)*”).

Standardized test suites should be developed for each OSI* protocol Recommendation, for use by suppliers or implementors in self-testing, by users of OSI products, by the Administrations* or other third party testers. This should lead to comparability and wide acceptance of test results produced by different testers, and thereby minimize the need for repeated conformance testing of the same system.

The standardization of test suites requires international definition and acceptance of a common testing methodology and appropriate testing methods and procedures. It is the purpose of this Recommendation to define the methodology, to provide a framework for specifying conformance test suites and define the procedures to be followed during testing.

Conformance testing involves testing both the capabilities and behaviour of an implementation and checking what is observed against the conformance requirements in the relevant Recommendation(s)* and against what the implementor states the implementation's capabilities are.

Conformance testing does not include assessment of the performance nor the robustness or reliability of an implementation. It cannot give judgements on the physical realization of the abstract service primitives, how a system is implemented, how it provides any requested service, nor the environment of the protocol implementation. It cannot, except in an indirect way, prove anything about the logical design of the protocol itself.

The purpose of conformance testing is to increase the probability that different implementations are able to interwork. This is achieved by verifying them by means of a protocol test suite, thereby increasing the confidence that each implementation conforms to the protocol specification. Confidence in conformance to a protocol specification is particularly important when equipment supplied by different vendors is required to interwork.

However, it should be borne in mind that the complexity of most protocols makes exhaustive testing impractical on both technical and economic grounds. Also, testing cannot guarantee conformance to a specification since it detects errors rather than their absence. Thus conformance to a test suite alone cannot guarantee interworking. What it does do is give confidence that an implementation has the required capabilities and that its behaviour conforms consistently in representative instances of communication.

It should be noted that the OSI reference model for CCITT applications (Recommendation X.200) states (in § 4.3):

“Only the external behaviour of Open Systems is retained as the standard of behaviour of real Open Systems”.

This means that although aspects of both internal and external behaviour are described in OSI* Recommendations*, only the requirements on external behaviour have to be met by real open systems. Although some of the methods defined in this Recommendation do impose certain constraints on the implementor, for example that there be some means of realizing control and observation at one or more service access points, it should be noted that other methods defined herein do not impose such constraints.

However, in the case of partial OSI* end-systems which provide OSI* protocols up to a specific layer boundary, it is desirable to test both the external behaviour of the implemented protocol entities and the potential of those entities for supporting correct external behaviour in higher layers.

Detailed investigation of relative benefits, efficiency and constraints of all methods is addressed in various parts of this Recommendation. However, any organization contemplating the use of test methods defined in this Recommendation in a context such as certification should carefully consider the constraints on applicability and the benefits of the different possible test methods.

Testing is voluntary as far as ISO/CCITT is concerned. Requirements for testing in procurement and other external contracts are not a matter for standardization.

1 Scope and field of application

1.1 This Recommendation specifies a general methodology for testing the conformance to OSI* protocol Recommendation(s)* of products in which the Recommendation(s)* are claimed to be implemented. The methodology also applies to testing conformance to transfer syntax Recommendation(s)* to the extent that can be determined by testing each in combination with a specific OSI* protocol.

1.2 This Recommendation is structured into two separate parts:

Part 1 identifies the different phases of conformance testing process, these phases being characterized by four major roles. These roles are:

- a) the specification of abstract test suites for particular OSI* protocols;
- b) the derivation of executable test suites and associated testing tools;
- c) the role of a client of a test laboratory, having an implementation of OSI* protocols to be tested;
- d) the operation of conformance testing, culminating in the production of a conformance test report which gives the results in terms of the Recommendation(s)* and the test suite(s) used.

Additionally, this Part provides tutorial material, together with definition of concepts and terms.

Part 2 defines the requirements and guidance for the specification of abstract test suites for OSI* protocols.

1.3 In both Parts of this Recommendation, the scope is limited to include only such information as is necessary to meet the following objectives:

- a) to achieve an adequate level of confidence in the tests as a guide to conformance;
- b) to achieve comparability between the results of the corresponding tests applied in different places at different times;
- c) to facilitate communication between the parties responsible for the roles described above.

1.4 One such aspect of this scope involves the framework for development of OSI* test suites. For example:

- a) how they should relate to the various types of conformance requirement;
- b) the types of test to be standardized and the types not needing standardization;
- c) the criteria for selecting tests for inclusion in a conformance test suite;
- d) the notation to be used for defining tests;
- e) the structure of a test suite.

1.5 Certification, an administrative procedure which may follow conformance testing, is outside the scope of this Recommendation. Requirements for procurement and contracts are also outside the scope of this Recommendation.

1.6 The Physical layer and Media Access Control protocols are outside the field of application of this Recommendation.

2 References

Recommendation X.200 – Reference model of open systems interconnection for CCITT applications (see also ISO 7498).

Recommendation X.210 – Open systems interconnection layer service definition conventions (see also ISO TR 8509).

Recommendation X.209 – Specification of basic encoding rules for abstract syntax notation one (ASN.1) (see also ISO 8825).

3 Definitions

3.1 Reference model definitions

This Recommendation is based upon the concepts developed in reference model of open systems interconnection for CCITT applications (CCITT X.200), and makes use of the following terms defined in that Recommendation:

- a) (N)-entity
- b) (N)-service
- c) (N)-layer
- d) (N)-protocol
- e) (N)-service-access-point
- f) (N)-relay
- g) (N)-protocol-data-unit
- h) (N)-protocol-control-information
- i) (N)-user-data
- j) real open system
- k) subnetwork
- l) application-entity
- m) application-service-element
- n) transfer syntax
- o) physical layer
- p) data link layer
- q) network layer
- r) transport layer
- s) session layer
- t) presentation layer
- u) application layer
- v) systems-management
- w) application-management
- x) layer-management

3.2 Terms defined in other Recommendations

This Recommendation uses the following terms defined in the OSI Service Conventions (Recommendation X.210):

- a) service-user
- b) service-provider

This Recommendation uses the following term defined in the ASN.1 – Basic Encoding Rules Recommendation (Recommendation X.209):

- c) encoding

3.3 *Conformance testing definitions*

For the purposes of this Recommendation the definitions in §§ 3.4 to 3.8 apply.

3.4 *Basic terms*

3.4.1 **implementation under test (IUT)**

That part of a real open system which is to be studied by testing, which should be an implementation of one or more OSI* protocols in an adjacent user/provider relationship.

3.4.2 **system under test (SUT)**

The real open system in which the IUT resides.

3.4.3 **dynamic conformance requirements**

All those requirements (and options) which determine what observable behaviour is permitted by the relevant OSI* Recommendation(s)* in instances of communication.

3.4.4 **static conformance requirements**

Constraints which are placed in OSI* Recommendations* to facilitate interworking by defining the requirements for the capabilities of an implementation.

Note – Static conformance requirements may be at a broad level, such as the grouping of functional units and options into protocol classes, or at a detailed level, such as the ranges of values that are to be supported for specific parameters or timers.

3.4.5 **capabilities of an IUT**

That set of functions and options in the relevant protocol(s) and, if appropriate, that set of facilities and options of the relevant service definition which are supported by the IUT.

3.4.6 **protocol implementation conformance statement (PICS)**

A statement made by the supplier of an OSI* implementation or system, stating the capabilities and options which have been implemented, and any features which have been omitted.

3.4.7 **PICS proforma**

A document, in the form of a questionnaire, designed by the protocol specifier or conformance test suite specifier, which when completed for an OSI* implementation or system becomes the PICS.

3.4.8 **protocol implementation extra information for testing (PIXIT)**

A statement made by a supplier or implementor of an IUT which contains or references all of the information (in addition to that given in the PICS) related to the IUT and its testing environment, which will enable the test laboratory to run the appropriate test suite against the IUT.

3.4.9 **PIXIT proforma**

A document, in the form of a questionnaire, provided by the test laboratory, which when completed during the preparation for testing becomes a PIXIT.

3.4.10 **conforming implementation**

An IUT which is shown to satisfy both static and dynamic conformance requirements, consistent with the capabilities stated in the PICS.

3.4.11 **system conformance statement**

A document summarizing which OSI* Recommendations* are implemented and to which conformance is claimed.

3.4.12 **client**

The organization that submits a system or implementation for conformance testing.

3.4.13 **test laboratory**

An organization that carries out conformance testing. This can be a third party, a user organization, an Administration*, or an identifiable part of the supplier organization.

3.5 *Types of testing*

3.5.1 **active testing**

The application of a test suite to an SUT, under controlled conditions, with the intention of observing the consequent actions of the IUT.

3.5.2 **passive testing**

The observation of PDU activity on a link, and checking whether or not the observed behaviour is allowed by the relevant Recommendation(s)*.

3.5.3 **multi-layer testing**

Testing the behaviour of a multi-layer IUT as a whole, rather than testing it layer by layer.

3.5.4 **embedded testing**

Testing the behaviour of a single layer within a multi-layer IUT without accessing the layer boundaries for that layer within the IUT.

3.5.5 **basic interconnection testing**

Limited testing of an IUT to determine whether or not there is sufficient conformance to the main features of the relevant protocol(s) for interconnection to be possible, without trying to perform thorough testing.

3.5.6 **capability testing**

Testing to determine the capabilities of an IUT.

Note – This involves checking all mandatory capabilities and those optional ones that are stated in the PICS as being supported, but not checking those optional ones which are stated in the PICS as not supported by the IUT.

3.5.7 **static conformance review**

A review of the extent to which the static conformance requirements are met by the IUT, by comparing the static conformance requirements expressed in the relevant Recommendation(s)* with the PICS and the results of any associated capability testing.

3.5.8 **behaviour testing**

Testing the extent to which the dynamic conformance requirements are met by the IUT.

3.5.9 **conformance testing**

Testing the extent to which an IUT is a conforming implementation.

3.5.10 **conformance assessment process**

The complete process of accomplishing all conformance testing activities necessary to enable the conformance of an implementation or a system to one or more OSI* Recommendations* to be assessed. It includes the production of the PICS and PIXIT documents, preparation of the real tester and the SUT, the execution of one or more test suites, the analysis of the results and the production of the appropriate system and protocol conformance test reports.

3.6 *Terminology of test suites*

3.6.1 **abstract test method**

The description of how an IUT is to be tested, given at an appropriate level of abstraction to make the description independent of any particular implementation of testing tools, but with enough detail to enable tests to be specified for this method.

3.6.2 **abstract testing methodology**

An approach to describing and categorizing abstract test methods.

3.6.3 **abstract test case**

A complete and independent specification of the actions required to achieve a specific test purpose, defined at the level of abstraction of a particular abstract test method. It includes a preamble and a postamble to ensure starting and ending in a stable state (i.e., a state which can be maintained almost indefinitely, such as the “idle” state or “data transfer” state) and involves one or more consecutive or concurrent connections.

Note 1 – The specification should be complete in the sense that it is sufficient to enable a verdict to be assigned unambiguously to each potentially observable outcome (i.e., sequence of test events).

Note 2 – The specification should be independent in the sense that it should be possible to execute the derived executable test case in isolation from other such test cases (i.e., the specification should always include the possibility of starting and finishing in the “idle” state – that is without any existing connections except permanent ones). For some test cases, there may be pre-requisites in the sense that execution might require some specific capabilities of the IUT, which should have been confirmed by results of the test cases executed earlier.

3.6.4 **executable test case**

A realization of an abstract test case.

Note – In general the use of the word “test” will imply its normal English meaning. Sometimes it may be used as an abbreviation for abstract test case or executable test case. The context should make the meaning clear.

3.6.5 **test purpose**

A description of the objective which an abstract test case is designed to achieve.

3.6.6 **generic test case**

A specification of the actions required to achieve a specific test purpose, defined by a test body together with a description of the initial state in which the test body is to start.

3.6.7 **preamble**

The test steps needed to define the path from the starting stable state of the test case up to the initial state from which the test body will start.

3.6.8 **test body**

The set of test steps that are essential in order to achieve the test purpose and assign verdicts to the possible outcomes.

3.6.9 **postamble**

The test steps needed to define the paths from the end of the test body up to the finishing stable state for the test case.

3.6.10 **test step**

A named subdivision of a test case, constructed from test events and/or other test steps, and used to modularize abstract test cases.

3.6.11 **test event**

An indivisible unit of test specification at the level of abstraction of the specification (e.g., sending or receiving a single PDU).

3.6.12 **test suite**

A complete set of test cases, possibly combined into nested test groups, that is necessary to perform conformance testing or basic interconnection testing for an IUT or protocol within an IUT.

3.6.13 **test case**

A generic, abstract or executable test case.

3.6.14 **test group**

A named set of related test cases.

3.6.15 **generic test suite**

A test suite composed of generic test cases, with the same coverage as the complete set of test purposes for the particular protocol, this being the set or a superset of the test purposes of any particular abstract test suite for the same protocol.

3.6.16 **abstract test suite**

A test suite composed of abstract test cases.

3.6.17 **executable test suite**

A test suite composed of executable test cases.

3.6.18 **conformance test suite**

A test suite for conformance testing of one or more OSI* protocols.

Note – It should cover both capability testing and behaviour testing. It may be qualified by the adjectives: abstract, generic or executable, as appropriate. Unless stated otherwise, an “abstract test suite” is meant.

3.6.19 **basic interconnection test suite**

A test suite for basic interconnection testing of one or more OSI* protocols.

3.6.20 **selected abstract test suite**

The subset of an abstract test suite selected using a specific PICS.

3.6.21 **selected executable test suite**

The subset of an executable test suite selected using a specific PICS and corresponding to a selected abstract test suite.

3.6.22 **parameterized abstract test case**

An abstract test case in which all appropriate parameters have been supplied with values in accordance with a specific PICS and PIXIT.

3.6.23 **parameterized executable test case**

An executable test case in which all appropriate parameters have been supplied with values in accordance with a specific PICS and PIXIT.

3.6.24 **parameterized abstract test suite**

A selected abstract test suite in which all test cases have been made parameterized abstract test cases for the appropriate PICS and PIXIT.

3.6.25 **parameterized executable test suite**

A selected executable test suite in which all test cases have been made parameterized executable test cases for the appropriate PICS and PIXIT, and corresponding to a parameterized abstract test suite.

3.7 *Terminology of results*

3.7.1 **repeatability (of results)**

Characteristic of a test case, such that repeated executions on the same IUT lead to the same verdict, and by extension a characteristic of a test suite.

3.7.2 **comparability (of results)**

Characteristic of conformance assessment processes, such that their execution on the same IUT, in different test environments, leads to the same overall summary.

3.7.3 **outcome**

A sequence of test events together with the associated input/output, either identified by an abstract test case specifier, or observed during test execution.

3.7.4 **foreseen outcome**

An outcome identified or categorized in the abstract test case specification.

3.7.5 **unforeseen outcome**

An outcome not identified or categorized in the abstract test case specification.

3.7.6 **verdict**

Statement of “pass”, “fail” or “inconclusive” concerning conformance of an IUT with respect to a test case that has been executed and which is specified in the abstract test suite.

3.7.7 **system conformance test report (SCTR)**

A document written at the end of the conformance assessment process, giving the overall summary of the conformance of the system to the set of protocols for which conformance testing was carried out.

3.7.8 **protocol conformance test report (PCTR)**

A document written at the end of the conformance assessment process, giving the details of the testing carried out for a particular protocol, including the identification of the abstract test cases for which corresponding executable test cases were run and for each test case the test purpose and verdict.

3.7.9 **valid test event**

A test event which is allowed by the protocol Recommendation*, being both syntactically correct and occurring or arriving in an allowed context in an observed outcome.

3.7.10 **syntactically invalid test event**

A test event which syntactically is not allowed by the protocol Recommendation*.

Note – The use of “invalid test event” is deprecated.

3.7.11 **inopportune test event**

A test event which, although syntactically correct, occurs or arrives at a point in an observed outcome when not allowed to do so by the protocol Recommendation*.

3.7.12 **“pass” verdict**

A verdict given when the observed outcome satisfies the test purpose and is valid with respect to the relevant Recommendation(s)* and with respect to the PICS.

3.7.13 **“fail” verdict**

A verdict given when the observed outcome is syntactically invalid or inopportune with respect to the relevant Recommendation(s)* or the PICS.

3.7.14 **“inconclusive” verdict**

A verdict given when the observed outcome is valid with respect to the relevant Recommendation(s)* but prevents the test purpose from being accomplished.

3.7.15 **conformance log**

A record of sufficient information necessary to verify verdict assignments as a result of conformance testing.

3.8 *Terminology of test methods*

3.8.1 **point of control and observation (PCO)**

A point at which control and observation is specified in a test case.

3.8.2 **lower tester**

The abstraction of the means of providing, during test execution, control and observation at the appropriate PCO either below the IUT or remote from the IUT, as defined by the chosen abstract test method.

3.8.3 **upper tester**

The abstraction of the means of providing, during test execution, control and observation of the upper service boundary of the IUT, plus the control and observation of any relevant abstract local primitive.

3.8.4 **abstract (N)-service-primitive ((N)-ASP)**

An implementation independent description of an interaction between a service-user and a service-provider at an (N)-service boundary, as defined in an OSI* service definition Recommendation*.

3.8.5 **abstract local primitive (ALP)**

An abbreviation for a description of control and/or observation to be performed by the upper tester, which cannot be described in terms of ASPs but which relates to events or states defined within the protocol Recommendation(s)* relevant to the IUT.

Note – The PIXIT will indicate whether or not a particular ALP can be realized within the SUT. The ability of the SUT to support particular ALPs as specified in the PIXIT will be used as a criterion in the test selection process.

3.8.6 **test coordination procedures**

The rules for cooperation between the lower and upper testers during testing.

3.8.7 **test management protocol**

A protocol which is used as a realization of the test coordination procedures for a particular test suite.

3.8.8 **local test methods**

Abstract test methods in which the PCOs are directly at the layer boundaries of the IUT.

3.8.9 **external test methods**

Abstract test methods in which the lower tester is separate from the SUT and communicates with it via an appropriate lower layer service-provider.

Note – The service-provider is immediately beneath the (lowest layer) protocol which is the focus of the testing, and may involve multiple OSI layers.

3.8.10 **distributed test method**

An external test method in which there is a PCO at the layer boundary at the top of the IUT.

3.8.11 **coordinated test method**

An external test method for which a standardized test management protocol is defined as the realization of the test coordination procedures, enabling the control and observation to be specified solely in terms of the lower tester activity, including the control and observation of test management PDUs.

3.8.12 **remote test method**

An external method in which there is neither a PCO above the IUT nor a standardized test management protocol; some requirements for test coordination procedures may be implied or informally expressed in the abstract test suite but no assumption is made regarding their feasibility or realization.

3.8.13 **real tester**

The realization of the lower tester, plus either the definition or the realization of the upper tester, plus the definition of the test coordination procedures, as appropriate to a particular test method.

3.8.14 **test realizer**

An organization which takes responsibility for providing, in a form independent of client and IUT, the means of testing IUTs in conformance with the abstract test suite.

4 **Abbreviations**

For the purposes of this Recommendation the following abbreviations apply:

Administration*	Administration or recognized private operating agency
ALP	abstract local primitive
ASP	abstract service primitive
DTE	data terminal equipment
IUT	implementation under test
OSI	open systems interconnection
OSI*	OSI or related CCITT X-series or T-series Recommendations
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
BBSAP	service access point
SCTR	system conformance test report
Recommendation*	Standard or Recommendation
SUT	system under test
TM-PDU	test management PDU

5 The meaning of conformance in OSI

5.1 Introduction

In the context of OSI*, a real system is said to exhibit conformance if it complies with the requirements of applicable OSI* Recommendations* in its communication with other real systems.

Applicable OSI* Recommendations* include protocol Recommendations*, and transfer syntax Recommendations* inasmuch as they are implemented in conjunction with protocols.

OSI* Recommendations* form a set of interrelated Recommendations* which together define behaviour of open systems in their communication. Conformance of a real system will, therefore, be expressed at two levels, conformance to each individual Recommendation*, and conformance to the set.

Note – If the implementation is based on a predefined set of Recommendations*, often referred to as a functional standard or profile, the concept of conformance can be extended to specific requirements expressed in the functional standard or profile, as long as they do not conflict with the requirements of the base Recommendations*.

5.2 Conformance requirements

5.2.1 The conformance requirements in a Recommendation* can be:

- a) mandatory requirements: these are to be observed in all cases;
- b) conditional requirements: these are to be observed if the conditions set out in the Recommendation* apply;
- c) options: these can be selected to suit the implementation, provided that any requirements applicable to the option are observed. More information on options is provided in Annex A.

For example, CCITT essential facilities are mandatory requirements; additional facilities can be either conditional or optional requirements.

Note – The CCITT terms “essential facilities” and “additional facilities” need to be considered in the context of the scope of the CCITT Recommendation concerned; in many cases, essential facilities are mandatory for networks but not for DTEs.

5.2.2 Furthermore, conformance requirements in a Recommendation* can be stated

- a) positively: they state what shall be done;
- b) negatively (prohibitions): they state what shall not be done.

5.2.3 Finally, conformance requirements fall into two groups:

- a) static conformance requirements;
- b) dynamic conformance requirements;

these are discussed in §§ 5.3. and 5.5, respectively.

5.3 *Static conformance requirements*

Static conformance requirements are those that define the allowed minimum capabilities of an implementation, in order to facilitate interworking. These requirements may be at a broad level, such as the grouping of functional units and options into protocol classes, or at a detailed level, such as a range of values that have to be supported for specific parameters of timers.

Static conformance requirements and options in OSI* Recommendations* can be of two varieties:

- a) those which determine the capabilities to be included in the implementation of the particular protocol;
- b) those which determine multi-layer dependencies, e.g., those which place constraints on the capabilities of the underlying layers of the system in which the protocol implementation resides. These are likely to be found in upper layer Recommendations*.

All capabilities not explicitly stated as static conformance requirements are to be regarded as optional.

5.4 *Protocol implementation conformance statement (PICS)*

To evaluate the conformance of a particular implementation, it is necessary to have a statement of the capabilities and options which have been implemented, and any features which have been omitted, so that the implementation can be tested for conformance against relevant requirements, and against those requirements only. Such a statement is called a Protocol Implementation Conformance Statement (PICS).

In a PICS there should be a distinction between the following categories of information which it may contain:

- a) information related to the mandatory, optional and conditional static conformance requirements of the protocol itself;
- b) information related to the mandatory, optional and conditional static conformance requirements for multi-layer dependencies.

If a set of interrelated OSI* protocol Recommendations* has been implemented in a system, a PICS is needed for each protocol. A System Conformance Statement will also be necessary, summarizing all protocols in the system for each of which a distinct PICS is provided.

5.5 *Dynamic conformance requirements*

Dynamic conformance requirements are all those requirements (and options) which determine what observable behaviour is permitted by the relevant OSI* Recommendation(s)* in instances of communication. They form the bulk of each OSI* protocol Recommendation*. They define the set of allowable behaviours of an implementation or real system. This set defines the maximum capability that a conforming implementation or real system can have within the terms of the OSI* protocol Recommendation*.

A system exhibits dynamic conformance in an instance of communication if its behaviour is a member of the set of all behaviours permitted by the relevant OSI* protocol Recommendation(s)* in a way which is consistent with the PICS.

5.6 *A conforming system*

A conforming system or implementation is one which is shown to satisfy both static and dynamic conformance requirements, consistent with the capabilities stated in the PICS, for each protocol declared in the System Conformance Statement.

5.7 *Interworking and conformance*

5.7.1 The primary purpose of conformance testing is to increase the probability that different implementations are able to interwork.

Successful interworking of two or more real open systems is more likely to be achieved if they all conform to the same subset of an OSI* Recommendation*, or to the same selection of OSI* Recommendations*, than if they do not.

In order to prepare two or more systems to interwork successfully, it is recommended that a comparison be made of the System Conformance Statements and PICSs of these systems.

If there is more than one version of a relevant OSI* Recommendation* indicated in the PICSs, the differences between the versions need to be identified and their implications for consideration, including their use in combination with other Recommendations*.

5.7.2 While conformance is a necessary condition, it is not on its own a sufficient condition to guarantee interworking capability. Even if two implementations conform to the same OSI* protocol Recommendation*, they may fail to interwork because of factors outside the scope of that Recommendation.

Trial interworking is recommended in order to detect these factors. Further information to assist interworking between two systems can be obtained by extending the PICS comparison to other relevant information, including test reports and PIXIT (see § 6.2). The comparison can focus on:

- a) additional mechanisms claimed to work around known ambiguities or deficiencies not yet corrected in the Recommendations* or in peer real systems, e.g. solution of multi-layer problems;
- b) selection of free options which are not taken into account in the static conformance requirements of the Recommendations*;
- c) the existence of timers not specified in the Recommendation* and their associated values.

Note – The comparison can be made between two individual systems, between two or more types of product, or, for the PICS comparison only, between two or more specifications for procurement, permissions to connect, etc.

6 Conformance and testing

6.1 Objectives of conformance testing

6.1.1 Introduction

Conformance testing as discussed in this Recommendation is focused on testing for conformance to OSI* protocol Recommendations*. However, it also applies to testing for conformance to OSI* transfer syntax Recommendations*, to the extent that this can be carried out by testing the transfer syntax in combination with an OSI* protocol.

In principle, the objective of conformance testing is to establish whether the implementation being tested conforms to the specification in the relevant Recommendation*. Practical limitations make it impossible to be exhaustive, and economic considerations may restrict testing still further.

Therefore, this Recommendation distinguishes four types of testing, according to the extent to which they provide an indication of conformance:

- a) basic interconnection tests, which provide *prima facie* evidence that an IUT conforms;
- b) capability tests, which check that the observable capabilities of the IUT are in accordance with the static conformance requirements and the capabilities claimed in the PICS;
- c) behaviour tests, which endeavour to provide testing which is as comprehensive as possible over the full range of dynamic conformance requirements specified by the Recommendation*, within the capabilities of the IUT;
- d) conformance resolution tests, which probe in depth the conformance of an IUT to particular requirements, to provide a definite yes/no answer and diagnostic information in relation to specific conformance issues; such tests are not standardized.

6.1.2 Basic interconnection tests

6.1.2.1 Basic interconnection tests provide limited testing of an IUT in relation to the main features in a Recommendation*, to establish that there is sufficient conformance for interconnection to be possible, without trying to perform thorough testing.

6.1.2.2 Basic interconnection tests are appropriate:

- a) for detecting severe cases of non-conformance;
- b) as a preliminary filter before undertaking more costly tests;
- c) to give a *prima facie* indication that an implementation which has passed full conformance tests in one environment still conforms in a new environment (e.g. before testing an (N)-implementation, to check that a tested (N – 1)-implementation has not undergone any severe change due to being linked to the (N)-implementation);
- d) for use by users of implementations, to determine whether the implementations appear to be usable for communication with other conforming implementations, e.g. as a preliminary to data interchange.

6.1.2.3 Basic interconnection tests are inappropriate:

- a) as a basis for claims of conformance by the supplier of an implementation;
- b) as a means of arbitration to determine causes for communications failure.

6.1.2.4 Basic interconnection tests should be standardized as either a very small test suite or a subset of a conformance test suite (including capability and behaviour tests). They can be used on their own or together with a conformance test suite. The existence and execution of basic interconnection tests are optional.

6.1.3 *Capability tests*

6.1.3.1 Capability tests provide limited testing of each of the static conformance requirements in a Recommendation*, to ascertain what capabilities of the IUT can be observed and to check that those observable capabilities are valid with respect to the static conformance requirements and the PICS.

6.1.3.2 Capability tests are appropriate:

- a) to check as far as possible the consistency of the PICS with the IUT;
- b) as a preliminary filter before undertaking more in-depth and costly testing;
- c) to check that the capabilities of the IUT are consistent with the static conformance requirements;
- d) to enable efficient selection of behaviour tests to be made for a particular IUT;
- e) when taken together with behaviour tests, as a basis for claims of conformance.

6.1.3.3 Capability tests are inappropriate:

- a) on their own, as a basis for claims of conformance by the supplier of an implementation;
- b) for testing in detail the behaviour associated with each capability which has been implemented or not implemented;
- c) for resolution of problems experienced during live usage or where other tests indicate possible non-conformance even though the capability tests have been satisfied.

6.1.3.4 Capability tests are standardized within a conformance test suite. They can either be separated into their own test group(s) or merged with the behaviour tests.

6.1.4 Behaviour tests

6.1.4.1 Behaviour tests test an implementation as thoroughly as is practical, over the full range of dynamic conformance requirements specified in a Recommendation*. Since the number of possible combinations of events and timing of events is infinite, such testing cannot be exhaustive. There is a further limitation, namely that these tests are designed to be run collectively in a single test environment, so that any faults which are difficult or impossible to detect in that environment are likely to be missed. Therefore, it is possible that a non-conforming implementation passes the conformance test suite; one aim of the test suite design is to minimize the number of times that this occurs.

6.1.4.2 Behaviour tests are appropriate, when taken together with capability tests, as a basis for the conformance assessment process.

6.1.4.3 Behaviour tests are inappropriate for resolution of problems experienced during live usage or where other tests indicate possible non-conformance even though the behaviour tests have been satisfied.

6.1.4.4 Behaviour tests are standardized as the bulk of a conformance test suite.

Note – Behaviour tests include tests for valid behaviour by the IUT in response to valid, inopportune and syntactically invalid protocol behaviour by the real tester. This includes testing the rejection by the IUT of attempts to use features (capabilities) which are stated in the PICS as being not implemented. Thus, capability tests do not need to include tests for capabilities omitted from the PICS.

6.1.5 Conformance resolution tests

6.1.5.1 Conformance resolution tests provide diagnostic answers, as near to definitive as possible, to the resolution of whether an implementation satisfies particular requirements. Because of the problems of exhaustiveness noted in § 6.1.4.1, the definite answers are gained at the expense of confining tests to a narrow field.

6.1.5.2 The test architecture and test method will normally be chosen specifically for the requirements to be tested, and need not be ones that are generally useful for other requirements. They may even be ones that are regarded as being unacceptable for (standardized) abstract conformance test suites, e.g. involving implementation-specific methods using, say, the diagnostic and debugging facilities of the specific operating system.

6.1.5.3 The distinction between behaviour tests and conformance resolution tests may be illustrated by the case of an event such as a Reset. The behaviour tests may include only a representative selection of conditions under which a Reset

might occur, and may fail to detect incorrect behaviour in other circumstances. The conformance resolution tests would be confined to conditions under which incorrect behaviour was already suspected to occur, and would confirm whether or not the suspicions were correct.

6.1.5.4 Conformance resolution tests are appropriate:

- a) for providing a yes/no answer in a strictly confined and previously identified situation (e.g. during implementation development, to check whether a particular feature has been correctly implemented, or during operational use, to investigate the cause of problems);
- b) as a means for identifying and offering resolutions for deficiencies in a current conformance test suite.

6.1.5.5 Conformance resolution tests are inappropriate as a basis for judging whether or not an implementation conforms overall.

6.1.5.6 Conformance resolution tests are not standardized.

Note on § 6.1 – As a by-product of conformance testing, errors and deficiencies in protocol Recommendations may be identified.*

6.2 *Protocol implementation extra information for testing (PIXIT)*

In order to test a protocol implementation, the test laboratory will require information relating to the IUT and its testing environment in addition to that provided by the PICS. This “Protocol Implementation eXtra Information for Testing” (PIXIT) will be provided by the client submitting the implementation for testing, as a result of consultation with the test laboratory.

The PIXIT may contain the following information:

- a) information needed by the test laboratory in order to be able to run the appropriate test suite on the specific system (e.g., information related to the test method to be used to run the test cases, addressing information);
- b) information already mentioned in the PICS and which needs to be made precise (e.g. a timer value range which is declared as a parameter in the PICS should be specified in the PIXIT);
- c) information to help determine which capabilities stated in the PICS as being supported are testable and which are untestable;
- d) other administrative matters (e.g. the IUT identifier, reference to the related PICS).

The PIXIT should not conflict with the appropriate PICS.

The abstract test suite specifier, test realizer and test laboratory will all contribute to the development of the PIXIT proforma.

6.3 *Conformance assessment process outline*

6.3.1 The main feature of the conformance assessment process is a configuration of equipment allowing exchanges of information between the IUT and a real tester. These are controlled and observed by the real tester.

6.3.2 In conceptual outline, conformance testing should include several steps, involving both static conformance reviews and live testing phases, culminating in the production of a test report which is as thorough as is practical.

6.3.3 These steps are:

- a) analysis of the PICS;
- b) test selection and parameterization;
- c) basic interconnection testing (optional);
- d) capability testing;
- e) behaviour testing;
- f) review and analysis of test results;
- g) synthesis, conclusions and conformance test report production.

These are illustrated in Figure 1/X.290 Part 1.

Prior to the execution of any of the tests, the IUT's PICS and PIXIT are input to the test case selection and parameterization process.

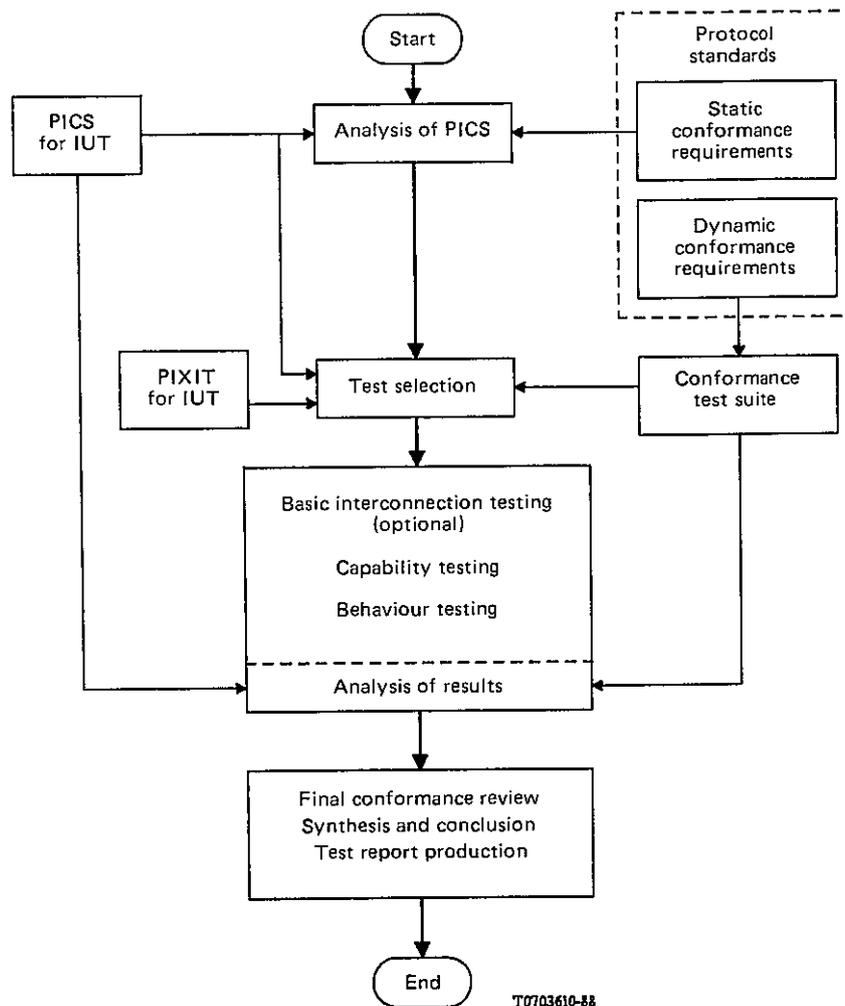


FIGURE 1/X.290, Part 1
Conformance assessment process outline

6.4 *Analysis of results*

6.4.1 *General*

6.4.1.1 *Outcomes and verdicts*

The observed outcome (of the test execution) is the series of events which occurred during execution of a test case; it includes all input to and output from the IUT at the points of control and observation.

The foreseen outcomes are identified and defined by the abstract test case specification taken in conjunction with the protocol Recommendation*. For each test case, there may be one or more foreseen outcomes. Foreseen outcomes are defined primarily in abstract terms.

A verdict is a statement of pass, fail or inconclusive to be associated with every foreseen outcome in the abstract test suite specification.

The analysis of results is performed by comparing the observed outcomes with foreseen outcomes.

The verdict assigned to an observed outcome is that associated with the matching foreseen outcome. If the observed outcome is unforeseen then the abstract test suite specification will state what default verdict shall be assigned.

The means by which the comparison of the observed outcomes with the foreseen outcomes is made is outside the scope of this Recommendation.

Note – Amongst the possibilities are:

- a) manual or automated comparison (or a mixture);
- b) comparison at or after execution time;
- c) translating the observed outcomes into abstract terms for comparison with the foreseen outcomes or translating the foreseen outcomes into the terms used to record the observed outcomes.

The verdict will be pass, fail or inconclusive:

- a) pass means that the observed outcome satisfies the test purpose and is valid with respect to the relevant Recommendation(s)* and with respect to the PICS;
- b) fail means that the observed outcome is syntactically invalid or inopportune with respect to the relevant Recommendation(s)* or the PICS;
- c) inconclusive means that the observed outcome is valid with respect to the relevant Recommendation(s)* but prevents the test purpose from being accomplished.

The verdict assigned to a particular outcome will depend on the test purpose and the validity of the observed protocol behaviour.

The verdicts made in respect of individual test cases will be synthesized into an overall summary for the IUT based on the test cases executed.

6.4.1.2 *Conformance test reports*

The results of conformance testing will be documented in a set of conformance test reports. These reports will be of two types: a System Conformance Test Report (SCTR), and a Protocol Conformance Test Report (PCTR).

The SCTR, which will always be provided, gives an overall summary of the conformance status of the SUT, with respect to its single or multi-layer IUT. A standard proforma for the SCTR is for further study.

The PCTR, one of which will be issued for each protocol tested in the SUT, documents all of the results of the test cases giving references to the conformance logs which contain the observed outcomes. The PCTR also gives reference to all necessary documents relating to the conduct of the conformance assessment process for that protocol.

A standard proforma for the PCTR is for further study. The ordered list of test cases to be used in the PCTR will be specified in the conformance test suite Recommendation*.

6.4.2 *Repeatability of results*

In order to achieve the objective of credible conformance testing, it is clear that the result of executing a test case on an IUT should be the same whenever it is performed. Statistically, it may not be possible to perform a complete conformance test suite and observe outcomes which are completely identical to those obtained on another occasion: unforeseen events do occur, and this is a feature of the environments involved. Nevertheless, at the test case level, it is very important that every effort is made by the test specifiers and test laboratories to minimize the possibility that a test case produces different outcomes on different occasions.

6.4.3 *Comparability of results*

In order to achieve the ultimate objectives of conformance testing, the overall summary concerning conformance of an IUT has to be independent of the test environment in which the testing takes place. That is to say, the standardization of all of the procedures concerned with conformance testing should result in a comparable overall summary being accorded to the IUT, whether the testing is done by the supplier, a user, or by any third party test house. There are a large number of factors to be studied to achieve this, of which some of the more important are:

- a) careful design of the abstract test case specification to give flexibility where appropriate, but show which requirements have to be met (which is the subject of this Recommendation);
- b) careful specification of the real tester which should be used to run the test suite; again this specification should give flexibility where appropriate, but show which requirements have to be met, including all test coordination procedures (if any);
- c) careful specification of the procedure to be followed in determining how the contents of the PICS are to be used in the analysis of outcomes of test cases; there should be no room for “optimistic” interpretation;
- d) careful specification of the procedures to be followed by test laboratories as regards the repetition of a test case before making a final verdict for that test purpose;
- e) a proforma for a conformance test report;
- f) careful specification of the procedures necessary when synthesizing an overall summary.

6.4.4 *Auditability of results*

For legal reasons, as well as others, it may be necessary to review the observed outcomes from the execution of a conformance test suite in order to make sure that all procedures have been correctly followed. Whether or not analysis has been carried out in a manual or automatic mode, it is essential that all inputs, outputs, and other test events are carefully logged, and the analysis of the results recorded. In some cases this may be the responsibility of the test realizer, who may elect to include the test criteria in the conformance log, as well as all outcomes. In others, it may be the responsibility of the test laboratory, which might be required to follow all standard procedures concerning the recording of results.

Note – As far as auditability is concerned, some automatic procedures would be preferred, but in the event it should be appreciated that from a legal standpoint such automatic procedures would have to be accredited themselves, if they are to be credible.

7 **Test methods**

7.1 *Introduction*

The testing of a given OSI* protocol can require the use of several test methods, as systems under test can come in several configurations, and vary in terms of their ability to allow ways of producing effects applicable to a layer boundary.

This section first characterizes the features of the system under test which are to be taken into consideration, next defines the possible test methods in abstract terms, and finally provides guidance on their applicability to real systems.

7.2 *Classification of real open systems and IUTs for conformance testing*

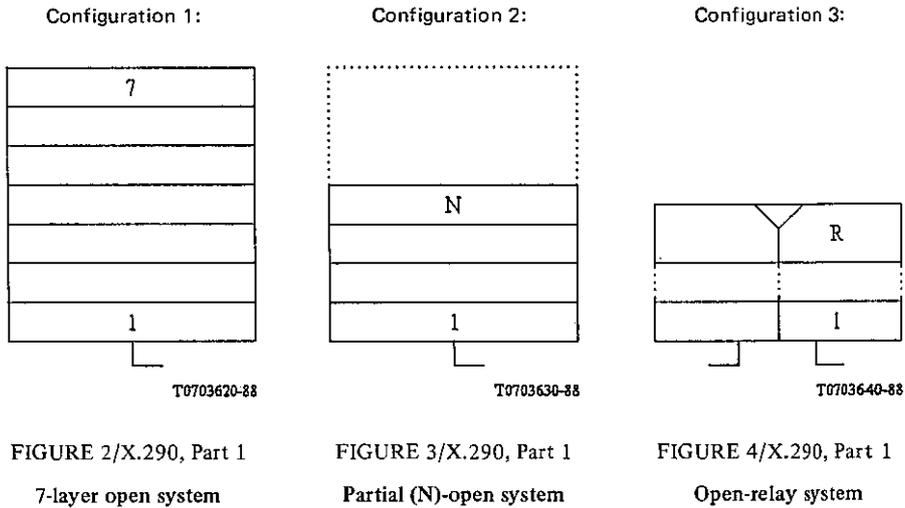
7.2.1 *Classification of systems under test*

7.2.1.1 There is a relation between the test methods and the configurations of the real open systems to be tested. The appropriate test methods vary according to:

- a) the main function of the system (end-system or relay-system);
- b) which layers use OSI* protocols;
- c) whether the alternative of non-OSI* protocols is also available.

7.2.1.2 The following configurations of systems have been identified for the purposes of conformance testing, as illustrated in Figures 2/X.290 to 4/X.290, Part 1. Configurations 1 to 3 are the basic configurations of systems under test (SUTs):

- a) Configuration 1: 7-layer open system (end-system)
These systems use OSI* Recommendation* protocols in all 7 layers.
- b) Configuration 2: Partial (N)-open system (end-system)
These systems use OSI* Recommendation* protocols in layers 1 to N.
- c) Configuration 3: Open relay-systems
These use OSI* protocols in layers 1 to 3 (Network relay-systems) or 1 to 7 (Application relay-systems).



7.2.1.3 Other configurations can be derived from the basic configurations.

A SUT can be a combination of basic configurations 1 and 2, allowing the alternative of using OSI* and non-OSI* protocols above layer N (see Figure 5/X.290, Part 1).

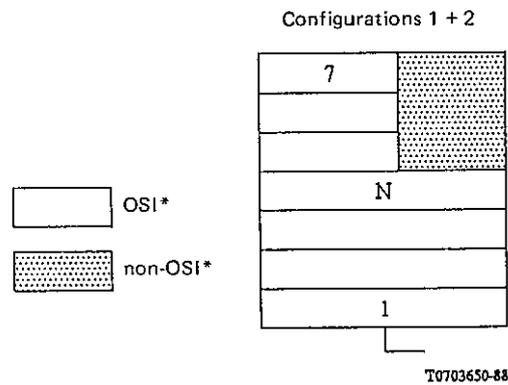


FIGURE 5/X.290, Part 1
Combination of 7-layer and partial (N)-open systems

7.2.2 Identification of the implementation under test (IUT)

An implementation under test (IUT) is that part of a real open system which is to be studied by conformance testing. It should be an implementation of one or more adjacent OSI* protocols.

IUTs can be defined for configurations 1 and 2 of SUTs as single-layer IUTs (one single layer of the SUT is to be tested), or as multi-layer IUTs (a set of any number of adjacent layers of the SUT to be tested in combination).

An IUT defined in an open relay-system will include at least the layer which provides the relay function.

When OSI* and non-OSI* protocols exist in a system, the IUT(s) will be defined for the OSI* mode(s) of operation. Testing non-OSI* protocols is outside the scope of this Recommendation.

Clients and test laboratories will agree on what part of the SUT will be considered to be the IUT.

Test methods need to refer to an abstract testing methodology, based upon the OSI reference model. Considering first end-systems (7-layer or partial (N)-open systems) and single layer IUTs within these systems, abstract test methods are described in terms of what outputs from the IUT are observed and what inputs to it can be controlled. More specifically, an abstract test method is described by identifying the points closest to the IUT at which control and observation are to be exercised.

The OSI* protocol Recommendations* define allowed behaviour of a protocol entity (i.e. the dynamic conformance requirements) in terms of the protocol data units (PDUs) and the abstract service primitives (ASPs) both above and below that entity. Thus the behaviour of an (N)-ASPs and (N – 1)-ASPs (the latter including the (N)-PDUs).

If an IUT comprises more than one protocol entity, the required behaviour can be defined in terms of the ASPs above and below the IUT, including the PDUs of the protocols in the IUT.

The starting point for developing test methods is the conceptual testing architecture, illustrated in Figure 6/X.290, Part 1. It is a “black-box” active testing architecture, based on the definition of behaviour required of the IUT.

The action of the tester shown in Figure 6/X.290, Part 1, can either be applied locally, in which case there is a direct coupling within the system under test, or externally via a link or network. The two sets of interactions, above and below the IUT, can, in practice, be observed and controlled from several different points, locally or externally.

The possible points of control and observation (PCOs) are identified by three factors:

- a) whether it is the ASPs or PDUs which are observed and controlled;
- b) the layer identity of the ASPs or PDUs concerned;
- c) whether they are controlled and observed within the system under test or in a system remote from the system under test – if the latter then the ASPs are distinguished by the addition of a double-quote character (”).

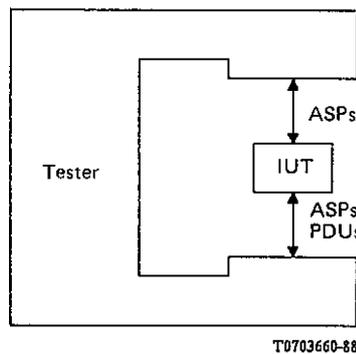


FIGURE 6/X.290, Part 1
Conceptual testing architecture

Possible PCOs within the SUT are illustrated in Figure 7a)/X.290, Part 1. Possible PCOs, when the activity below the IUT is controlled and observed externally are illustrated in Figure 7b)/X.290, Part 1. It can be seen from these figures that there is a multiplicity of possible PCOs in different layers, which offer different degrees of control and observation of IUT behaviour. This Recommendation makes a selection from this set of possible PCOs, defining a limited number of abstract test methods.

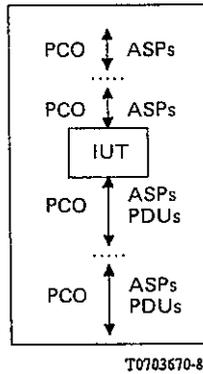


FIGURE 7 a)/X.290, Part 1
Possible PCOs within an SUT

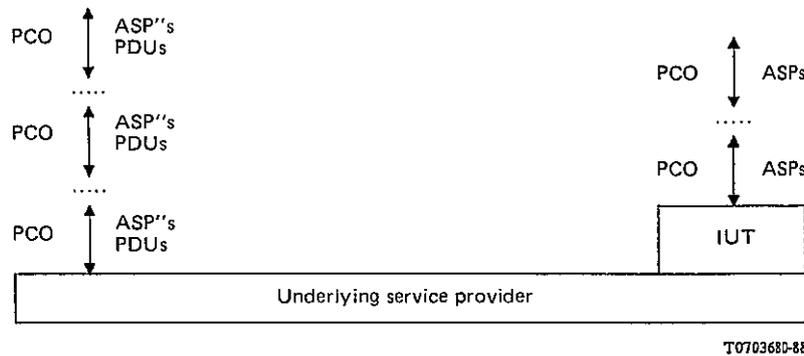


FIGURE 7 b)/X.290, Part 1
Possible PCOs for external testing

If control and observation below, or external from the IUT is specified in terms of ASPs, it will include control and observation of the PDUs carried by those ASPs; but if it is specified in terms of PDUs (at layer N) then the ASPs (at layer N – 1) are not considered to be controlled or observed.

It is assumed that the ASP activity below the IUT can at least be observable and controllable via the peer activity in a remote testing system – i.e. the corresponding ASP's. Thus when the ASPs below the IUT are not controllable nor observable locally, conformance testing can be carried out externally, provided that the underlying service offered is sufficiently reliable for control and observation to take place remotely.

It is possible that the ASP activity above the IUT might not be controllable nor observable, in which case this activity is said to be hidden.

SUTs are not required to provide access to layer boundaries. However, the possible provision of such access and the possible positions of such boundaries with respect to the layers of the IUT are factors to be taken into consideration in the definition of the test methods, which may take advantage of this access to define the test suites in terms of the corresponding ASPs. It does not matter whether the accessible boundaries are accessed via service access points (SAPs) or via some other PCOs.

Figure 8/X.290, Part 1 shows examples of IUTs, with respect to layer boundary accessibility.

Note – In addition, a conformance test suite Recommendation* may define "abstract local primitives". These are used to specify control and observation of events or states which are referred to in the protocol Recommendation* but which are internal to the IUT and which cannot be expressed in terms of ASPs. They are abbreviations for text descriptions of control and observation to be performed by the upper tester.

Similar consideration apply to relay systems (see Part 2 of the Recommendation for details).

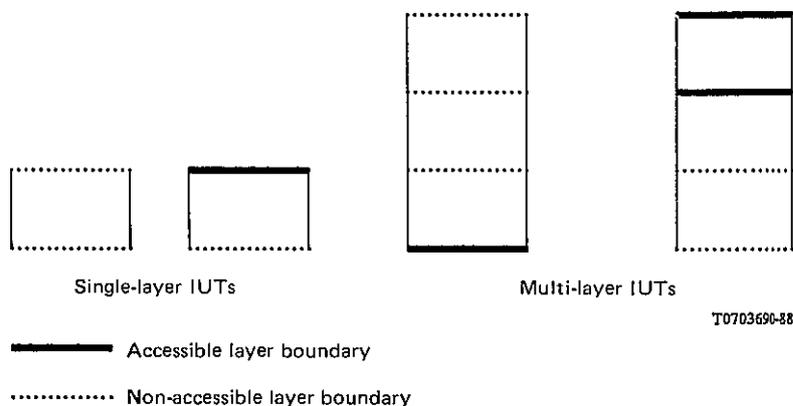


FIGURE 8/X.290, Part 1
Examples of IUT configurations

7.4 *Abstract testing functions*

The definition of abstract test methods requires that the PCOs be distributed over two abstract testing functions, the lower and upper testers.

The lower tester is the abstraction of the means of providing, during test execution, control and observation at the appropriate PCO either below the IUT or remote from the IUT, as defined by the chosen abstract test method. Thus, it is the testing function related to the control and observation of the lower boundary of the IUT. If the action of the tester is local to the SUT, the lower tester will take the place of the lower part of the SUT. If the action of the tester is external to the SUT, the lower tester will rely on the (N – 1)-Service, provided jointly by the lower tester itself, a link and the SUT.

The upper tester is the abstraction of the means of providing, during test execution, control and observation of the upper service boundary of the IUT and of any relevant abstract local primitive.

There is a need for cooperation between the upper tester and the lower tester; the rules for such cooperation are called the test coordination procedures.

The test methods will vary in the way that they specify the test coordination procedures. In some cases it is possible to define a test management protocol to provide the coordination between the upper and lower testers. In other cases it is only possible to describe the requirements to be met by the test coordination procedures without specifying what mechanisms might be used to realize them.

7.5 *Overview of abstract test methods*

7.5.1 *End-system IUTs*

For the IUTs defined within end-system SUTs (configurations 1 and 2 in Figures 2/X.290, Part 1 and 3/X.290, Part 1) four categories of abstract test methods are defined, one local, and three external ones which assume the lower tester is located remotely from the SUT and connected to it by a link or network.

7.5.2 *The local test methods*

The local abstract test methods define the PCOs as being at the service boundaries above and below the IUT. The test events are specified in terms of the ASPs above the IUT and the ASPs and PDUs below the IUT, as illustrated in Figure 9a/X.290, Part 1. Abstractly, a lower tester is considered to observe and control the ASPs and PDUs below the IUT, while an upper tester observes and controls the ASPs above the IUT. Requirements to be met by test coordination procedures used to coordinate the realizations of the upper and lower testers are defined in the abstract conformance test suites, although the test coordination procedures themselves are not.

7.5.3 External test methods

The external test methods use control and observation of the ASPs below the IUT by means of a lower tester separated from the SUT, together with control and observation of the ASPs above the IUT. Three different categories of external abstract test methods are defined, which are referred to as distributed, coordinated, and remote. They vary according to the level of requirement or standardization put on the test coordination procedures, on the access to the layer boundary above the IUT, and on the requirements on an upper tester. They are illustrated in Figures 9b), c) and d)/X.290, Part 1.

The coordinated test method requires that the test coordination procedures used to coordinate the realization of the upper and lower testers be achieved by means of test management protocols. The other two methods do not make any assumptions about the realization of the test coordination procedures.

The distributed and coordinated test methods require specific functions from the upper tester above the IUT. The remote method does not.

The distributed method requires access to the upper boundary of the IUT. The other two methods do not.

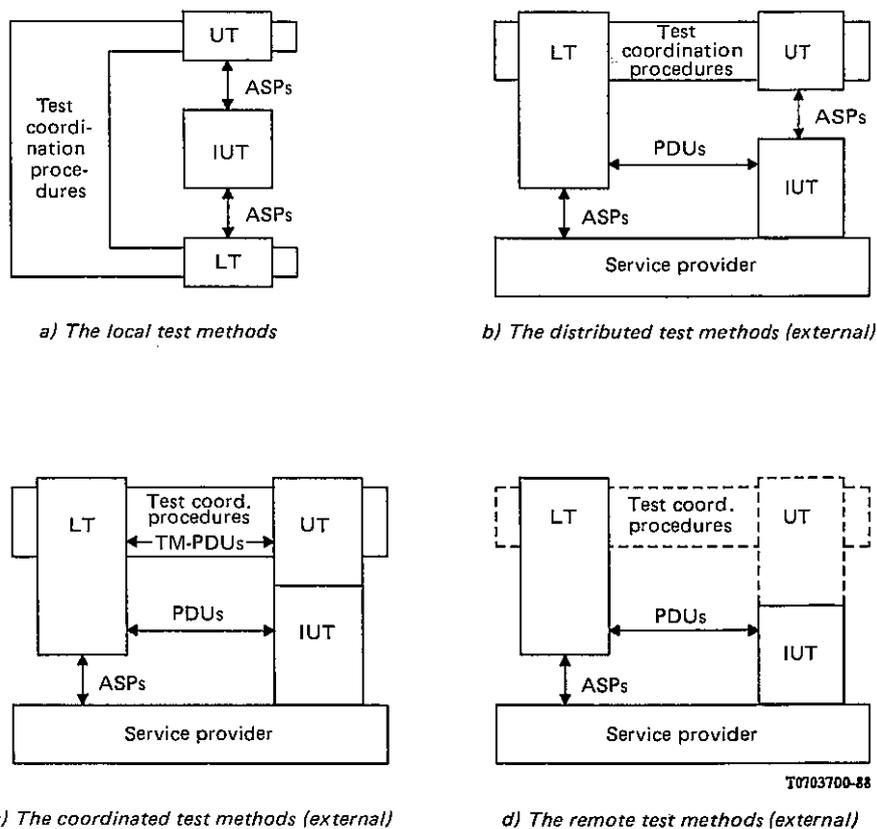


FIGURE 9/X.290, Part 1

Overview of abstract test methods

7.5.4 Variants of end-system test methods

Each category of test methods has variants which can be applied to single-layer IUTs or to multi-layer IUTs. For multi-layer IUTs in which the protocols are to be tested layer by layer, embedded variants can be used.

All abstract test methods for end-systems are fully specified in section 8 of Part 2 of this Recommendation, including their single-layer, multi-layer and embedded variants where applicable.

7.5.5 Relay-system IUTs

For open relay-systems, two test methods are defined, loop-back and transverse. These are fully specified in section 8 of Part 2 of this Recommendation.

7.6 *Applicability of test methods to real open systems*

The architecture and stage of development of a real open system determines the applicability of test methods to it.

Local test methods are useful for systems under development, when their architecture permits the isolation of an IUT, be it single-layer or multi-layer.

External test methods are useful for testing complete or partial end-systems which can attach to telecommunications networks.

Coordinated test methods apply where it is possible to implement a standardized test management protocol in an upper tester in the SUT, above the IUT.

Remote test methods apply when it is possible to make use of some functions of the SUT to control the IUT during testing, instead of using a specific upper tester.

Distributed test methods apply when it is necessary to allow complete freedom for the implementation of the test coordination procedures between the SUT and the lower tester, but to specify in detail the control and observation requirements at both ends.

Single-layer test methods are the most appropriate methods for testing the majority of the protocol conformance requirements.

Multi-layer test methods will be used when conformance to true multi-layer dynamic conformance requirements is to be tested.

Embedded test methods permit the application of single-layer testing to all layers of a multi-layer IUT.

For 7-layer open systems, the preferred methods are the incremental use of appropriate external single-layer embedded methods with the following PCOs:

- a) the upper interface of the application layer as provided by the 7-layer open system, when applicable;
- b) successively, each SAP (or corresponding PCO if there is no SAP as such) below the protocol which is the focus of the testing, as controlled and observed in the external lower tester, starting from the lowest protocol of the IUT and working upwards.

7.7 *Applicability of the test methods to OSI* protocols and layers*

The test methods defined in this Recommendation apply to all layers, with the exception of the Physical layer and the Media Access Control protocols which are outside the scope of this Recommendation. Appendix I of this Recommendation provides guidance on the applicability of test methods to all other layers.

8 **Test suites**

8.1 *Structure*

Test suites have a hierarchical structure (see Figure 10/X.290, Part 1) in which an important level is the test case. Each test case has a narrowly defined purpose, such as that of verifying that the IUT has a certain required capability (e.g. the ability to support certain packet sizes) or exhibit a certain required behaviour (e.g. behave as required when a particular event occurs in a particular state).

Within a test suite, nested test groups are used to provide a logical ordering of the test cases. Test groups may be nested to an arbitrary depth. They may be used to aid planning, development, understanding or execution of the test suite.

Test cases are modularized by using named subdivisions called test steps. Each test case comprises at least one test step: the ordering of events covered by the test purpose ("test body"). It may include further test steps to put the IUT into the state required for the test body to start from (the "preamble") or return to a quiescent state after the test body has finished (the "postamble").

For practical reasons, common test steps may be grouped together into test step libraries. Test step libraries may be structured into nested sets of test steps, to any depth of nesting. Test step libraries may be associated with the whole test suite or with a particular test group or test case.

Furthermore, all test steps consist of an ordering of other test steps and/or test events (e.g. the transfer of a single PDU or ASP to or from the IUT). All test steps are, therefore, equivalent to an ordering of test events (after expansion of the inner test steps).

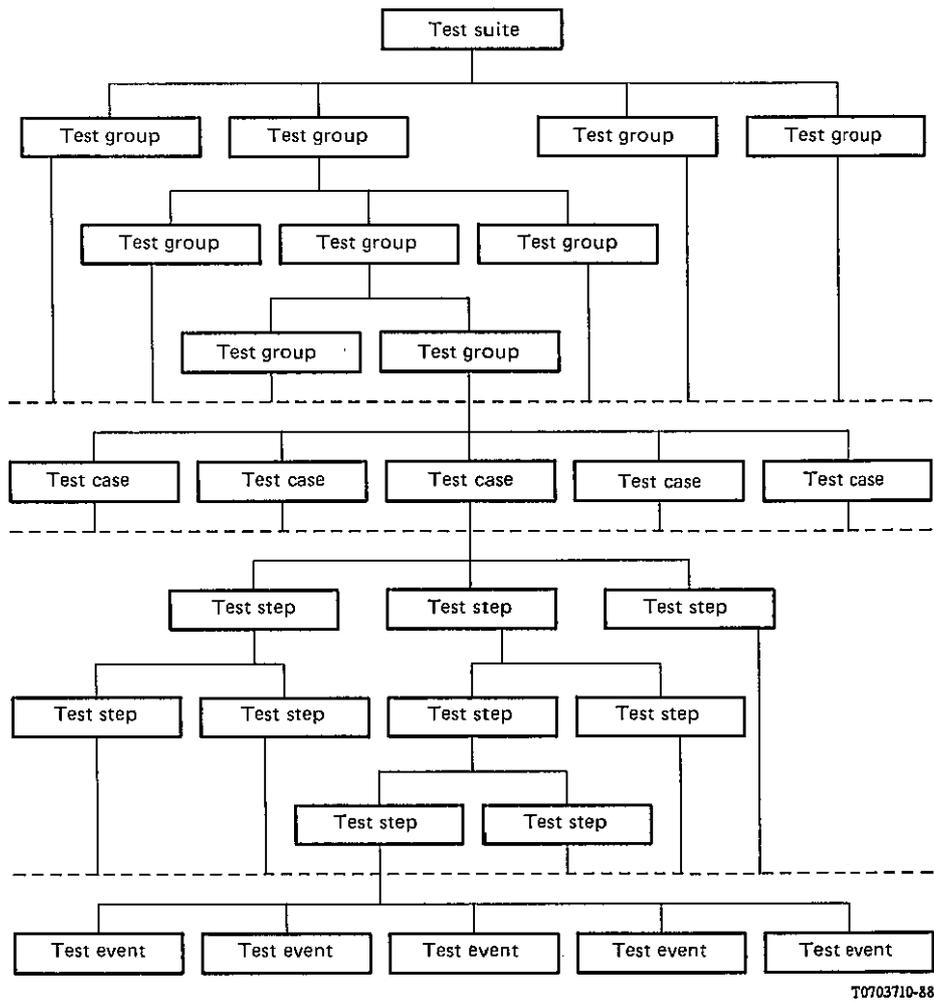


FIGURE 10/X.290, Part 1

Test suite structure

8.2 Generic, abstract and executable test cases

8.2.1 A generic test case is one which:

- a) provides a refinement of the test purpose;
- b) specifies that all sequences of test events (paths) in the test body which correspond to verdicts of “pass”, “fail” and “inconclusive”, using a specialized notation;
- c) is used as the common root of corresponding abstract test cases for different abstract test suites for the same protocol;
- d) includes a description of the initial state in which the test body should start, in lieu of a preamble;
- e) need not describe the postamble;
- f) is specified using the style of either the Remote or Distributed Single-layer test methods.

8.2.2 An abstract test case is derived from a generic case and the relevant protocol specification; it:

- a) specifies the test case in terms of a particular test method;
- b) adds a more precise specification for sequences of events which are only described informally in the generic test case;
- c) adds the sequences of test events required to achieve the objectives of the preamble and postamble of the generic test case using a specialized notation.

8.2.3 An executable test case is derived from an abstract test case, and is in a form which allows it to be run on a real tester for testing a real implementation.

8.2.4 The terms generic, abstract and executable are used to describe test suites, which comprise generic, abstract and executable test cases, respectively.

8.2.5 A generic test suite has the coverage of the set or a superset of all possible abstract test suites for a particular protocol.

9 Relationships between concepts and roles

Figure 11/X.290, Part 1 is a pictorial representation of the relation between the various Recommendations and the processes of producing generic, abstract and executable test suites and test reports.

Part 2 concerns the production of testable protocol Recommendations and abstract test suite Recommendations. Part 1 provides general concepts and definitions.

Note – Other aspects of the conformance assessment process, such as executable test derivation, preparation of the IUT, PICS and PIXIT by the client and test laboratory role are for further study.

10 Compliance

In this Recommendation, “compliance” refers to meeting the requirements specified by the Recommendation. This word is used as an attempt to eliminate confusion between *compliance* to this Recommendation and *conformance* of a protocol implementation to protocol Recommendations.

This part of the Recommendation contains no compliance requirements.

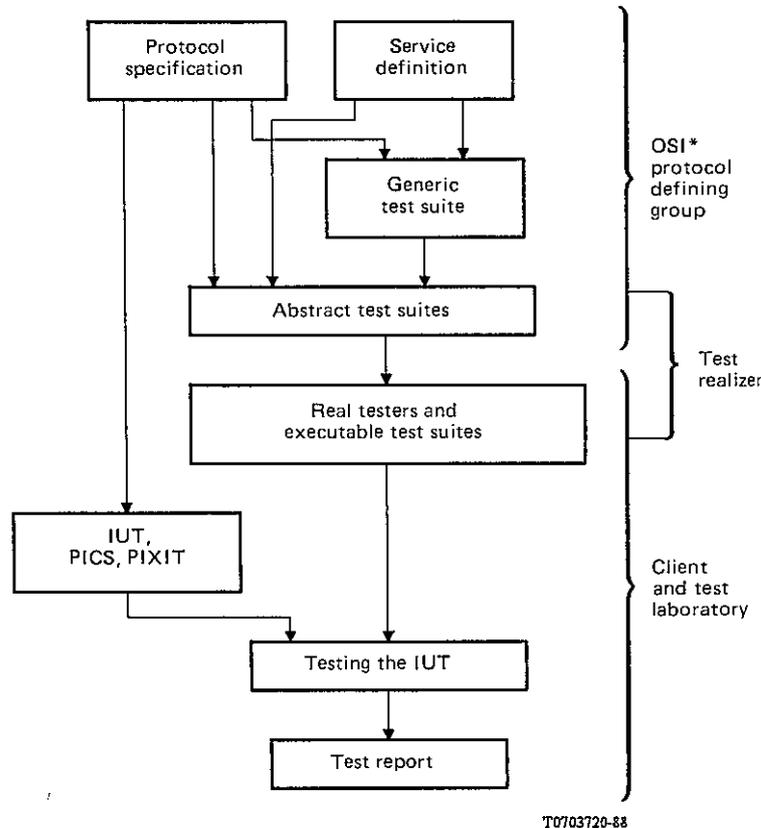


FIGURE 11/X.290, Part 1
Relationships between concepts and roles

Part 2 – Abstract test suite specification

0 Introduction

This part of the Recommendation provides a common approach to the specification of “OSI or related CCITT X-series or T-series” (hereafter abbreviated to “OSI*”) conformance test suites at a level which is independent of the means of executing those test suites (hereafter called “abstract test suites”). This level of abstraction is suitable for standardization and facilitates the comparison of results produced by different organizations which run the corresponding executable test suites.

Section One recalls that there are requirements put on the OSI* protocol specifiers which have to be fulfilled before there can be an objective basis for the process of developing an abstract test suite. The need for consistent conformance sections and for PICS and PIXIT proformas in OSI* protocol Recommendations* is expressed.

Section Two describes the process of developing an abstract test suite, including the design criteria to be used and guidance on its structure and coverage. The possible abstract test methods are defined and guidance is given to help the test suite specifier to decide which method(s) to use in the production of a particular test suite. The relationship between abstract test suites for different methods is provided by a generic test suite which is independent of the test method. A test notation is defined and requirements and guidance are given on its use for specifying both generic and abstract test cases. These include the subdivision of test cases into test steps and the assignment of verdicts to outcomes.

The test suite specifier is also required to provide information to the test realizers, particularly on the constraints governing test case selection and ordering.

Finally, guidance and requirements are given on test suite maintenance.

1 Scope and field of application

This part of the Recommendation specifies the requirements and provides guidance for the production of system-independent conformance test suites for one or more OSI* Recommendations*. In particular, it applies to the production of all conformance test suite Recommendations* for OSI* protocols.

It applies to the production of conformance test cases which check the adherence of an implementation to the relevant static and/or dynamic conformance requirements by controlling and observing protocol behaviour. The abstract test methods included in this Recommendation are in fact capable of being used to specify any test case which can be expressed abstractly in terms of control and observation of protocol data units, abstract service primitives and abstract local primitives. Nevertheless, for some protocols, test cases may be needed which cannot be expressed in these terms. The specification of such test cases is outside the scope of this Recommendation, although those test cases may need to be included in a Recommendation* for a conformance test suite.

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

The production of conformance test suites for multi-peer or physical layer protocols is outside the scope of this Recommendation.

The relation between abstract test specification and formal description techniques is outside the scope of this Recommendation.

2 References

- | | |
|----------------------|--|
| Recommendation X.200 | <i>Reference model of open systems interconnection for CCITT applications</i> (see also ISO 7498). |
| Recommendation X.214 | <i>Transport service definition for open systems interconnection for CCITT applications</i> (see also ISO 8072). |
| Recommendation X.224 | <i>Transport protocol specification for open systems interconnection for CCITT applications</i> (see also ISO 8073). |
| Recommendation X.210 | <i>Open systems interconnection layer service definition conventions</i> (see also ISO TR 8509). |
| Recommendation X.208 | <i>Specification of abstract syntax notation one (ASN.1)</i> (see also ISO 8824). |
| Recommendation X.209 | <i>Specification of basic encoding rules for abstract syntax notation one (ASN.1)</i> (see also ISO 8825). |

Recommendation X.290/1 *OSI conformance testing methodology and framework for protocol Recommendations for CCITT applications – Part 1: General concepts.*

ISO 8571-4 *Information processing systems – open systems interconnection – File protocol specification.*

3 Definitions

For the purposes of this Part of the Recommendation, all the definitions in Part 1 of the Recommendation apply.

4 Abbreviations

For the purposes of this Recommendation the abbreviations given in section 4 of Part 1 of this Recommendation apply. The following abbreviations also apply to this Recommendation:

ASP	the abstract service primitive on the side of the service provider remote from the IUT
CM	coordinated multi-layer (test method)
CS	coordinated single-layer (test method)
CSE	coordinated single-layer embedded (test method)
DM	distributed multi-layer (test method)
DS	distributed single-layer (test method)
DSE	distributed single-layer embedded (test method)
FDT	formal description technique
LM	local multi-layer (test method)
LS	local single-layer (test method)
LSE	local single-layer embedded (test method)
RM	remote multi-layer (test method)
RS	remote single-layer (test method)
RSE	remote single-layer embedded (test method)
TTCN	tree and tabular combined notation
YL	loop-back (test method)
YT	transverse (test method)

5 Compliance

5.1 A protocol Recommendation* which complies with this Part of this Recommendation shall satisfy all the requirements stated in Section 1.

Note – Such compliance is a precondition for the protocol Recommendation* to be an effective basis for conformance testing of implementations.

5.2 An abstract test suite specification which complies with this part of this Recommendation:

- a) shall be a conformance test suite;
- b) shall be specified in a test notation standardized by ISO or CCITT;
- c) shall satisfy all the requirements stated in Section 2.

5.3 It is recommended that the test notation used be the Tree and Tabular Combined Notation (TTCN). If TTCN is used, the abstract test suite shall comply with all the requirements in Annex D.

6 Conformance requirements in OSI* Recommendations*

6.1 *Introduction*

The meaning of conformance in OSI* is discussed in Part 1 of this Recommendation. It is necessary that there be an unambiguous and objective understanding of the conformance requirements of an OSI* protocol or transfer syntax Recommendation*, as a prerequisite to the production of an abstract test suite for that Recommendation*. This section states the requirements on protocol specifiers to ensure that there is such an understanding of the conformance requirements.

6.2 *General requirements*

6.2.1 A clear distinction shall be made between static and dynamic conformance requirements. To avoid ambiguity, they should be stated separately from one another.

6.2.2 It shall be clear what conformance to the Recommendation* means, in the sense of what shall be done, what is permitted but not mandatory, and what shall not be implemented, to conform to the Recommendation*.

6.2.3 It shall always be decidable whether an instance of communication conforms dynamically or not.

For example, one should be able to look at a record of PDU activity and decide whether it is valid.

6.2.4 Requirements on the need to produce and content of a PICS shall be stated separately from the requirements on the protocol implementation itself.

6.3 *Conformance sections*

6.3.1 Each OSI* protocol and transfer syntax Recommendation* shall include a conformance section, which shall be expressed clearly and unambiguously.

6.3.2 Conformance sections shall distinguish between the following categories of information that they may contain:

- a) references to sections which state dynamic conformance requirements;
- b) static conformance requirements concerning the protocol implementation;
- c) static conformance requirements concerning multi-layer dependencies;
- d) what has to be stated in the PICS concerning (b) above;
- e) what has to be stated in the PICS concerning (c) above;
- f) what other information shall be provided (e.g. to assist testing) and whether this should be in the PICS or elsewhere.

6.3.3 In connection-oriented protocol Recommendations*, the conformance section should also include:

- a) the option to support either the initiation of a connection or the acceptance of a connection, or both;
- b) the requirement to be able to accept all correct sequences of PDUs received from peers, and respond with correct PDU sequences appropriate to the defined state of the connection;
- c) the requirement to be able to respond correctly to all incorrect sequences of PDUs received, the response being appropriate to the defined state of the connection.

6.4 *Additional guidance for new protocol Recommendations**

It is recognized that although existing protocol Recommendations* can be improved by the progression of an addendum to add a PICS proforma and make the conformance section align with the requirements stated above, it is unrealistic to expect any greater improvement. However, new protocol Recommendations* should be developed using the additional guidance given in Annex B to make the task of understanding the conformance requirements easier and less prone to ambiguity.

7 **PICS proformas**

7.1 *Requirements on PICS proformas*

7.1.1 The specific requirements to be met by suppliers in respect of each PICS they provide shall normally be stated in the relevant protocol Recommendation*. The specification of these requirements shall include a PICS proforma. In exceptional circumstances, the PICS proforma may be found in the abstract test suite Recommendation* rather than in protocol Recommendation*; in particular, this applies when the PICS proforma has to cover different versions of the same protocol coming from both ISO and CCITT. In normal circumstances, the PICS proforma should be found in an annex to the protocol Recommendation* and referenced in the conformance section, and, if necessary, progressed as an addendum rather than as part of the original Recommendation*.

Note – A PICS for a specific protocol implementation will need to be accompanied by administrative and documentary information relating to the supplier, the system and its intended environment.

7.1.2 The PICS proforma shall be in the form of a questionnaire or checklist to be completed by the supplier or implementor of an implementation of the relevant OSI* protocol.

7.1.3 The PICS proforma shall cover all functions, elements of procedure, parameters, options, PDUs, timers, multi-layer dependencies and other capabilities identified in the protocol Recommendation*, including optional and conditional capabilities. It is desirable, where practicable, to include all mandatory features. There shall be a well-defined mapping between static conformance requirements and the PICS proforma and there shall be consistency between them.

7.1.4 The PICS proforma shall be preceded by a section that states:

“The supplier of a protocol implementation which is claimed to conform to this Recommendation shall complete the following Protocol Implementation Conformance Statements (PICS) proforma and shall provide the information necessary to identify uniquely both the supplier and the implementation.”

7.1.5 The name, version and date of the relevant protocol Recommendation* shall be stated on each page of the PICS proforma.

7.1.6 The PICS proforma for a specific protocol shall contain:

- a) explanations of special symbols, abbreviations, special terms, together with appropriate references;
- b) explicit instructions for completing and interpreting the PICS;
- c) provision for mentioning any mandatory feature that has not been implemented, with the appropriate rationale;
- d) one or more tables (or other kinds of form as necessary) to be completed to state the capabilities of the implementation in detail, including:
 - 1) name of the feature, PDU type, timer, parameter, and other capabilities;
 - 2) a column stating whether each is mandatory, optional, negotiable, or conditional;
 - 3) wherever feasible, a column giving references to the relevant sections in the Recommendation;
 - 4) a column giving the permitted range or values, if appropriate;
 - 5) a column to be filled in to give the supported values or range of values, if appropriate;
 - 6) a column to be filled in to state if each capability has been implemented;
- e) the proforma shall give a clear indication of the preferred data types (e.g. number bases, string types, octets, bits, seconds, minutes, etc.) for responses.

7.1.7 The PICS proforma shall use the following abbreviations as appropriate, unless they conflict with the abbreviations used in the specific protocol Recommendation*:

- m mandatory
- o optional
- c conditional
- n negotiable (using the protocol)
- x exclusion of capability
- not applicable
- s ability to send
- r ability to receive

7.2 *Guidance on PICS proformas*

Appendix III provides some general purpose examples to give guidance on the construction of PICS proformas.

8 Test suite production process

In order to present the requirements and general guidance for abstract test suite specification, it is useful to assume a normal form of the process of test suite production. This section describes the process in just such a normal form. Abstract test suite specifiers are not required to follow this normal form exactly, however, they are recommended to use a similar process involving the same steps, possibly in a different order.

For the purposes of this Recommendation, the abstract test suite production process is assumed to be as follows:

- a) study the relevant Recommendation(s)* to determine what the conformance requirements (including options) are which need to be tested, and what needs to be stated in the PICS (see § 9);
- b) decide on the test groups which will be needed to achieve the appropriate coverage of the conformance requirements and refine the test groups into sets of test purposes (see § 10);
- c) specify generic test cases for each test purpose, using some appropriate test notation (see § 11);
- d) choose the test method(s) for which the complete abstract test cases need to be specified, and decide what restrictions need to be placed on the capabilities of the lower tester and [if appropriate to the chosen test method(s)] the upper tester and test coordination procedures (see § 12);
- e) choose the test notation for specifying the complete abstract test cases, and specify the complete abstract test cases, including the test step structure to be used (see § 13);
- f) specify the inter-relationships among the test cases and those between the test cases and the PICS and, as far as possible, the PIXIT, in order to determine the restrictions on the selection and parameterization of test cases for execution, and on the order in which they can be executed (see § 14);
- g) consider the procedures for maintaining the abstract test suite (see § 15).

The remainder of this section provides requirements and guidance which relate to each step in the above process.

9 Determining the conformance requirements and PICS**9.1 Introduction**

Before an abstract test suite can be specified, the test suite specifier shall first determine what the conformance requirements are for the relevant Recommendation(s)* and what is stated in the PICS proforma concerning the implementation of those Recommendation(s)*.

9.2 Conformance requirements

Section 1 of this part of the Recommendation specifies the requirements to be met by protocol specifiers as a prerequisite to the production of an abstract test suite for a particular protocol.

In practice, early OSI* Recommendations* might not contain a clear specification of all the relevant conformance requirements. In particular, the static conformance requirements might be badly specified or even omitted. In such cases, the test suite specifier shall contribute to the production of an amendment or addendum to the relevant Recommendation(s)* to clarify the conformance requirements. If, however, an abstract test suite has to be produced in advance of the conformance requirements being clarified in the relevant Recommendation(s)*, then the test suite specifier shall adopt the short-term solution given in Annex C and state clearly in the test suite Recommendation* what the implications of this are (i.e. what is assumed to be mandatory, what conditional and what the conditions are, and what is assumed to be optional).

9.3 PICS proforma

If no PICS proforma is included in a relevant protocol Recommendation*, the test suite specifier shall provide a PICS proforma to be processed as an addendum to that Recommendation*, or in exceptional circumstances (as discussed in § 7.1.1) as an annex to the abstract test suite Recommendation*.

Note – Progressing an addendum to an existing protocol Recommendation* may well be faster than progressing the abstract test suite Recommendation* because the PICS proforma is likely to be less controversial than the test suite and therefore is likely to require fewer revisions before a final text can be agreed.

10 Test suite structure

10.1 Basic requirements

An abstract test suite shall comprise a number of test cases. The test cases shall be grouped into test groups, if necessary nested. The structure shall be hierarchical in the sense that an item at a lower level shall be completely contained within a higher level item. The structure need not, however, be strictly hierarchical in the sense that any one test case may occur in more than one test suite or test group. Similar test groups may occur in more than one higher level test group or test suite.

The abstract test suite specifier shall ensure that a subset of the test purposes of each abstract test suite is concerned with capability testing, and another subset is concerned with behaviour testing. This need not lead to distinct test cases for behaviour and capability testing because it may be possible to use the same test steps for both a behaviour test purpose and for a capability test purpose. The test suite specifier shall provide an explanation of how the test purposes are derived from or relate to the protocol Recommendation*. The test suite specifier shall also provide a summary of the coverage achieved by the test suite.

10.2 The test group structure

In order to ensure that the resulting abstract test suite provides adequate coverage of the relevant conformance requirements, the test suite specifier is advised to design the test suite structure in terms of nested test groups in a top down manner (see Figure 1/X.290, Part 2).

There are many ways of structuring the same test suite into test groups; no one way is necessarily right and the best approach for one test suite may not be appropriate for another test suite. Nevertheless, the test suite specifier shall ensure that the test suite includes test cases for whichever of the following categories is relevant:

- a) capability tests (for static conformance requirements);
- b) behaviour tests of valid behaviour;
- c) behaviour tests of syntactically invalid behaviour;
- d) behaviour tests of inopportune behaviour;
- e) tests focusing on PDUs sent to the IUT;
- f) tests focusing on PDUs received from the IUT;
- g) tests focusing on interactions between what is sent and received;
- h) tests related to each protocol mandatory feature;
- i) tests related to each optional feature which is implemented;
- j) tests related to each protocol phase;
- k) variations in the test event occurring in a particular state;
- l) timing and timer variations;
- m) PDU encoding variations;
- n) variations in values of individual parameters;
- o) variations in combinations of parameter values.

This list is not exhaustive; additional categories might be needed to ensure adequate coverage of the relevant conformance requirements for a specific test suite. Furthermore, these categories overlap one another and it is the task of the test suite specifier to put them into an appropriate hierarchical structure. The following structure is an example of a single-layer test suite, provided for guidance:

- A Capability tests
 - A.1 Mandatory features
 - A.2 Optional features said by the PICS to be supported
- B Behaviour tests: response to valid behaviour by peer implementation
 - B.1 Connection establishment phase (if relevant)
 - B.1.1 Focus on what is sent to the IUT
 - B.1.1.1 Test event variation in each state
 - B.1.1.2 Timing/timer variation
 - B.1.1.3 Encoding variation

- B.1.1.4 Individual parameter value variation
 - B.1.1.5 Combination of parameter values
 - B.1.2 Focus on what is received from the IUT
 - substructured as B.1.1
 - B.1.3 Focus on interactions
 - substructured as B.1.1
 - B.2 Data transfer phase
 - substructured as B.1
 - B.3 Connection release phase (if relevant)
 - substructured as B.1
- C Behaviour tests: response to syntactically invalid behaviour by peer implementation
 - C.1 Connection establishment phase (if relevant)
 - C.1.1 Focus on what is sent to the IUT
 - C.1.1.1 Test event variation in each state
 - C.1.1.2 Encoding variation of the invalid event
 - C.1.1.3 Individual invalid parameter value variation
 - C.1.1.4 Invalid parameter value combination variation
 - C.1.2 Focus on what the IUT is requested to send
 - C.1.2.1 Individual invalid parameter values
 - C.1.2.2 Invalid combinations of parameter values
 - C.2 Data transfer phase
 - substructured as C.1
 - C.3 Connection release phase (if relevant)
 - substructured as C.1
- D Behaviour tests: response to inopportune events by peer implementation
 - D.1 Connection establishment phase (if relevant)
 - D.1.1 Focus on what is sent to the IUT
 - D.1.1.1 Test event variation in each state
 - D.1.1.2 Timing/timer variation
 - D.1.1.3 Special encoding variations
 - D.1.1.4 Major individual parameter value variations
 - D.1.1.5 Variation in major combination of parameter values
 - D.1.2 Focus on what is requested to be sent by the IUT
 - substructured as D.1.1
 - D.2 Data transfer phase
 - substructured as D.1
 - D.3 Connection release phase (if relevant)
 - substructured as D.1

If the test suite is to cover more than one layer, then a single-layer test suite structure such as this could be replicated for each layer concerned. In addition, a correspondingly detailed structure could be produced for testing the capabilities and behaviour of multiple layers taken as a whole, including the interaction between the activities in adjacent layers.

10.3 *Test purposes*

The test suite specifier shall ensure that, for each test case in an abstract test suite, there is a clear statement of the test purpose. It is suggested that these test purposes should be produced as the next refinement of the test suite, after its structure in terms of test groups has been defined. The test purposes could be produced directly from studying those sections in the relevant Recommendation(s)* which are appropriate to the test group concerned. For some test groups, the test purposes might be derivable directly from the protocol state table; for others, they might be derivable from the PDU encoding definitions or the descriptions of particular parameters, or from text which specifies the relevant conformance requirements. Alternatively, the test suite specifier could employ a formal description of the protocol(s) concerned and derive test purposes from that by means of some automated method.

Whatever method is used to derive the test purposes, the test suite specifier shall ensure that they provide an adequate coverage of the conformance requirements of the Recommendation(s)* concerned. There shall be at least one test purpose related to each distinct conformance requirement.

In addition, it is possible to give guidance on the meaning of “adequate coverage” with reference to the above example. In order to express this, a shorthand notation will be used: the letter “x” will represent all appropriate values for the first digit in the test group identifier, and similarly “y” for the second digit, so that B.x.y.1 stands for B.1.1.1, B.1.2.1, B.1.3.1, B.2.1.1, B.2.2.1, B.2.3.1, B.3.1.1, B.3.2.1 and B.3.3.1. With this notation, a minimum “adequate coverage” for the above example is considered to be as follows:

- a) for capability test groups (A.1, A.2):
 - 1) at least one test purpose per relevant feature, class or subset;
 - 2) at least one test purpose per relevant PDU type and each major variation of each such type, using “normal” or default values for each parameter;
- b) for test groups concerned with test event variation in each state (B.x.y.1, C.x.1.1, D.x.y.1):
 - at least one test purpose per relevant state/event combination;
- c) for test groups concerned with timers and timing (B.x.y.2, D.x.y.2):
 - 1) at least one test purpose concerned with the expiry of each defined timer;
 - 2) at least one test purpose concerned with very rapid response for each relevant type of PDU;
 - 3) at least one test purpose concerned with very slow response for each relevant type of PDU;
- d) for test groups concerned with encoding variations (B.x.y.3, C.x.1.2, D.x.y.3):
 - at least one test purpose for each relevant kind of encoding variation per relevant PDU type;
- e) for test groups concerned with valid individual parameter values (B.x.y.4, D.x.y.4):
 - 1) for each relevant integer parameter, test purposes concerned with the boundary values and one randomly selected mid-range value;
 - 2) for each relevant bitwise parameter, test purposes for as many values as practical, but not less than all the “normal” or common values;
 - 3) for other relevant parameters, at least one test purpose concerned with a value different from what is considered “normal” or default in other test groups;
- f) for test groups concerned with invalid individual parameter values (C.x.1.3, C.x.2.1):
 - 1) for each relevant integer parameter, test purposes concerned with invalid values adjacent to the allowed boundary values plus one other randomly selected invalid value;
 - 2) for each relevant bitwise parameter, test purposes for as many invalid values as practical;
 - 3) for all other relevant types of parameter, at least one test purpose per parameter;
- g) for test groups concerned with combinations of parameter values (B.x.y.5, C.x.1.4, C.x.2.2, D.x.y.5):
 - 1) at least one test purpose per “critical” value pair;
 - 2) at least one test purpose per pair of interrelated parameters to test a random combination of relevant values.

11 **Generic test case specification**

11.1 *Introduction*

The test suite specifier is recommended to specify a generic test suite, particularly if there is an intention to produce more than one abstract test suite.

A generic test suite shall consist of one generic test case for each test purpose. Each generic test case will be a refinement of its test purpose, which can be used as a common root of corresponding abstract test cases for different test methods.

If a generic test suite is produced in advance of abstract test suites, then it will be a useful step in the design process. If the generic test suite is produced after the production of at least one abstract test suite, then it will provide a means of relating different test suites to one another and analyzing where there may be gaps in their coverage.

11.2 *Description of generic test cases*

A generic test case consists of a test description of an “initial state” [of the test body] and a specification of the test body in a standardized test notation. The “initial state” includes not only the protocol state, but also any necessary information concerning the state of the SUT and the testing environment.

The test body is that part of a test case in which verdicts related to the test purpose are assigned to foreseen outcomes. The test body:

- a) shall be defined in the style of either the Remote or Distributed test method;
Note – Generic test cases may use the full expressive power of these methods (see § 12.3.3 for details), including the use of abstract local primitives.
- b) shall assign verdicts to all possible outcomes; all outcomes yielding “pass” verdicts shall be explicitly identified, whereas all outcomes yielding “fail” or “inconclusive” verdicts shall be either identified or categorized (which may include categorization of default verdicts);
- c) shall be described using a standardized test notation; the Tree and Tabular Combined Notation (TTCN) (defined in Annex D) is recommended.

11.3 *Relation of generic to abstract test cases*

For a particular abstract test method there may be many abstract test cases that can be derived from a single generic test case. A major difference between an abstract test case and a generic test case is that the abstract test case includes specifications of a preamble and a postamble. The preamble starts from a chosen stable state and leads to the required initial state of the test body. The postamble starts from the end of the test body and returns to a chosen stable state.

The preamble and postamble may be realized in different ways depending on the degree of control and observation provided by the test method used, or on the variety of different possible stable states which the derived abstract test case can start from and end in. These abstract test cases are simply different ways of achieving the same test purpose.

In addition, the test body of an abstract test case may be different from the corresponding generic test case if the test method used for the abstract test case is different from that used for the generic test case.

If a generic test suite is produced, it shall be used as the means of relating corresponding abstract test suites for different abstract test methods.

12 Abstract test methods

12.1 *Introduction*

Each abstract test suite shall be specified in terms of the control and observation of events in accordance with one of the abstract test methods defined in this section. The chosen test method determines the points at which control and observation shall be specified and the categories of events which shall be used [e.g. (N – 1)-ASPs, (N)-PDUs].

12.2 *General specification of the abstract test methods*

12.2.1 *End-system IUTs*

For IUTs within end-system SUTs there are four categories of abstract test methods, one local, and three external ones which assume the lower tester is located remotely from the SUT and connected to it by a link or network.

For generality, the test methods are described with reference to an IUT in which the highest layer is numbered “N_t” (for “top”) and in which the lowest layer protocol to be tested is numbered “N_b” (for “bottom”). The IUT may implement protocols in layers lower than “N_b”, but these are not of interest in the test method descriptions. The description applies to single-layer IUTs by taking N_t to be equal to N_b.

12.2.2 *Abstract local primitives*

Abstract test suite specifiers may, if necessary, define a set of abstract local primitives (ALPs) to be used in specifying the test suite. An ALP is an abbreviation for a description of control and/or observation to be performed by the upper tester, which cannot be described in terms of ASPs but which initiate events or state changes defined within the relevant protocol Recommendation(s)*. ALPs may be used with any abstract test method for end-system IUTs, but, for simplicity, will not be shown in any of the figures which illustrate those methods.

Any test case which uses an ALP shall be optional in the abstract test suite.

12.2.3 *The local test methods*

Abbreviation: L

In this method:

- a) the abstract test suite is specified in terms of control and observation of $(N_b - 1)$ -ASPs and (N_b) to (N_i) -PDUs;
- b) the abstract test suite is also specified in terms of control and observation of (N_i) -ASPs;
- c) the abstract test suite may also be specified in terms of control and observation by the upper tester of abstract local primitives (ALPs);
- d) the method requires access to the lower and upper boundaries of the IUT and a mapping between the specified ASPs and their realization within the SUT;
- e) the requirements for the test coordination procedures are specified in the abstract test suite but no assumption is made regarding their realization;
- f) the upper and lower testers are required to achieve control and observation of the specified ASPs and the required test coordination procedures. They are assumed to be integrated into the SUT.

This method is illustrated in Figure 1/X.290, Part 2.

12.2.4 *External test methods*

12.2.4.1 *The distributed test method*

Abbreviation: D

In this method:

- a) the abstract test suite is specified in terms of control and observation of $(N_b - 1)$ -ASP's and (N_b) to (N_i) -PDUs;
- b) the abstract test suite is also specified in terms of control and observation of (N_i) -ASPs; the method requires access to the upper boundary of the IUT and a mapping between the (N_i) -ASPs and their realization within the SUT;
- c) the abstract test suite may also be specified in terms of control and observation by the upper tester of ALPs;
- d) the requirements for the test coordination procedures are specified in the abstract test suite but no assumption is made regarding their realization;
- e) the upper tester is required to achieve control and observation of the specified (N_i) -ASPs and to achieve the effects of the required test coordination procedures; no other assumptions are made;
- f) the lower tester is required to achieve control and observation of specified (N_b) -ASP's and specified PDUs and to achieve the required test coordination procedures.

This method is illustrated in Figure 2/X.290, Part 2.

12.2.4.2 *The coordinated test method*

Abbreviation: C

In this method:

- a) the abstract test suite is specified in terms of control and observation of $(N_b - 1)$ -ASP's, (N_b) to (N_i) -PDUs and test management PDUs (TM-PDUs);
- b) (N_i) -ASPs need not be used in the specification of the abstract test suite; no assumption is made about the existence of an upper boundary of the IUT;
- c) the requirements for the test coordination procedures are specified in the abstract test suite by means of a standardized test management protocol;

- d) TM-PDUs may be defined which correspond to ALPs;
- e) the upper tester is required to implement the test management protocol and achieve the appropriate effects on the IUT;
- f) the lower tester is required to achieve control and observation of specified (N_b)-ASP's and specified PDUs (including TM-PDUs).

This method is illustrated in Figure 3/X.290, Part 2.

12.2.4.3 *The remote test method*

Abbreviation: R

In this method, provision is made for the case where it is not possible to observe and control the upper boundary of the IUT. Also in this method:

- a) the abstract test suite is specified in terms of control and observation of ($N_b - 1$)-ASP's and (N_b) to (N_i)-PDUs ;
- b) (N_i)-ASPs are not used in the specification of the abstract test suite; no assumption is made about the existence of an upper boundary of the IUT;
- c) the abstract test suite may also be described in terms of control and observation of ALPs within the SUT;
- d) some requirements for test coordination procedures may be implied or informally expressed in the abstract test suite but no assumption is made regarding their feasibility or realization;
- e) abstractly the SUT needs to carry out some upper tester functions to achieve whatever effects of test coordination procedures and whatever control and/or observation of the IUT are implied, informally expressed or described by ALPs in the abstract test suite for a given protocol; these functions are not specified nor are any assumptions made regarding their feasibility or realization;
- f) the lower tester is required to achieve control and observation of specified (N_b)-ASP's and specified PDUs and should attempt to achieve the implied or informally expressed test coordination procedures in accordance with the relevant information in the PIXIT.

This method is illustrated in Figure 4/X.290, Part 2.

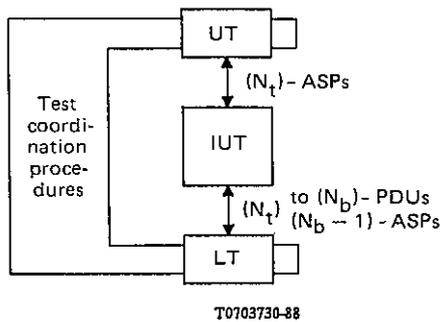


FIGURE 1/X.290, Part 2
The local test method

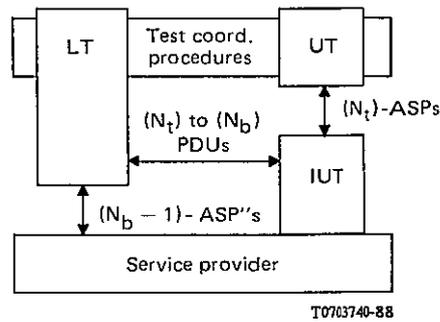


FIGURE 2/X.290, Part 2
The distributed test method

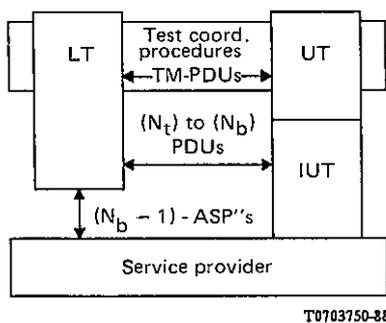


FIGURE 3/X.290, Part 2
The coordinated test method

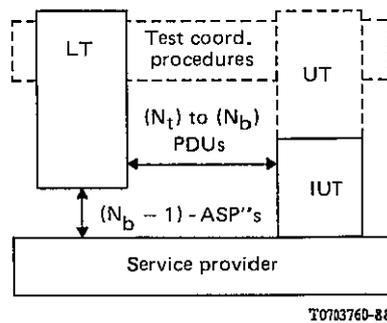


FIGURE 4/X.290, Part 2
The remote test method

12.2.5 Single-layer, multi-layer and embedded variants

Each category of test methods has a variant which can be applied to single-layer IUTs (abbreviation: S), and another which can be applied to multi-layer IUTs (abbreviation: M), when the set of adjacent layers is to be tested in combination (as a whole).

For a multi-layer IUT in which the protocols are to be tested layer by layer, an embedded variant of the test methods has been defined (abbreviation: E).

If control and observation are applied to a means of access to the upper boundary of the entities under test within SUT, then the test methods are normal (and no E is added to the abbreviated name). If, however, control and observation are applied through one or more OSI* layer entities above the entities under test, the test methods are called embedded (and an E is appended to the abbreviated name).

Names of particular variants of the test methods shall be formed as follows:

$$\begin{array}{|c|} \hline L \\ \hline R \\ \hline C \\ \hline D \\ \hline \end{array} \quad \begin{array}{|c|} \hline S \\ \hline M \\ \hline \end{array} \quad [E]$$

For example, DSE is the abbreviation for the “distributed single embedded” test method.

12.2.6 Open relay-systems

For open relay-systems, loop-back and transverse abstract test methods are defined. They are given the abbreviated names: YL and YT, respectively.

12.3 Single-layer test methods for single-layer IUTs in end-systems

For single-layer test methods, the abstract model of the IUT is called the (N)-entity under test.

12.3.1 The local single-layer test method

The Local Single-layer (LS) abstract test method defines the points of control and observation as being at the service boundaries above and below the (N)-entity under test. The test events are specified in terms of (N)-ASP's above the IUT, and (N – 1)-ASP's and (N)-PDUs below the IUT, as shown in Figure 5/X.290, Part 2. In addition, ALPs may be used as test events. Abstractly, a lower tester is considered to observe and control the (N – 1)-ASP's and (N)-PDUs while an upper tester observes and controls the (N)-ASP's and ALPs. The requirements to be met by test coordination procedures used to coordinate the realizations of the upper and lower testers are defined in the abstract test suites, although the test coordination procedures themselves are not.

12.3.2 The distributed single-layer test method

The Distributed Single-layer (DS) abstract test method defines the points of control and observation as being at the service boundaries above the (N)-entity under test and above the (N – 1)-Service Provider at the SAP remote from the (N)-entity under test. The test events are specified in terms of (N)-ASP's above the IUT and (N – 1)-ASP's and (N)-PDUs remotely, as shown in Figure 6/X.290, Part 2. In addition, ALPs may be used as test events. Abstractly, lower and upper testers are again considered to observe and control the behaviour at the respective points. The requirements to be met by the test coordination procedures are again defined in the abstract test suites, although the procedures themselves are not.

For lower layers (1-3) where it may be unrealistic to specify observation and control of (N – 1)-ASP's, the lower tester observation and control shall be specified in terms of (N)-PDUs and, when necessary, changes in the state of the underlying connection.

Note – For example, the state of the underlying connection could be changed by setting up a new connection, or resetting or closing an existing connection.

The observation and control to be performed by the lower tester can optionally be specified in terms of (N)-ASP's where this will reduce the size of the test case specification without loss of required precision.

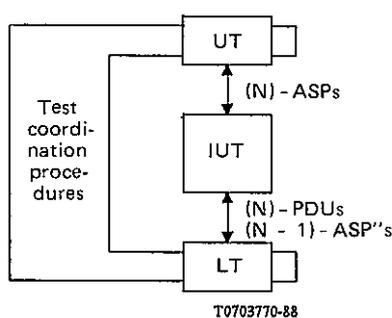


FIGURE 5/X.290, Part 2
The LS test method

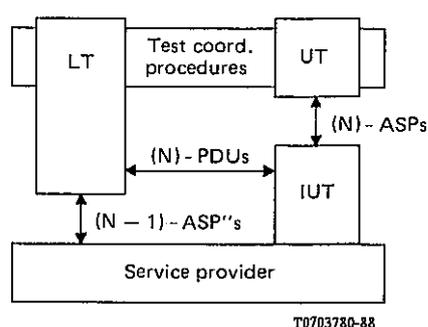


FIGURE 6/X.290, Part 2
The DS test method

12.3.3 The coordinated single-layer test method

The Coordinated Single-layer (CS) abstract test method is an enhanced version of the DS method, using a standardized upper tester and the definition of a test management protocol to realize the test coordination procedures between the upper and lower testers. The same standardized upper tester and test management protocol are not necessarily applicable to all test suites which use the coordinated method.

Standardized upper testers and test management protocols are applicable to a particular standardized abstract test suite for the coordinated test method and may not be applicable to other abstract test suites for the coordinated method.

There is only a single point of control and observation, above the (N – 1)-Service Provider at the SAP remote from the (N)-entity under test. Test events are specified in terms of (N – 1)-ASP's, (N)-PDUs and TM-PDUs, as shown in Figure 7/X.290, Part 2.

For lower layers (1-3) where it may be unrealistic to specify observation and control of (N – 1)-ASP's, the observation and control shall be specified in terms of TM-PDUs, (N)-PDUs and, when necessary, changes in the state of the underlying connection.

Concerning the test management protocol:

- a) the test management protocol shall be implemented within the SUT directly above the abstract service boundary at the top of the IUT;
- b) the IUT shall not be required to interpret TM-PDUs, only pass them to and from the upper tester;
- c) a test management protocol is only defined for testing a particular protocol and so does not need to be independent of the underlying protocol;
- d) verdicts on test cases shall not be based on the ability of the SUT to exhibit any ASP or parameter of an ASP at the upper service boundary of the IUT, since this would contradict the definition of the coordinated method in that the upper service boundary of the IUT is not a point of control and observation in this method. However, it is recommended that the test management protocol be defined separately from the abstract test suite(s) in order to facilitate the task of the implementor of an upper tester. This definition (as with the definition of any OSI* protocol defined by ISO or CCITT) can refer to the ASPs of its underlying service (i.e. the ASPs at the upper service boundary of the IUT);
- e) TM-PDUs which correspond to ALPs are optional and there is no requirement for the upper tester to support them.

12.3.4 *The remote single-layer test method*

The Remote Single-layer (RS) abstract test method defines the point of control and observation as being above the (N – 1)-Service Provider at the SAP remote from the (N)-entity under test. The test events are specified in terms of the (N – 1)-ASP's and (N)-PDUs remotely, as shown in Figure 8/X.290, Part 2. In addition, ALPs may be used as test events. Some requirements for test coordination procedures may be implied or informally expressed in the abstract test suites, but no assumptions shall be made regarding their feasibility or realization.

For the lower layers (1-3) where it is unrealistic to specify observation and control of (N – 1)-ASP's, the observation and control shall be specified in terms of (N)-PDUs and when necessary the state of the underlying connection.

In addition, in order to overcome the lack of specification of behaviour above the (N)-entity under test, where necessary the required behaviour of the system under test shall be specified in terms of the (N – 1)-ASP's or (N)-PDUs which need to be observed by the lower tester. This form of implicit specification shall be taken to mean "do whatever is necessary within the system under test in order to provoke this required behaviour".

Note – Such implicit specification is necessary with this test method for any test case which requires an IUT initiated event which cannot be initiated by means of an ALP. Since ALPs may only be defined if the same effect cannot be achieved by ASPs, then any PDU which can be initiated by an ASP needs implicit specification to allow it to be initiated in this test method.

However, it is possible that some of the test cases in the abstract test suite cannot be executed (e.g. transmission and maintenance of busy conditions, transmission of consecutive unacknowledged Data PDUs, etc.). In such instances, it is left to the test laboratory and client to negotiate the method by which these tests can be accomplished.

Even with such implicit specification of control of the IUT, in this method it is possible to specify control but not observation above the IUT, except through the use of ALPs. This is a major difference between this and the DS and CS test methods.

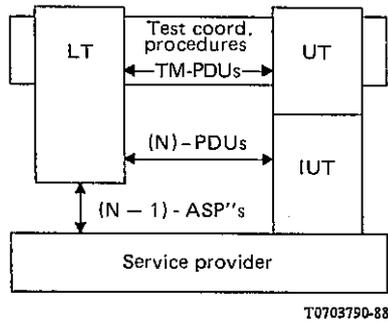


FIGURE 7/X.290, Part 2
The CS test method

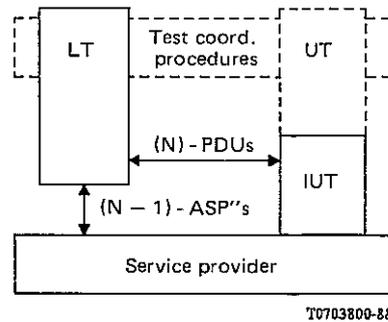


FIGURE 8/X.290, Part 2
The RS test method

12.4 Multi-layer test methods for multi-layer IUTs (LM, DM, CM, RM)

Multi-layer testing, when the combined allowed behaviour of the multi-layer implementation is known, involves testing all the layers of a multi-layer IUT as a whole, without controlling or observing any of the inter-layer boundaries within the IUT.

In the Local Multi-layer (LM) test method the points of observation and control are the service boundaries directly above and below the IUT. The test events shall be specified in terms of the (N_i) -ASPs and ALPs above the IUT and the $(N-1)$ -ASPs and (N) to (N_i) -PDUs below the IUT.

In the Distributed Multi-layer (DM) test method the points of observation and control are at the service boundary above the IUT and above the $(N-1)$ -Service Provider at the SAP remote from the IUT. The test events shall be specified in terms of the (N_i) -ASPs and ALPs above the IUT and the $(N-1)$ -ASP's and (N) to (N_i) -PDUs remotely.

In the Coordinated Multi-layer (CM) test method the point of observation and control is above the $(N-1)$ -Service Provider at the SAP remote from the IUT. The test events shall be specified in terms of the $(N-1)$ -ASP's, the (N) to (N_i) -PDUs and the TM-PDUs. The test management protocol shall be designed to operate over the (N_i) -Service, where (N_i) is the highest layer in the IUT.

In the Remote Multi-layer (RM) test method the point of observation and control is above the $(N-1)$ -Service Provider at the SAP remote from the IUT. The test events shall be specified in terms of the $(N-1)$ -ASP's and the (N) to (N_i) -PDUs, ALPs and the implicit specification of the control of the SUT behaviour when necessary. Some requirements for test coordination procedures may be implied or informally expressed, but no assumptions shall be made regarding their feasibility or realization.

12.5 Single-layer testing of multi-layer IUTs or SUTs (Embedded methods)

In embedded single-layer test methods testing is specified for a single-layer within a multi-layer IUT, including the specification of the protocol activity in the layers above the one being tested but without specifying control or observation at service boundaries within the multi-layer IUT. Thus in a multi-layer IUT from layer (N) to (N_i) , abstract test cases for testing layer (N_i) shall include the specification of the PDUs in layers (N_i+1) to (N_i) as well as those in layer (N_i) .

The Local Single-layer Embedded (LSE) test method uses the same points of control and observation as the LM test method for the same set of layers. The test events shall also be specified in the same terms as for the LM test method. The difference is that the LSE test method concentrates on a single-layer at a time, whereas the LM test method tests the multi-layer IUT as a whole.

In the Distributed Single-layer Embedded (DSE) test method for layer (N_i) within a multi-layer IUT from layer (N) to (N_i) the points of observation and control are at the service boundary above the IUT and above the (N_i-1) -Service Provider at the SAP remote from the IUT, as illustrated in Figure 9a)/X.290, Part 2. The test events shall be specified in terms of (N_i) -ASPs and ALPs above the IUT and (N_i-1) -ASP's and (N_i) to (N_i) -PDUs remotely.

Note – For the top layer in the multi-layer IUT, (N_i) , this method is the same as the DS test method.

The Coordinated Single-layer Embedded (CSE) test method uses features of both the CM and DSE test methods. The test events shall be specified in terms of (N_i-1) -ASP's, (N_i) to (N_i) -PDUs and TM-PDUs and the test management protocol shall be designed to operate over the (N_i) -Service. This is illustrated in Figure 9b)/X.290, Part 2.

The Remote Single-layer Embedded (RSE) test method uses the same point of control and observation as the RS test method for the same layer, but differs from the RS test method in that (N_i+1) to (N_i) -PDUs shall be specified in test cases for layer (N_i) .

Successive use of a single-layer embedded test method (from layer (N) to (N_i)) is called incremental testing of a multi-layer IUT.

The DSE/CSE/RSE methods are defined for a single layer under test in a multi-layer IUT. This does not mean that there cannot be accessible service boundaries within the multi-layer IUT, just that no such boundaries are used in the test methods. Thus, all layers between the layer under test and the highest layer for which PDUs are expressed as test events in the abstract test suite shall be regarded as being part of the multi-layer IUT.

Note – DME/CME/RME test methods could theoretically be defined to test multiple layers as a whole within a larger multi-layer IUT, but in order to avoid excess complexity, they are not detailed in this Recommendation.

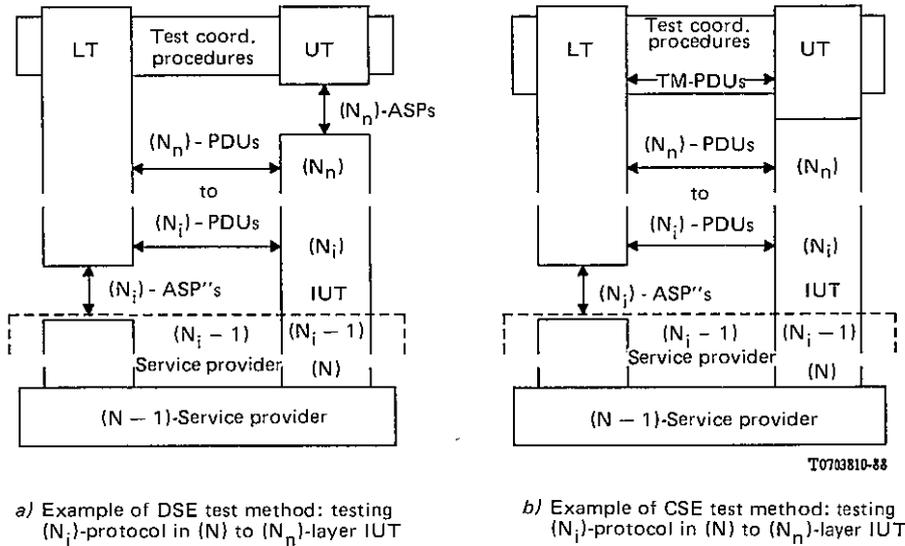


FIGURE 9/X.290, Part 2
The embedded test methods

12.6 Relay test methods

There are two abstract test methods for relay system testing:

- a) the loop-back test method (YL): used for testing a relay system from one subnetwork.

This test method is illustrated in Figure 10/X.290, Part 2.

For this test method there are two points of observation and control on one subnetwork at SAPs remote from the (N) -Relay. For connection-oriented protocols, it requires that the two test connections are looped together on the far side of the (N) -Relay, but it is not specified whether this looping is performed within the (N) -Relay or in the second subnetwork. For connectionless protocols, it requires that the PDUs are looped back within the second subnetwork and addressed to return to the second point of observation and control.

- b) the transverse test method (YT): used for testing a relay system from two subnetworks.

This test method is illustrated in Figure 11/X.290, Part 2.

In this test method there are two points of observation and control, one on each subnetwork, at SAPs remote from the (N) -Relay.

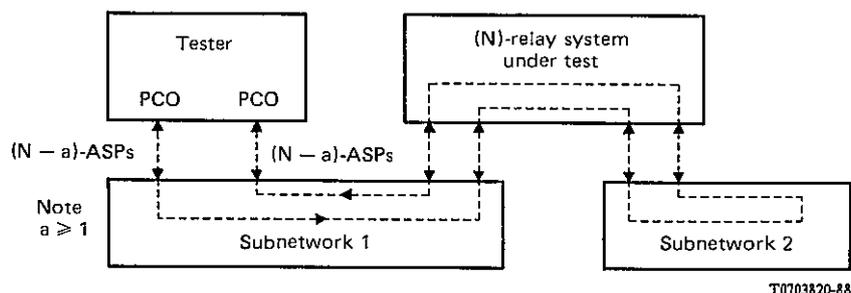


FIGURE 10/X.290, Part 2
Loop-back test method (YL)

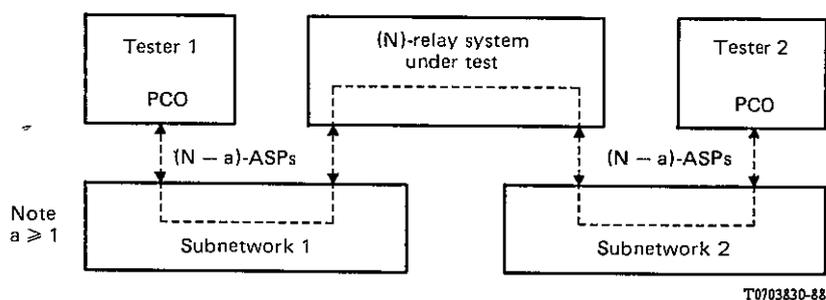


FIGURE 11/X.290, Part 2
Transverse test method (YT)

12.7 Choice of test method

12.7.1 Introduction

Before an abstract test suite can be defined, it is necessary to study all the environments in which the protocol is likely to be tested and establish accordingly the abstract test method(s) to be used for the production of one or more abstract test suites.

Abstract test suite specifiers shall place a requirement in the abstract test suite Recommendation* defining which abstract test method(s) shall be supported as a minimum by any organization claiming to provide a comprehensive testing service for the protocol(s) in question.

12.7.2 IUT environments

There is a relation between the test methods and the configurations of the real open systems to be tested.

Section 7.2 Part 1 of this Recommendation gives a full account of the classification of systems and IUTs.

When choosing a test method, the test suite specifiers should identify, if they have not already done so, whether the test suites they plan to produce are for IUTs which

- are single or multi-layer;
- belong to end or relay systems;
- belong to complete or partial systems;
- belong to fully open or mixed systems;
- have service boundaries accessible or not;
- are special purpose (i.e. to be used by a single application) or general purpose (i.e. to be used by several applications).

12.7.3 *Applicability of the abstract test methods*

The possibility of developing an abstract test suite for a given abstract test method will depend on the protocol(s) being considered, together with the results of the study described in § 12.7.2. This applies to the possibility of developing test suites for a given combination of methods (e.g. incremental) or a given variant of a method (e.g. embedded).

Some considerations concerning the applicability of the methods to different layers are discussed in Appendix I of Part 1 of this Recommendation.

One or more appropriate abstract test methods shall be selected for the protocol being considered.

Priorities should be assigned to the various test methods which have been identified, according to the configurations which are most likely to be found in real systems.

12.7.4 *Illustrative examples*

Appendix III provides an illustrative example of the choice of abstract test methods for given protocols.

12.8 *Test coordination procedures*

For effective and reliable execution of conformance tests, some set of rules is required for the coordination of the test process between the lower tester and the upper tester. The general objective of these rules is to enable the lower tester to control remotely the operation of the upper tester or vice versa, in ways necessary to run the test suite selected for the IUT.

These rules lead to the development of test coordination procedures to achieve the synchronization between the lower tester and the upper tester and the management of information exchanged during the testing process. The details and how these effects are achieved are closely related to the characteristics of the SUT, as well as the external test methods.

For each abstract test suite using the coordinated, distributed or local test methods, a standard set of test coordination procedures should be developed. This is because those procedures depend on the functionality of the upper tester and definitions of test cases.

In the case of the coordinated test methods (CS, CSE, CM) the test coordination procedures are realized by the standardization of a test management protocol. The test management protocol needs to be able to convey requests to the IUT to achieve the effect of a service primitive or ALP and to convey back to the lower tester the record of observations of effects equivalent to the occurrence of service primitives or ALPs. The upper tester should be an implementation of the test management protocol. It will be necessary to add test cases to the abstract test suite for the purpose of testing that the upper tester conforms to the requirements of the test management protocol specification. Such test cases do not contribute to the conformance assessment of the IUT.

When defining test cases for the local and distributed test methods, the test suite specifier shall record any constraints on the upper tester and/or test coordination procedures which may be necessary.

13 Specification of abstract test suites

13.1 *Test cases*

13.1.1 The abstract test suite specifier shall select an appropriate standardized notation in which to define the abstract test cases. The Tree and Tabular Combined Notation (TTCN), defined in Annex D, is defined for this purpose.

13.1.2 Once the test notation and test method have been chosen, then the generic test cases can be expanded into abstract test cases. There are two main kinds of change required to convert a generic test case into an abstract test case. The first is to express the test body in terms of control and observation required by the test method, and, if relevant, include a description of the synchronization needed between upper and lower testers. The second kind of change is to specify the preamble and postamble.

13.1.3 Specification of preambles and postambles may result in more than one test step for each. The preamble shall start in a stable state and progress to the desired state. Conversely, the postamble shall progress from the final state of the test body to a stable state. A small number of stable states shall be defined for the protocol concerned, always containing as a minimum the “idle” state. Each abstract test case shall be capable of being run on its own and shall therefore include test steps to start the preamble from the “idle” state and to end the postamble in the “idle” state.

However, other stable starting and final states for an abstract test case can be useful when efficient concatenation of abstract test cases is required.

Furthermore, in designing the test step structure within the abstract test cases, the test suite specifier can benefit from using the same test steps in several abstract test cases.

13.1.4 In converting from generic test cases to abstract test cases, the test suite specifier shall ensure that the initial state for the test body is preserved, the pass paths through the test body are preserved and the assignment of verdicts to outcomes remains consistent.

In order to maintain consistency, the following conditional requirements apply when assigning verdicts to outcomes:

- a) if the behaviour of the preamble and the postamble are valid then the verdict assigned to a particular outcome shall be the same as that assigned to the corresponding outcome in the generic test case;
- b) if the preamble results in the initial state of the test body not being reached, although there is no invalid behaviour, then the verdict shall be inconclusive;
- c) if the preamble results in the initial state of the test body not being reached, because of invalid behaviour, then the verdict shall be “fail but test purpose inconclusive” (“fail type 3”);
- d) if the postamble exhibits invalid behaviour and the generic test case (or test body) verdict is “pass” or “inconclusive”, then the verdict shall be “fail but test purpose passed” (“fail type 2”) or “fail but test purpose inconclusive” (“fail type 3”), respectively;
- e) if the generic test case (or test body) verdict is “fail” then invalid behaviour in the postamble shall not change the verdict (“fail type 1”).

13.1.5 The test suite specifier shall also ensure that each abstract test case explicitly identifies all the outcomes assigned the verdict “pass” and identifies or categorizes all the remaining foreseen outcomes, assigning to each individual outcome or category a verdict “fail” or “inconclusive”. All unforeseen outcomes in the test body shall be assigned either:

- a) the verdict “fail”; or
- b) the verdict “inconclusive”.

The test suite specifier shall ensure that the application of a) or b) is consistent throughout the abstract test suite. If a) is chosen, then any unforeseen outcome in the preamble shall be assigned the verdict “fail but test purpose inconclusive” (“fail type 3”), and any unforeseen outcome in the postamble shall be treated as a protocol violation, leading to the appropriate type of fail verdict.

If b) is chosen, then any unforeseen outcome in the preamble shall be assigned the verdict “fail but test purpose inconclusive” (“fail type 3”), and any unforeseen outcome in the postamble shall be assigned the appropriate type of fail verdict.

13.2 *Test suites*

An abstract test suite specification comprises a set of test cases and test steps. Preceding the test cases themselves shall be the following information:

- a) abstract test suite name, date of origin and version number;
- b) names (and version numbers) of the protocol Recommendation(s)* for which test cases are provided;
- c) names (and version numbers) of the service Recommendation(s)* whose primitives are specified as being observed;
- d) name (and version number) of the Recommendation* defining the test notation, or a reference to an annex for the same if it is not standardized elsewhere;
- e) name of target test method;
- f) description of the coverage of the test suite; for example, the functional subsets of the protocol Recommendation(s)* that are tested;
- g) description of the structure of the test suite, in terms of test groups and their relationship to the protocol Recommendation(s)*;
- h) description of the test coordination procedures (if applicable in the test method);
- i) statement of which test cases are optional and which mandatory for conformance to the abstract test suite Recommendation*;
- j) information to assist the test realizer and test laboratory in their use of the abstract test suite Recommendation* (see § 14).

14 Use of an abstract test suite specification

The abstract test suite specifier shall provide information in the abstract test suite Recommendation* to assist the test realizer and test laboratory in their uses of the test suite. This information shall include, but is not limited to, the following:

- a) a mapping of the abstract test cases to the PICS proforma entries to determine whether or not each abstract test case is to be selected for the particular IUT; the mapping should be specified in a suitable notation for Boolean expressions;
- b) a mapping of the abstract test cases to the PIXIT proforma entries, to the extent that they are known by the abstract test suite specifier, to parameterize the test suite for the particular IUT and to determine which selected test cases cannot be run with the particular IUT.

The test suite specifier shall define a partial PIXIT proforma. This shall contain a list of all the ALPs used in the test suite (or test management protocol) and a list of all parameters for which the test suite requires values. If any of the required parameter values will be found in the PICS, the PIXIT proforma entry for each such parameter shall reference the corresponding entry in the PICS proforma;

Note – Other aspects of the PIXIT proforma are for further study;

- c) a list of the abstract test cases in the order that shall be used in the Protocol Conformance Test Report (PCTR), together with any information which shall be preserved in the PCTR on the status of each test case; this is a contribution to the development of a PCTR proforma;
- d) any restrictions that there may be on the order in which the test cases may be executed;
- e) reference to the description of the test coordination procedures (if applicable in the chosen test method);
- f) any necessary timing information.

15 Test suite maintenance

Once an abstract test suite has been specified and is in use, it can be expected that errors and omissions in it will be detected by those who are using the test suite. The abstract test suite specifier shall in such circumstances progress the updating of the test suite via the relevant rapid amendment procedures.

In addition, from time-to-time, changes will be made to the protocol Recommendation(s)* to which the test suite relates. The abstract test suite specifier shall ensure that the test suite is updated as soon as possible after changes to the relevant protocol Recommendation* have been ratified.

ANNEX A

(to Recommendation X.290, Part 1)

Options

A.1 Options are those items in a Recommendation* for which the implementor may make a choice regarding the item to suit the implementation.

A.2 Such a choice is not truly free. There are requirements which specify the conditions under which the option applies and the limitations of the choice.

Conversely, there may be mandatory or conditional requirements, or prohibitions, in a Recommendation* which are dependent on the choice made or on a combination of the choices already made.

A.3 The following are examples of options and associated requirements; the list is not exhaustive:

- a) “Boolean” options: the option is “do or do not do”; the requirement is “if do, then do as specified.”
- b) Mutually exclusive options: the requirement is to do just one of n actions, the option is which of them to do.
- c) Selectable options: the option is to do any m out of n actions, with a requirement to do at least one action ($1 \leq m \leq n$ and $n \geq 2$).

A.4 Options may apply to anything within the scope of a Recommendation* (e.g. static or dynamic aspects, use or provision of a service, actions to be taken, presence/absence or form of parameters, etc.).

A.5 Another type of distinction is between service-user options and service-provider options.

A.6 In a wider context, the choice will be determined by conditions which lie outside the scope of the Recommendation* (e.g. other Recommendations* which apply to the implementation, the protocols used in the (N + 1) and (N – 1) layers, the intended application, conditions of procurement, target price for the implementation, etc.). However, these have no bearing on conformance to the Recommendation* in which the option appears.

ANNEX B

(to Recommendation X.290, Part 2)

Guidance for protocol Recommendations* writers to facilitate conformance testing

B.1 Introduction

This Annex gives guidance, primarily for the specifiers of new protocol Recommendations*, to facilitate conformance testing by ensuring a very clear understanding of the conformance requirements.

B.2 Guidance on scope and field of application

B.2.1 Precision in the scope and field of application sections sets the tone for precision in the rest of the Recommendation*. The requirements stated in the Recommendation* should be consistent with the scope and field of application and *vice versa*.

B.2.2 The scope should distinguish clearly between the following three types of information included in the protocol Recommendation*:

- a) the definition of the procedures for communication to be followed at the time of communication;
- b) requirements to be met by suppliers of implementations of the procedures;
- c) guidance on how to implement the procedures.

Guidance on how to implement the procedures does not constitute additional requirements nor does it have any bearing on conformance. If such guidance is included, the scope should make these points and indicate how guidance can be distinguished from the requirements of the Recommendation*. This distinction is much easier to make if guidance is separated from requirements. The recommended method of such separation in the ISO Directives is to place guidance in notes and annexes.

B.2.3 It should be clear to whom the Recommendation* applies.

B.2.4 It should be clear at what time the Recommendation* applies.

Protocol procedures apply between pairs of communicating parties at the time of communication. If there might be any ambiguity over which communicating parties are involved, this should be resolved in the scope.

It is best if protocol Recommendations* are written in such a way that the requirements are to be met by a single communicating party (the 'first' communicating party for this purpose) for the benefit of one or more other communicating parties (the 'second' communicating parties). Then when two (or more) communicating parties are all expected to communicate in conformance with the Recommendation*, the Recommendation* is first applied to one party, treating it as the 'first' one, and then to the other(s) in turn. This ensures that if the procedures are violated, it is clear which party is at fault.

B.2.5 If any guidance is given about factors which are not standardized definitively, the scope should make it clear that any such guidance can be ignored without affecting conformance.

B.2.6 The aspects which are excluded from the scope should be identified clearly.

Not all factors relevant to the procedures or to products which implement them need to be standardized; indeed it is often desirable to leave some implementor freedom. For instance, it may be desirable to omit in a protocol Recommendation* any requirements for explicit values of timeouts, but to give guidance instead.

The scope should make it clear which aspects are standardized definitively, which are covered by guidance but not by any requirements, and which are excluded from consideration by the Recommendation*. Any aspects which one might think would be covered, on the basis that they are closely related to aspects which are standardized, need explicit mention.

B.2.7 All options should, if possible, be identified clearly in the scope.

Options are one of the most troublesome, but unfortunately necessary parts of protocol Recommendations*. They fall somewhere between what is standardized and what is not. They will be covered in greater depth below. At this point, what is important is that options are not buried deep inside the Recommendation* but are declared clearly at the beginning. If the number and detailed nature of the options makes this impractical, one should seriously ask whether such complexity is really necessary. Can detailed options be grouped together in some way (e.g. classes) to simplify the Recommendation*?

B.2.8 The scope and field of application should be reviewed after considering the rest of the Recommendation*.

It is not often possible to satisfy some of the above suggestions until the rest of the Recommendation* has been considered. Therefore, it is generally necessary to return to the scope, to check that it really does agree with the contents of the Recommendation*. It is common to find that sections quite outside the scope have found their way in.

B.3 *Guidance on references*

B.3.1 OSI protocol Recommendations* should refer to the OSI reference model, the relevant service Recommendations* and to any relevant Recommendations* for protocol conventions, guidelines, or formal description techniques.

B.3.2 It should be made clear whether conformance to the protocol Recommendation* requires conformance to any part of any other Recommendation*.

B.3.3 It should be made clear whether each reference is to a particular version of the reference Recommendation* or to each successive version.

Normally, the latest version is required, but this can cause problems as changes to the other Recommendation* might affect conformance to this Recommendation*.

B.4 *Guidance on requirements and options*

B.4.1 The status of every requirement should be unambiguous.

Since optional and conditional requirements are so common, there is a tendency to interpret everything which can be interpreted as optional as being optional.

B.4.2 It should be possible for an instance of communication to conform with all the mandatory dynamic conformance requirements.

B.4.3 The conditions under which conditional requirements apply should be spelt out clearly.

B.4.4 It should not be impossible for the implementor or supplier to know what these conditions are.

B.4.5 There should be no possibility of confusion between what is optional dynamically and what is optional statically.

There may be mandatory static conformance requirements for the support of features whose use at the time of communication is optional. Conversely, a message whose use is mandatory in a given context at the time of communication may be part of a protocol mechanism whose support is optional statically.

B.4.6 If the Recommendation* contains a 'shopping list' of options, and there are restrictions on the allowed combinations of such options, then the restrictions should be specified clearly. These should include identification of any mutual exclusions and any minimum and maximum limits to the allowed range of options.

B.4.7 If the Recommendation* does not give any rules for selection of options, it should be made clear in the scope that only the total range and individual options are standardized, but not the selection.

B.4.8 Legitimizing options should be avoided. These are options which allow alternative and incompatible versions of the same thing to claim conformance to the same Recommendation*. While they do not of themselves prevent an objective understanding of conformance, they may frustrate the aims of OSI.

B.4.9 There should be no options which give the implementor permission to ignore important requirements of the Recommendation*. Such options devalue the Recommendation* and the meaning of conformance to it.

B.4.10 If there are prohibitions in the Recommendation*, they should be sufficiently precise to be meaningful.

Many Recommendations* have sections which say in effect 'do all of this and nothing else'. Such prohibitions may be meaningless, because every protocol conveys some information which is not standardized, the so-called 'user data', and every standardized product has attributes which are not standardized, e.g. weight. It may be difficult to draw a clear objective line between things the Recommendation* cannot forbid and those which the writers of the Recommendation* want to forbid, unless the prohibitions are stated explicitly.

B.5 *Guidance on protocol data units*

B.5.1 The permitted set of PDU types and parameter encodings should be stated clearly.

B.5.2 The permitted range of values should be stated clearly for each parameter.

B.5.3 All values, which fall outside the stated permitted range, should be stated explicitly to be invalid.

If not, some people will argue that such values are undefined but allowed, whilst other will argue that they are invalid.

B.5.4 It should be clear whether or not undefined PDU types are allowed.

It is safer if all undefined PDU types are declared to be invalid.

B.5.5 Critical undefined values should be mentioned explicitly in the scope as being undefined.

B.5.6 There should be a defined procedure to be followed by the first communicating party in each case of it receiving an invalid or undefined PDU type or parameter.

B.5.7 It should be possible to detect whether the defined procedure has been followed in such cases. If it is not, then it should be because it does not matter.

Sometimes the procedure to be followed upon receipt of an invalid PDU is intentionally the same as when some valid PDUs are received in the same circumstances. For example, the procedure might be to do nothing until one specific type of PDU is received, everything else being ignored. In such cases, it probably does not matter that the error has apparently gone undetected. In some other cases, the intention may be that error cases should be given special treatment, but that the procedure has been poorly chosen with the result that it cannot be distinguished from the action in non-error cases.

B.5.8 If, in the encoding of PDUs, there are any fields declared to be reserved, then there should be a clear statement of what values, if any, are allowed or disallowed in these fields.

B.5.9 If inter-related parameters can be carried on separate PDUs, then the set of permitted relationships between the values of these parameters should be precisely and clearly defined.

B.5.10 If the parameter encoding allows for parameters to be specified in any order, and the PDU format places restrictions on the permitted orders, then these restrictions should be clearly stated. It should be recognized that if many different orders are permitted, then a large representative sample of different orders ought to be tested. The added complexity of testing conformance should, therefore, be adequately compensated by some advantage in allowing this freedom.

B.5.11 The order in which the bits, octets, etc. should be carried in the underlying protocol should be stated clearly.

For example, should a two octet integer travel most or least significant octet first? It is surprising how often such simple causes of ambiguity are overlooked.

B.5.12 The relationship between SDUs and PDUs should be defined clearly.

B.6 *Guidance on state*

B.6.1 Protocol procedures are often defined using a finite state approach, whether formalized or not. The specification of these states is often incomplete.

B.6.2 Each state should be defined clearly.

B.6.3 If there are events which can only occur in a subset of the possible states, then possible occurrence of an event should be distinguished from valid occurrence.

B.6.4 The required actions and state transitions should be defined for each possible state/event pair. In particular, they should be defined for possible but invalid state/event pairs.

B.7 *Guidance on formal description techniques*

B.7.1 The following requirements apply only to those Recommendations* which include a formal description. Precise, unambiguous Recommendations* can be written without the aid of a formal description technique (FDT), but in complex Recommendations* such as protocols formal descriptions are highly recommended. It should, however, be realized that they can create problems themselves in relation to conformance.

B.7.2 It should be clear whether the formal description forms an essential part of the Recommendation* or is provided only for guidance.

It is very important to have a clear understanding of the status of the formal description. Ideally there should be no discrepancies between the text and the formal description, but because this is very difficult to achieve in practice it is important that the reader knows which takes precedence. If the formal description is provided only for guidance, it cannot define conformance requirements.

B.7.3 The FDT should be well-defined, stable and properly referenced.

B.7.4 If the formal description defines requirements, but not all the requirements of the Recommendation*, then it should be stated clearly that the text includes requirements which are not covered by the formal description and these additional requirements should be identified clearly.

B.7.5 If the formal description defines requirements, and it also defines an allowed way of implementing some aspects of the protocol, but there is intended to be freedom for the implementor to implement those aspects in some other way, then this constitutes over-definition. This is all too common in formal descriptions, and creates difficulties in relation to conformance. If the formal description is an essential part of the Recommendation*, then text should be provided to qualify it, indicating where such over-definition exists and what the real requirements are.

The problem usually arises because the formal description describes the internal behaviour of an idealized implementation, rather than the observable external behaviour that is required. It is only the observable external behaviour which can be tested, and therefore it is only this which should constitute requirements for conformance purposes. It may well be that a different FDT should be used for defining the requirements from that used to provide guidance to implementors.

B.8 *Miscellaneous guidance*

B.8.1 Information which may appear obvious should nevertheless be stated.

If something is omitted because it is 'obvious', some readers will assume it is required because it is 'obvious', but others will assume that it is omitted to provide freedom for implementors. For example, does the existence of a checksum imply that it has to be checked?

ANNEX C

(to Recommendation X.290, Part 2)

Incomplete static conformance requirements

C.1 As a matter of historical record, the development of protocol Recommendations* has been undertaken in parallel with the determination of the meaning of conformance, and in particular with the gaining of understanding of the distinction between static and dynamic conformance.

C.2 Therefore, some early protocol Recommendations* will not give a complete specification of the requirements for static conformance. Typical of the incompleteness is the requirement to support a particular function without saying whether this applies to sending, receiving or both; or the absence of precise conditions of detection of protocol errors in received incoming messages.

C.3 Accordingly, there may be different interpretations of what a conforming implementation is.

C.4 In future Recommendations* or at the time of revision of existing Recommendations* it will be necessary to provide a thorough specification of static conformance requirements. This would include the specification of conditions applicable to the implementation or non-implementation of everything that is neither always mandatory nor always optional.

C.5 In the short term, it is essential that, at very least, current drafts are modified to clarify the present situation: it is not considered acceptable that Recommendations* should be issued in a form in which implementation requirements suffer from extensive ambiguity. Consideration also needs to be given to what should be done where the protocols have already reached the status of ISO International Standard or CCITT Recommendation.

C.6 No other short-term solution is seen other than to accept and state clearly that all capabilities not covered explicitly in the static conformance requirements are optional; and to minimize the potential problems that this may cause by specifying that:

- a) only implementations which
 - 1) implement everything that is explicitly specified as mandatory; and
 - 2) do not omit anything unless it is explicitly stated to be optional, even though there may be a general clause of the sort 'if not specified, then optional';are to be designated as 'conforming' without qualification;
- b) any implementation which
 - 1) implements everything that is explicitly specified as mandatory; and
 - 2) omits things which are not explicitly stated to be optional, perhaps because of a general clause of the sort 'if not specified then optional';shall be described as conforming to a subset.

C.7 Implementations which omit anything that is mandatory do not conform at all.

Note – A system conforming without qualification will not necessarily interwork with another system, nor will it necessarily work better than a system conforming to a subset. The 'perfect' system may refuse from other systems PDUs which it would consider as incorrect or incomplete. Thus, it may reject or abort connections.

Therefore, special consideration should be given to conformance with respect to the detection of protocol errors, especially when this detection can be explicitly or implicitly optional.

ANNEX D

(to Recommendation X.290, Part 2)

The tree and tabular combined notation

D.0 Introduction

In constructing an abstract test suite, a test notation is used to describe abstract test cases. The test notation can be an informal notation (without precisely defined semantics) or a formal description technique (FDT). This Annex describes an informal notation called the tree and tabular combined notation (TTCN).

TTCN serves the following purposes:

- a) to provide a common reference for assessing other test notations and to assist in examining the problems arising in test case and test suite design;
- b) to provide a basis for the translation of test cases into other test notations;
- c) TTCN behaviour descriptions are appropriate for specifying test cases and test steps.

A test suite can be looked upon as a hierarchy ranging from the complete test suite down to test events (see Part 1, § 8.1). TTCN assumes that the basic types of test events are abstract service primitives (ASPs), abstract local primitives (ALPs) and timer events.

Abstract test cases can also be expressed in terms of PDUs (by using an abbreviation mechanism described in § D.5.11).

D.1 *Components of a TTCN test suite*

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

a) Suite overview (§ D.4)

Information needed for the general presentation and understanding of the test suite, such as test reference and a description of its overall purpose.

b) Declarations (§ D.5)

The alphabet of the events to be used in the test suite (e.g., SAPS, Timers, ASPs, PDUs, ALPs and their parameters) is described. This section shall contain the definition of any abbreviations to be used later in the test suite.

c) Dynamic part (§ D.6)

Tables containing trees of behaviour expressed mainly in terms of the occurrence of ASPs at points of control and observation. A set of main behaviour descriptions is followed by a set of default behaviour descriptions.

d) Constraints part (§ D.8)

This section specifies values for the ASPs, PDUs, and their parameters used in the Dynamic part.

D.2 *The syntax form of TTCN*

TTCN is provided in graphical form (TTCN-GR), suitable for human readability.

Note – A machine processable form of (TTCN-MP), suitable for transmission of TTCN descriptions between machine and possibly suitable for other automated processing is for further study.

D.3 *Conventions*

D.3.1 *Table proformas*

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

a) Bold text (**like this**), or text in this font shall appear verbatim in each actual table;

b) Text in italics (*this font*) shall not appear verbatim. This font is used to indicate that the actual text must be substituted for the italicised symbol.

Additionally, within table proformas comments may be placed in fields not reserved for commentary, by delimiting the comments with the symbols /* and */.

D.3.2 *Types of bracket*

The terms that will be used when referring to the various different types of bracket are defined in Figure D-2/X.290 Part 2.

Brackets: [. .]
Braces: { . . }
Paréntheses: (. .)

FIGURE D-2/X.290, Part 2

Brackets

D.3.3 *Naming conventions*

D.3.3.1 *Test group and test case references*

a) The reference (name) of a test group shall be of one of the forms shown in Figure D-3/X.290, Part 2 and illustrated in Example D-2.

(Test-suite)/(Test-group)/
(Test-suite)/(Test-group₁)/.../(Test-group_n)/

FIGURE D-3/X.290, Part 2

Test group references

Example D-2 – A transport group reference:

Transport/Class0/Conn-Estab/

- b) The reference (name) of a test case shall be of one of the forms shown in Figure D-4/X.290, Part 2 and illustrated in Example D-3.

(TestSuite)/(TestCase)/
(TestGroupReference)/.../(TestCase)/

FIGURE D-4/X.290, Part 2

Test case references

Example D-3 – A transport test case reference:

Transport/Init

Transport/Class0/Conn-Estab/LT-Init

- c) The test group references, together with the test case references define the structure of the test suite.

D.3.3.2 Test step references

- a) Test steps are attached at a given point in the structure of the test suite, as defined by the group and case references (see § D.3.3.1).
 b) Test steps, at their point of attachment may be grouped into “libraries”, organized in hierarchical fashion.
 c) A step reference is therefore of the form:

(PointOfAttachment) : (LibraryStructure)

where:

- 1) *(PointOfAttachment)* is either:
 A) *(TestSuite)* or
 B) *(TestGroupReference)* as defined in § D.3.3.1; or
 C) *(TestCaseReference)* as defined in § D.3.3.1;
 2) and *(LibraryStructure)* is either:
 A) *(TestStep)* or
 B) *(Component)₁/. . ./(Component)_n/(TestStep)*

Example D-4 – Transport Test Step References:

Transport:Step-A
 Transport/Class0/Conn-Estab/:Step-B
 Transport/Class0/Conn-Estab/LT-Init:Step-C
 Transport:Common-lib/Preambles/Step-C

Note 1 – The colon (:) in the test step reference separates the first part which refers to a point in the already defined test suite structure from the second part which defines the library structure.

Note 2 – The components allow the steps to be grouped into an arbitrary hierarchy within libraries, which have no influence on the test suite structure itself.

D.4 *Suite overview*

This section shall include at least the following information:

- a) references to the relevant base standards;
- b) a reference to the PICS and PIXIT and how they are used;
- c) an indication of the test method or methods to which the test suite applies;
- d) a complete index to the test suite, consisting of the test reference, test identifier, page number and test purpose for each test case and test step in the test suite. The test purposes shall be organized according to the structure of the test suite.

Other information which may aid understanding of the test suite, such as how it has been derived, should also be included as commentary.

This information shall be provided in the format shown in Figure D-5/X.290, Part 2.

D.5 *Declarations*

The purpose of the declarations section is to describe the set of test events and all other attributes to be used in the test suite. All objects used in the dynamic part shall have been declared in the declaration part. There are two sorts of test event:

- a) abstract service primitives (ASPs) which occur at the points of control and observation (PCOs) used by the tester (§ D.5.8);
- b) timer events (§ D.5.10).

Other attributes are also specified:

- a) user defined types (§ D.5.1.3);
- b) user defined operators (§ D.5.3);
- c) test suite parameters (§ D.5.4);
- d) global constants (§ D.5.5);
- e) global variables (§ D.5.6);
- f) PCOs (§ D.5.7);
- g) ASP parameters (§ D.5.8);
- h) data types (including PDUs and their parameters) (§ D.5.9);
- i) abbreviations (§ D.5.11).

Suite Overview			
Reference to Standards: ... Reference to PICS: ... Reference to PIXIT: ... How Used: ... Test Method(s): ... Comments: ...			
Test Case identifier	Test Case Reference	Page	Purpose
. . . (Test Case Identifier) (Test Case Reference) (Page) (Purpose) . . .

FIGURE D-5/X.290, Part 2

Suite overview proforma

D.5.1 *General TTCN types*

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

D.5.1.1 *Predefined types*

A number of commonly used types are predefined for use in TTCN. These types may be referenced even though they do not appear in a type declaration in a test suite. All other types used in a test suite must be named in the User Type Declarations (according to § D.5.1.2) and referenced by name.

- a) **Integer Predefined Type:** a type with distinguished values which are the positive and negative whole numbers, including zero (as a single distinguished value).
- b) **Bitstring Predefined Type:** a type whose distinguished values are the ordered sequences of zero, one, or more bits.
- c) **Octetstring Predefined Type:** a type whose distinguished values are the ordered sequences of zero, one, or more octet being an ordered sequence of eight bits.
- d) **Character String Predefined Types:** types whose distinguished values are zero, one or more characters from some character set. The character string types listed in Figure D-6/X.290, Part 2 may be used. They are defined in section two of Recommendation X.208).

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

FIGURE D-6/X.290, Part 2
Predefined character string types

- e) **The Connection Endpoint Identifier Predefined Type (CEId):** a type consisting of an unlimited number of distinguished values.

D.5.1.2 *User defined types*

Types specific to a test suit may be introduced by the TTCN user. To define a new type, the following information shall be provided:

- a) a name for the type;
- b) the base type (if any);
- c) a definition of the type, provided in one of the following manners:
 - 1) by giving a precise reference to a clause, or clauses, of a standard which defines the type;
 - 2) by giving an ASN-1 [Recommendation X.208] type reference of the form:
(modulereference).(typereference)
 - 3) by enumerating the set of named distinguished values comprising the type (see Figure D-7/X.290, Part 2);
 - 4) by specifying a subset of the distinguished values of another type.
This may be done in a number of ways:
 - A) by specifying a subrange of the *Integer* predefined type;
 - B) by specifying a subrange of an enumerated type;
 - C) by restricting the length of a *BitString*, *OctetString*, or a character string predefined type.

This information shall be provided in the format shown in Figure D-8/X.290, Part 2.

User Type Definition			
Name	Base Type	Definition	Comments
.	.	.	.
.	.	.	.
.	.	.	.
(Name)	(Base Type)	(Definition)	(Comments)
.	.	.	.
.	.	.	.
.	.	.	.

FIGURE D-7/X.290, Part 2
User type definition proforma

User Type Definitions			
Name	Base Type	Definition	Comments
Transport-Classes	None	(Class 0, Class 1, Class 2, Class 3, Class 4)	Classes that may be used for Transport layer connection

FIGURE D-8/X.290, Part 2
Example user type declaration

D.5.2 Value denotation

Values of the different types shall be denoted in the following fashion:

D.5.2.1 Values of predefined types

The values of predefined types shall be denoted as follows:

- a) **Integer Values:** values of type *Integer* shall be denoted by one or more digits. The first digit shall not be zero unless the value is 0.
- b) **BitString and OctetString Values:** values of type *BitString* and *OctetString* shall be denoted in one of the following ways:
 - 1) by a list of bits;

In this case, the value shall be denoted by an arbitrary number (possibly zero) of zeros and ones, preceded by a single ' and followed by the pair of characters 'B.

Example D-5 – '01101100'B

- 2) by a list of semi-octets.

In this case, the value shall consist of an arbitrary number (possibly zero) of the characters

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

preceded by a single ' and followed by the pair of characters 'H. Each character is used to denote the value of a semi-octet using a hexadecimal representation.

Example D-6 – 'AB0196'H

- c) **Character String Values:** values of character string types shall be denoted by an arbitrary number (possibly zero) of characters from the character set referenced by the character string type, preceded and followed by ". If the character string type includes the character ", this character shall be represented by a pair of "" in the denotation of any value.
- d) **Connection Endpoint Identifier Values:** values of the *CEId* type have no denotation.

Note – None is required, since the only operation defined over the *CEId* type is equality (see § D.5.8).

D.5.2.2 Values of user defined types

Values of the user defined types shall be denoted as follows:

- a) The denotation of values of the types introduced by reference to the section, or sections, of a Recommendation* shall specify how the distinguished values of that type are to be denoted in the test suite, in the comments associated with the type definition.
- b) a value of a referenced ASN-1 type shall be denoted using ASN-1 either by
 - 1) a reference of the form:
 $(modulereference).(valuereference)$ or
 - 2) specifying an ASN-1 value of the given type.
Note – The ASN-1 modular method has extensions that allow the value to be partially specified (§ D.8.3).
- c) a value of an enumerated type shall be denoted by its name.
- d) a value of a type obtained by subsetting another type shall have the same denotation as the values of the type which was subset.

D.5.3 Operators

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

- a) a name for the operation;
- b) the signature of the operation, comprising:
 - 1) a list of input types;
 - 2) a name for each input component;
 - 3) the type of the result;
- c) a description of the operation.

This shall be provided in the format shown in Figure D-9/X.290, Part 2.

Operation Definition	
Operation Name:	<i>(Operation Name)</i>
Input Types:	List of <i>(Type Names)</i>
Components:	List of <i>(Name)</i>
Result Type:	<i>(Type Name)</i>

Description:	<i>(Description)</i>
--------------	----------------------

FIGURE D-9/X.290, Part 2

Operation definition proforma

The definitions of two string operations are given in Figures D-10/X.290, Part 2 and D-11/X.290, Part 2.

Operation Definition	
Operation Name:	<i>substr</i>
Input Types:	<i>string x integer x integer</i>
Components:	<i>source, start, length</i>
Result:	<i>string</i>

Description:	<i>substr (source, start, length)</i> is the string of length <i>length</i> starting from index <i>start</i> of <i>source</i>
For example:	$\text{substr}(\text{"abcde"}, 3, 2) = \text{"cd"}$ $\text{substr}(\text{"abcde"}, 4, 9999) = \text{"de"}$

FIGURE D-10/X.290, Part 2

Definition of operation substr

Operation Definition	
Operation Name:	<i>length</i>
Input Types:	<i>string</i>
Components:	<i>source</i>
Result:	<i>integer</i>

Description:	<i>length (source)</i> is the length of the string <i>source</i> For example: <i>length ("abcde") = "5"</i>
--------------	--

FIGURE D-11/X.290, Part 2

Definition of operation length

Note – An operation may be compared to a function in an ordinary programming language. However, the arguments to the operation are not altered as a result of any call of the operation – there are no side effects.

D.5.4 *Test suite parameters*

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

Note – In most real cases of testing Test Suite Parameters will be bound to a value when the PICS/PIXIT processing occurs.

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

- a) its name;
- b) its type.

This information shall be provided in the format shown in Figure D-12/X.290, Part 2.

Test Suite Parameters		
Name	Type	Comments
.	.	.
.	.	.
.	.	.
<i>(Name)</i>	<i>(Type)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-12/X.290, Part 2

Test suite parameters proforma

D.5.5 *Global constants*

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

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

- a) its name;
- b) its type;
- c) its value.

This information shall be provided in the format shown in Figure D-13/X.290, Part 2.

D.5.6 *Global variables*

A test suite may make use of a set of variables that are global to both the dynamic part and constraints part together. Typically, these variables will be used to reference values of PDU or ASP constraint fields, or components, from the dynamic part. Variables as in a conventional programming language (e.g. as counters).

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

- a) its name;
- b) its type.

This information shall be provided in the format shown in Figure D-14/X.290, Part 2.

Global Constants			
Name	Type	Value	Comments
.	.	.	.
.	.	.	.
.	.	.	.
<i>(Name)</i>	<i>(Type)</i>	<i>(Value)</i>	<i>(Comments)</i>
.	.	.	.
.	.	.	.
.	.	.	.

FIGURE D-13/X.290, Part 2
Global constants proforma

Global Variable Declarations		
Variable Name	Type	Comments
.	.	.
.	.	.
.	.	.
<i>(Variable Name)</i>	<i>(Type)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-14/X.290, Part 2
Global variable declarations proforma

Initially, all variables are unbound.

Variables may become bound (or be rebound) in the following contexts:

- a) when the variable appears on the left-hand side of an assignment statements (D.6.7.1);
- b) when the unbound variable appears in a Boolean expressions (§ D.6.7.1);
- c) when the variable appears in a constraints reference (§ D.8).

D.5.7 PCO declarations

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

Note – The test method chosen determines the PCOs required to define the test suite.

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

- a) its name:
 - the name will be used in the Behaviour Section to specify where particular events occur;
- b) an explanation of which type of tester is placed at the PCO and what role that tester plays.

This information shall be provided in the format shown in Figure D-15/X.290, Part 2.

PCO Declarations	
Name	Rôle
.	.
.	.
.	.
(Name)	(Rôle)
.	.
.	.
.	.

FIGURE D-15/X.290, Part 2
PCO declarations proforma

Example D-7 – An example PCO declaration is shown in Example D-8.
Example D-8 – PCO Declarations.

PCO Declarations	
Name	Rôle
L	SAP at the lower tester/(N-1)-service [lower tester is (N-1)-service user]
U	SAP at the upper tester/(N)-service [upper tester is (N)-service user]

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

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

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

Example D-9 – A typical example in which one PCO corresponds to several SAPs could be an Internet lower tester which uses one PCO representing all the *subnetwork points of attachment* for sending several Internet PDUs over different routes. Alternatively the same example may be written with several PCOs.

It is also possible to consider having a single SAP correspond to several PCOs. In this case there would be one PCO per connection.

Note – This makes it easier to identify each test event with the appropriate connection.

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

D.5.8 *ASP declarations*

This section lists the set of ASPs which may occur at the PCOs listed in § D.5.7.

Normally, the declared information can be found in the appropriate service definition. However, declaring it explicitly allows the addition of commentary specific to testing and to a particular test suite, as well as providing for cases where no explicit OSI* service definition exists (e.g. X.25).

The following information shall be supplied for each ASP:

- a) its name;
if an abbreviated name is used, then the full name (as given in any appropriate service specification) shall follow in parentheses;
- b) the PCO or PCOs at which it may occur;
all such PCOs shall have been declared in the PCO declarations section of the test suite;
- c) whether or not a connection endpoint identifier is used to distinguish different instances of the ASP;
If it is stated that a connection endpoint identifier is used (see c), above), a parameter named *ceid*, of type *CEId* is available without further declaration,
upon receipt of an ASP which uses this parameter, the value of *ceid* shall become instantiated to the connection endpoint at which the ASP was received. This value is then available for use in the test case.

Example D-10 – Using a CEId:

PCO? AN-ASP [ceid=3]

- d) a list of the parameters associated with the ASP;
the following information shall be supplied for each parameter:
 - 1) its name;
if an abbreviated name is used, then the full name (as given in any appropriate service specification) shall follow in parentheses;
 - 2) its type.

ASP declarations have only one level of parameter. However, these parameters may be of an arbitrarily complex type (e.g. a complex ASN-1 type).

This information shall be provided in the format shown in Figure D-16/X.290, Part 2.

Example D-11 – Figure D-17/X.290, Part 2, shows an example from the Transport Service (Recommendation X.214). This could be part of the alphabet of ASPs used to describe the behaviour of an abstract upper tester in a DS test suite for the Class 0 Transport. CDA, CGA and QOS are user defined types. (See protocols, Recommendation X.224.)

ASP Declaration		
ASP: <i>(ASP)</i>	PCO: <i>(PCO) (list)</i>	CEId: Used or Unused

Service Control Information		
Parameter Name	Type	Comments
.	.	.
.	.	.
.	.	.
<i>(Parameter Name)</i>	<i>(Type)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-16/X.290, Part 2
Abstract service primitive declaration

ASP Declaration		
ASP: CONreq (T-CONNECT request)	PCO: TSAP	CEId: USED

Service Control Information		
Parameter Name	Type	Comments
CdA (Called Address)	CDA	... of upper tester
CgA (Calling Address)	CGA	... of lower tester
QoS	Implementation dependent	Should ensure that Class 0 is used

FIGURE D-17/X.290, Part 2
Example abstract service primitive declaration

D.5.8.1 *ALP declarations*

ALPs are declared using a proforma similar to the ASP proforma. The following information shall be supplied for each ALP:

- a) its name;
- b) the PCO or PCOs at which it may occur;
- c) a functional description of the ALP;
- d) a list of the parameters associated with the ALP;

The following information shall be supplied for each parameter:

- 1) its name;
if an abbreviated name is used, then the full name (as given in any appropriate service specification) shall follow in parentheses;
- 2) its type.

This information shall be provided in the format shown in Figure D-18/X.290, Part 2.

ALP Declaration		
<i>ALP: (ALP)</i>	<i>PCO: (PCO list)</i>	<i>Description: (Functional Description)</i>

Service Control Information		
Parameter Name	Type	Comments
.	.	.
.	.	.
.	.	.
<i>(Parameter Name)</i>	<i>(Type)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-18/X.290, Part 2

Abstract local primitive declaration

D.5.9 *Data type declarations*

The purpose of this section is to declare the data types which are used in the test suite. These types will be used mainly in ASP parameter declarations.

The most common data type declarations are PDU declarations. Other data type declarations may include ASN-1 type declarations, if appropriate.

D.5.9.1 *PDU declarations*

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

- a) its name;
if an abbreviated name is used, the full name (as given in any appropriate protocol specification) shall follow in parentheses;
- b) a list of the parameters or, more generally fields associated with the PDU.

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

The following information shall be supplied for each parameter:

- 1) its name;
if an abbreviated name is used, the full name (as given in any appropriate protocol specification) shall follow in parentheses;
- 2) its type.

This information shall be provided in the format shown in Figure D-19/X.290, Part 2.

Data Type Declaration	
PDU: <i>(PDU)</i>	Comments: <i>[General Comments (e.g., restrictions on use)]</i>

Protocol Control Information		
Field Name	Type	Comments
.	.	.
.	.	.
.	.	.
<i>(Field Name)</i>	<i>(Type)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-19/X.290, Part 2

Data type declaration proforma

Where more appropriate, a precise reference to an ASN-1 type description of a PDU can be used, instead of supplying all of the above information. In this case, the information shall be provided in the format show in Figure D-20/X.290, Part 2.

Data Type Declaration	
PDU Name	ASN.1 Type Definition
.	.
.	.
.	.
<i>(PDU Name)</i>	<i>(ASN.1 Type Definition)</i>
.	.
.	.
.	.

FIGURE D-20/X.290, Part 2
Data type declaration proforma

Example D-12 – Figure D-21/X.290, Part 2 shows an example ASN-1 PDU type definition from FTAM (ISO 8571).

ASN.1 Data Type Declaration	
PDU Name	ASN.1 Type Definition
F-INIT	ISO 8571-FTAM.PDU

FIGURE D-21/X.290, Part 2
ASN.1 Data type declarations

D.5.10 Timers

A test suite may make use of several types of timers. These types are used to distinguish the length of time the timers take to expire. A test case may make use of any number of instances of the same type of timer.

The following information shall be provided for each type of timer:

- a) the timer type name;
- b) the duration of the timer, which may be specified as a value or a range of values.

This information shall be provided in the format shown in Figure D-22/X.290, Part 2.

Timer Declarations		
Timer Type Name	Duration	Comments
.	.	.
.	.	.
.	.	.
(Timer Type Name)	(Duration)	(Comments)
.	.	.
.	.	.
.	.	.

FIGURE D-22/X.290, Part 2

Timer declarations proforma

Example D-13 – Figure D-23/X.290, Part 2 shows an example timer declaration.

Timer Declaration		
Timer Type Name	Duration	Comments
Receive-Timer	1. .3	Define a set of timers to be used when receiving data

FIGURE D-23/X.290, Part 2

Example timer declaration

D.5.11 *Abbreviations*

This section defines any abbreviations that are to be used in the rest of the test suite. Abbreviations are used as a macro facility, performing simple textual substitution operations. They may be used throughout a test suite.

An abbreviation may replace any piece of text within any single table box. The test case writer shall ensure that the resulting expansion follows the syntax of TTCN.

An abbreviation definition shall provide the following information.

- a) an abbreviation identifier, or token;
- b) its expansion, to be substituted for every occurrence of the identifier throughout the test suite.

This information shall be provided in the format shown in Figure D-24/X.290, Part 2.

Abbreviations		
Abbreviation	Expansion	Comments
.	.	.
.	.	.
.	.	.
<i>(Abbreviation)</i>	<i>(Expansion)</i>	<i>(Comments)</i>
.	.	.
.	.	.
.	.	.

FIGURE D-24/X.290, Part 2

Abbreviations proforma

Example D-14 – Figure D-25/X.290, Part 2 shows an example abbreviation declaration from a Transport (Recommendation X.224) test suite.

Note – The ~ operator is defined in § D.6.8.

Abbreviation Declaration		
Abbreviation	Expansion	Comments
CR	N-DATAind [NSDU ~ CR-TPDU]	CR denotes any N-DATA indication whose Network Service Data Unit is the encoding of a Connection Request Transport Protocol Data Unit

FIGURE D-25/X.290, Part 2

Example abbreviations declaration

D.6 *Dynamic part*

The Dynamic Part contains the main body of the test suite – the test case and/or test step behaviour descriptions.

D.6.1 *Dynamic behaviour description*

The following information shall be supplied for the behaviour description of each test case, test step or set of related test steps:

- a) a reference; the reference gives a name to the test case or step behaviour description. A test case reference shall conform to the requirements of § D.3.3.1. A test step reference shall conform to the requirements of § D.3.3.2;
- b) an identifier; a reference reflects the structure of the test suite and the purpose of the test case or test step, and so may be quite lengthy. It is sometimes desirable to have a short name for a test case or test step. The identifier serves this purpose and may be used interchangeably with a reference. The test identifier shall be unique within a particular test suite;
- c) a statement of purpose; this shall be an informal statement of purpose of the test case or test step or steps;
- d) Defaults Reference; this shall be reference to a default behaviour description, if any, which applies to this behaviour specification (§ D.6.17);
- e) a behaviour description; this section shall describe the behaviour of the tester or testers in terms of test events (and their parameters) in the tree notation described in § D.6.2.

This information shall be provided in the format shown in Figure D-26/X.290, Part 2.

Dynamic Behaviour				
Reference: <i>(Test Case or Test Step Reference)</i>				
Identifier: <i>(Identifier)</i>				
Purpose: <i>(Purpose)</i>				
Defaults Reference: <i>(Defaults Reference)</i>				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
.
.
.
<i>(Behaviour Description)</i>	<i>(Label)</i>	<i>(Constraints Reference)</i>	<i>(Verdict)</i>	<i>(Comments)</i>
.
.
.
Synchronization Requirements <i>(optional)</i>				
Extended Comments <i>(optional)</i>				

FIGURE D-26/X.290, Part 2
Dynamic behaviour declaration proforma

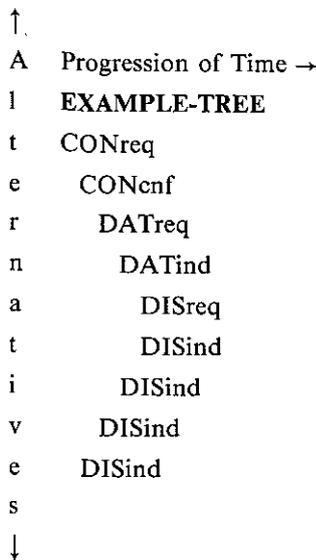
D.6.2 *The tree notation*

The tree notation is used to provide the behaviour descriptions for a test suite. The behaviour descriptions are enumerations of possible observable sequences of test events.

Example D-15 – Suppose that the following sequences of events could occur during a test whose purpose was simply to establish a connection exchange some data, an close the connection:

- a) CONreq, CONcnf, DATreq, DATind, DISreq
- b) CONreq, CONcnf, DATreq, DATind, DISind
- c) CONreq, CONcnf, DATreq, DISind
- d) CONreq, CONcnf, DISind
- e) CONreq, DISind

Progress can be thwarted at any time by the underlying service-provider. The tree notation simply factors out common initial sequences of the complete set. Using the tree notation, this would be written:



Events at a same level of indentation represent the possible alternative events which may occur at that time. Alternative events shall be given in the order in which the tester shall attempt to complete them.

D.6.3 *Test events*

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

Example D-16 – Using this convention, the Example Tree could be represented as follows:



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

In cases where behaviour cannot be specified in terms of TTCN events (e.g. by virtue of the test method being used) then behaviour descriptions shall be given in natural language instead.

Example D-17 – Test Step from Generic Test Suite.

PARTIAL-TREE

```
?N-DATAind[UserData^DR]
  !N-DATAreq[UserData^DC]
  +POSTAMBLE
```

POSTAMBLE

```
/* close all open network connections */
```

D.6.3.1 *?OTHERWISE*

The predefined pseudo-event (*PCO*)*?OTHERWISE* may be used to denote any ASP or ALP event which the tester may receive at the PCO.

Example D-18 – Assume that events A, B and C may be received at a given PCO:

```
?A [X=1]
?B
?OTHERWISE
  some behaviour . . .
? TIMEOUT
```

is the same as

```
?A
?B
?A

  some behaviour . . .
?B

  some behaviour . . .
?C

  some behaviour . . .
?TIMEOUT
```

When using *?OTHERWISE* the following points should be noted:

- a) *?OTHERWISE* is always expanded with all possible events that may be received at the PCO since, even if an event is present at the same level, a Boolean expression or synchronisation requirement may prevent this event occurring;
- b) if a tree makes use of multiple PCOs then a PCO shall be stated on the *?OTHERWISE* statement;
- c) *?OTHERWISE* need not be the last among a set of alternatives;
- d) *?OTHERWISE* may have a Boolean expression and/or an assignment associated with it;

Example D-19 – Illustration of a qualified *?OTHERWISE*

MAIN-TREE (X)

```
?A pass
?B[X=2] fail1
?OTHERWISE[X=2] fail1
?C fail1
?OTHERWISE pass
```

if the tree is invoked with X = 2 then:

```
A => pass
B => fail1
any other event => fail1
```

if the tree is invoked with X ≠ 2 then:

```
A => pass
C => fail1
any other event => pass
```

- e) ?OTHERWISE does not prevent the use of default. The defaults are appended to the set of alternatives and will be examined in order. This means that ?OTHERWISE will invalidate some defaults but not necessarily all.

Example D-20 – Use of ?OTHERWISE in both the main and default tree:

MAIN-TREE

PCO1? A
PCO2? B
PCO1? OTHERWISE

DEFAULT-TREE

PCO2

?OTHERWISE fail₁
PCO1? C fail
 ?TIMEOUT pass

The first and third events of the default are not invalidated by the ?OTHERWISE in the main tree.

D.6.4 *Tree names*

A behaviour description shall contain at least one behaviour tree. Each behaviour tree shall be prefixed by a tree name which is unique within a behaviour description (see § D.6.5).

Note – Many of the examples show tree names in bold text. This is for typographical clarity only and has no other significance.

D.6.4.1 *Formal PCOs*

Where a test suite involves the prescription of behaviour at more than PCO, a tree name shall be followed by a list of the (formal) PCO names used in the tree enclosed in brackets. Where a test suite only prescribes behaviour at one PCO, then this PCO list may be omitted.

Example D-21 – A tree involving the PCOs L and U might be named: **TREE-NAME[L,U]**.

D.6.4.2 *Formal parameters*

If a test step makes use of input parameters then after the PCO list (if any), a list of formal parameters to the tree may appear. These shall be enclosed in parentheses. TTCN does not support output parameters.

Example D-22 – **TREE-NAME[L,U](X,Y)**

D.6.5 *Tree attachment*

Trees may be attached to other trees by substituting the name of the tree to be attached, prefixed by a +, for an event name. The name of the attached tree shall be of the form:

- a) <TreeReference>, or
- b) <TreeReference> [<ActualPCOs>], or,
- c) <TreeReference> (<ActualParameters>), or,
- d) <TreeReference> [<ActualPCOs>](<ActualParameters>)

When a tree is attached to another tree, all the actuals are substituted for the formals using simple textual replacement. So an attached tree is analogous to a macro.

D.6.5.1 *Scope of attachment*

Behaviour descriptions may contain more than one tree. However, only the first tree in the behaviour description is accessible from outside the behaviour description. Any subsequent trees are considered to be local to the test steps, and thus not externally accessible. Test cases, of course, are not attachable.

Thus the <TreeReference> shall be one of the forms:

- a) <TreeName>

In this case <TreeName> shall be the name of one of the trees in the current behaviour description.

- b) <TestStepReference>|<TreeName>

In this case, <TreeName> shall be the name of the *first* tree in the behaviour description of the test step referred to by the test step reference.

As usual, the equivalent *<TestStepIdentifier>* may be substituted for the *<test step reference>*.

Example D-23 – The following pair of trees:

MAIN-TREE [L,U] and	SUB-TREE[X]
L!CONreq	X?CONind
+ SUB-TREE[U]	

is equivalent to:

COMPOUND-TREE [L,U]
L!CONreq
U?CONind

Since tree attachment is really only a shorthand, variables are used in attached trees with the value they have at the point when the tree is attached, and may subsequently be used in assignment and/or Boolean expressions in the attached tree.

Attached trees may have parameters. Actuals are substituted for formals when the tree is attached.

Example D-24 – The following pair of trees.

MAIN-TREE and	SUB-TREE (X,Y)
(M:= 1)	(X:= Y)
+ SubTree(M,2)	
(M:= 3)	

is equivalent to

COMPOUND-TREE
(M:= 1)
(M:= 2)
(M:= 3)

D.6.6 *Labels and GOTO*

A set of alternative events may be labelled by placing a label in the label column of the *first* event of the alternatives.

If a set of alternatives is labelled it is permitted to go to that set from any point in the tree. This is accomplished by placing → or the keyword GOTO followed by the name of a label that is defined in the same tree, immediately after an event. Should the event occur, the test proceeds with the set of alternatives referred to by the label.

Example D-25 – Figure D-27/X.290, Part 2 illustrates the use of the GOTO.

Events, not necessarily the first of a set of alternative events, may also have a label for the specification of synchronization (§ D.6.11) requirements. Such labels shall not be used as the object of a GOTO.

In the case where a first event needs to be labelled for both GOTO and for a synchronization requirement then a common label shall be used.

D.6.7 *Expressions*

The terms of an expression may be:

- constants (including test suite parameters);
- bound variables;
- parameter values of the associated event (if any) referenced by the parameter name given in the event declaration;
- references to encoded PDU field values in ASP parameters. These references take the form:
<ASP Parameter>.<FieldName>

Dynamic Behaviour				
Reference: Example/GOTO Identifier: GT Purpose: To illustrate the use of "GOTO" Defaults Reference: –				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
<pre> /* Loop around echoing data. Pass, if data is received in response to each data sent.*/ EchoUntilDis !DATAreq ?DATAind → again ?DISind ?DISind </pre>	Again		Fail Pass	Did not receive echo

FIGURE D-27/X.290, Part 2
Example illustrating the use of GOTO

D.6.7.1 Assignment clauses

Any event may be followed by a series of assignments. The effect of the assignment clause is to bind the global variable to the value of the expression if, and only if, the event occurs.

The assignments occur in the order in which they appear in the assignment clause. The expression shall contain no unbound variables.

D.6.7.2 Boolean expressions

An event may be qualified. This qualification is achieved by placing a bracketed Boolean expression after the event. This qualification shall be taken to mean the combination of the event occurring and the Boolean expression being satisfied.

If both a Boolean expression and an assignment clause are associated with the same event, the Boolean expression shall appear first.

The terms used in a Boolean expression can be any of the following:

- a) constants (including test suite parameters);
- b) variables;
 if any unbound variables appear in a Boolean expression, then the event occurs only if values exist for the unbound variables which satisfy the Boolean expression. If the event does occur, all unbound variables become bound to any values consistent with their type and which satisfy the Boolean expression;
- c) parameter values of the associated ASP (if any) referenced by the parameter name given in the ASP declaration;
- d) references to a PDU encoding in the Constraints part (§ D.8). This encoding may be qualified by writing:
`<PDUReference>[<Boolean Expression>]`
 where the terms of the Boolean Expression may be references to fields of the referenced PDU;
- e) references to encoded PDU field values in ASP parameters in the Constraints Part. These references take the form:
`<ASPPParameter>.<FieldName>`

D.6.7.3 *Assignment clauses and Boolean expressions with events*

It is allowed to associate an event with either an assignment, or a Boolean expression or both. However, only the following combinations may be used:

- a) *<Event>* The event is not qualified.
- b) *<Event>* (*<Assignment>*) The assignment is executed only if the event occurs.
- c) *<Event>*[*<Boolean Expression>*] The event may occur only if the boolean expression holds.
- d) *<Event>*[*<Boolean Expression>*](*<Assignment>*) The assignment is executed if, and only if, both
 - 1) the event occurs, and
 - 2) the Boolean expression holds.

Since all variables occurring in the Boolean expression become bound, they may be used in the expression which is part of the assignment.

Boolean expressions may be broken down into an uninterrupted set of Boolean expressions. This facility may be useful when used in conjunction with abbreviations (§ D.5.11). Similarly, assignments may be broken down into an uninterrupted set of assignments.

Example D-26 – $PCO? CR[X=1][Y < 3]$ is equivalent to $PCO? CR[(X=1)AND(Y < 3)]$

D.6.7.4 *Assignment clauses and Boolean expressions without events*

It is permitted to use assignment clauses and Boolean expressions by themselves, without any associated event.

Only the following compositions are allowed:

- a) (*<Assignment>*) The assignment is executed only if the event occurs.
- b) [*<Boolean Expression>*] The event may occur only if the Boolean expression holds.
- c) [*<Boolean Expression>*](*<Assignment>*) the assignment is executed only if the Boolean expression holds.

D.6.8 *The encode/decode operator*

The encode/decode operator allows the specification of the encoding of PDU fields. The syntax for this is defined in § D.6.8.2. The ~ operator should be read as “is the encoding of”.

Example D-27 – Consider the event:

N-SAP? N-DATAind[UserData ~ CR-TPDU] CR1

means that the event matches if, and only if, at N-SAP we receive an N-DATAind whose UserData field is the correct encoding of a CR-TPDU according to the constraint CR1 (see § D.6.13).

Suppose that F1 is a field of the constraint CR1. If this event occurs, then F1 can be referenced by writing UserData.F1.

Note – If the content of F1 is a free variable (§ D.8.2) or a global variable (§ D.5.6), then the event occurring will bind this variable (See § D.6.7.2).

Similarly:

N-SAP! N-DATAreq[UserData ~ CC-TPDU] CC1

means that the tester sends an N-DATAreq whose UserData field is the encoding of a CC-TPDU according to the constraint CC1.

D.6.8.1 *Aliasing*

A simple mechanism for preserving the values of entire PDUs is provided. The alias is an implicitly bound variable that may be used as a shorthand or to distinguish between two fields (PDU or ASP) that have the same name.

Example D-28 – Use of the alias:

N-SAP? N-DATAind[UserData UD1 ~ CR-TPDU[UserData UD2 ~ TM-PDU]] CR1, TM1

the aliases UD1 and UD2 are bound to their respective UserData [or, more precisely, bound to the constraints CR(CR1) and TM(TM1)]. Suppose that F1 is a field of CR-TPDU(CR1) and that F2 is a field of the constraint TM(TM1). Then these fields may be referenced as UD1.F1 and UD2.F2.

D.6.9 *Parallel trees*

It is sometimes convenient to be able to describe behaviour in terms of separate behaviour descriptions occurring in parallel. Typically, each separate description will refer to behaviour at a different PCO.

Parallel trees are denoted as follows:

|| Tree₁ , . . . , Tree_n

The following rules shall apply:

- a) the subscript n shall be greater than 1;
- b) the tree which splits its behaviour into parallel trees shall have more than one PCO and at least as many PCOs as there are parallel trees;
- c) all PCOs of the current tree shall be a PCO of one and exactly one of the parallel trees;
- d) there shall be no timer(s) running or suspended;
- e) there shall be no other event at the same level of alternative;
- f) only the first tree in the statement (i.e. Tree₁) may assign a verdict;
- g) when the first tree (i.e. Tree₁) terminates:
 - 1) If Tree₁ terminates with a verdict assigned then the test case terminates with that verdict;
 - 2) If Tree₁ terminates without a verdict (i.e. a tip of Tree₁ is reached) then the behaviour continues as if Tree₁ was attached.
 - A) Tree₂ . . . Tree_n are disregarded;
 - B) subsequent behaviours of the main tree are appended to the tips of Tree₁;
 - 3) if a tip of Tree_i (i > 1) is reached, then:
 - A) if any further event occurs at a PCO of Tree₁ then this is an error;
 - B) otherwise (g)2 will apply (i.e. Tree₁ has to terminate at some point).

Example D-29 – Lower and upper tester behaviour as parallel trees:

```

MAIN-TREE [UT,LT]
+connect-link-layer(LT)
  UT! N-CONreq
    || LT-TREE(LT),UT-TREE(UT)
      +disconnect-link-layer(LT)
    pass

UT-TREE [UT]
  UT? N-CONcnf
    L
  UT! N-DTAreq → L
  ? Otherwise → L
  UT? Otherwise → L

LT-TREE [LT]
  LT? N-DTAind [call-request]
  LT! N-DTAreq [call-accept]
  LT? B-DTAind [dt-packet]

DEFAULT-TREE
  UT? Otherwise
    +disconnect-link-layer
    fail1
  
```

D.6.10 *Timer management*

It is assumed that the timers used in a test suite will always be in one of the following states:

- a) inactive,
- b) running,
- c) suspended.

D.6.10.1 *Timer operations*

A set of “operations”, which may be used like assignments, are used to model timer management. These operations can be applied to:

- a) a set of timers;
This is specified by following the timer operation with just the timer type name.
- b) an individual timer from a given set of timers of the same timer type.
This is specified by following the timer operation by the timer type name and an identifier for the individual timer. A timer identifier is a variable name.

Note – It is not necessary to declare timer identifiers explicitly. They are declared implicitly when stated as a parameter on the **Start** operation.

D.6.10.2 *Definition of timer operations*

The timer operations defined are:

- a) **Start** *<timer type>* [, *<timer id>* [, *<value>*]]
The **Start** operation is used to indicate that an inactive timer, or set of timers, should start running. (If the timer identifier parameter is omitted, this shall be interpreted as starting a timer of the given timer type. However, it is not then possible to manipulate this timer separately from any other timers of this type which may be defined.)
The optional value parameter shall be used if it is desired to assign an expiry time (i.e., duration) for a timer. This value is not required to be within the range of values defined for the timer. This provides the ability to test timer duration situations. Otherwise, any time within the range of values specified in the Declarations Part may be used.
- b) **Cancel** *<timer type>* [, *<timer id>*]
The **Cancel** operation is used to indicate that a running (or suspended) timer is to become inactive. (If the timer identifier parameter is omitted, all suspended timers of the given timer type shall become inactive.)
- c) **Suspend** *<timer type>* [, *<timer id>*]
The **Suspend** operation is used to indicate that a running timer is to become suspended. (If the timer identifier parameter is omitted, all timers of the given type shall become suspended.)
- d) **Resume** *<timer type>* [, *<timer id>*]
The **Resume** operation is used to indicate that a suspended timer is to become (i.e. resume) running. (If the timer identifier parameter is omitted, all timers of the given timer type shall resume running.)

D.6.10.3 *Timer pseudo-event*

In addition to the timer operations, two timer pseudo-events are defined.

- a) The **timeout** pseudo-event, which has the following form:
?Timeout *<timer type>* [, *<timer id>*]
This pseudo-event may be used within a behaviour tree to check for expiration of the specified timer. The timer identifier parameter may be omitted if:
 - 1) there is only one timer of type timer type defined;
 - 2) there are multiple timers of type timer identifier defined, and it is not necessary to distinguish between them.

D.6.10.4 *The elapse pseudo-event*

An **elapse** statement takes the following form:

Elapse *<timer type>* [, *<value>*]

An **elapse** statement puts an upper bound on the time in which a tree will remain at a given level of indentation (i.e. in a given state). When using **elapse** the following points should be noted:

- a) the **elapse** statement shall not have any Boolean expression or synchronization requirements associated with it;
- b) if more than one **elapse** is used then only the one which specifies the smallest time duration will be considered;
- c) it is recommended (but not mandatory) that the **elapse**, if any, be the last of a set of alternatives;

- d) the alternative specified by the **elapse** occurs only if, during the specified duration, no:
 - 1) other alternative is scheduled;
 - 2) error occurs (i.e. an even for which there is no alternative).

Note – Although **elapse** uses a timer-type for the convenience of giving a value and units, the **elapse** does not imply that any user accessible timer is started.

Example D-30 – Using **Elapse**:

```
?A
?B
ELAPSE
```

means that the tester will wait on the events ?A or ?B only until one of them occurs or until the ELAPSE expires.

D.6.11 Synchronization

The ability to specify the relative ordering of events in different trees, which may or may not be in the same behaviour description, is provided by the synchronization statement. The synchronization statement is a Boolean expression of the form:

- a) $(TreeReference_1)/(Label_1) < (TreeReference_2)/(Label_2)$ OR
- b) $(TreeReference_1)/(Label_1)_1 > (TreeReference_2)/(Label_2)$

where the predicates are to be interpreted as:

- a) the event labelled by $(Label_1)$ in test step $(TreeReference_1)$ occurs **before** the event labelled $(Label_2)$ in test step $(TreeReference_2)$;
- b) the event labelled by $(Label_1)$ in test step $(TreeReference_1)$ occurs **after** the event labelled $(Label_2)$ in test step $(TreeReference_2)$.

Note 1 – Synchronization is specified in terms of events in different trees rather than in terms of choices in the same tree.

Note 2 – Synchronization may only be used between parallel trees, which may, or may not, be part of the same behaviour description.

Note 3 – An alternative choice to synchronized parallel trees is one single tree dealing with several PCOs.

D.6.11.1 Interpreting synchronization statements

A synchronization statement shall be interpreted as follows:

IF

- a) an event among a set of alternatives has a label L AND,
- b) there exists a synchronization statement containing $(TreeReference/Label)$ in its right or left part, AND
- c) $(TreeReference)$ designates the behaviour description in which the event is

THEN

to the Boolean expression used to decide whether the event is selected append the following:

“AND the event designated by the label in the other part of the synchronization statement already occurred (respectively did not already occur)”.

Therefore a synchronization statement does not mean that the tree is “suspended” until the synchronization requirement is satisfied, but rather that the event involved in the synchronization is not a candidate for selection among the current set of alternatives if the synchronization is not satisfied. The normal rules for selecting an event apply, considering the synchronization as an extra Boolean expression. If no event can be selected then an error may occur if an event is present at the PCO which cannot be flow-controlled. The test writer shall therefore include the appropriate event at the same level of indentation.

Example D-31:

FIRST-TREE

LT! L-DTAreq [DT packet with D-bit set]	L1	
LT? L-DTAind [P(R) for this packet]	L2	fail ₁
LT? L-DTAind [P(R) < this packet] → L2		
LT? L-DTAind [P(R) < this packet]	L3	pass

SECOND-TREE

UT? N-DTAind	L	
--------------	---	--

with the synchronization requirements:

FIRST-TREE/L3 > SECOND-TREE/L

FIRST-TREE/L2 < SECOND-TREE/L

D.6.12 *The repeat construct*

This section describes a mechanism in TTCN behaviour descriptions to iterate a test step a variable number of times. The form of the construct is:

REPEAT (*TreeReference*) [(*Varid*)]

and it must be followed by a set of alternative Boolean expressions (guards). It should be noted that:

- a) the guards need not be mutually exclusive. The repeat terminates when at least one guard holds;
- b) no other event or pseudo-event may appear at the same level of indentation as the guards;
- c) the VARid (when specified)
 - 1) may be used within the repeated tree;
 - 2) may be used inside the guards;
 - 3) is not a global variable (i.e. its scope is local to the REPEAT);
 - 4) is implicitly of type Integer.

D.6.12.1 *Repeat with no VARid*

When VARid is not specified then the construct:

```
REPEAT TreeReference (a1, . . . , an)
  [b1]
  . . .
  .
  .
  .
  [bn]
  . . .
```

shall be interpreted as meaning:

```
+ TreeReference (a1, . . . , an)L
  [b1]
  . . .
  .
  .
  .
  [bn]
  . . .
  [true] → L
```

Where (a₁, . . . , a_n) are the actual tree parameters (if any).

D.6.12.2 REPEAT with VARid

When VARid is specified the construct:

REPEAT TreeReference (a_1, \dots, a_n) [VARid]

expands to:

+ repeat-tree ($a_1, \dots, a_n, 0$)

and this repeat-tree is defined as:

Dummy-repeat-tree ($f_1, \dots, f_n, \text{VARid}$)
+ treeReference ($f_1, \dots, f_n, \text{VARid}$) L
(VARid:=VARid+1)
[b₁]
...
.
.
.
[b_n]
...
[true] → L

Where (a_1, \dots, a_n) are the actual tree parameters (if any) and (f_1, \dots, f_n) are the formal tree parameters and tree-id (f_1, \dots, f_n) is redefined as tree-id ($f_1, \dots, f_n, \text{VARid}$)

D.6.13 Constraints reference

This column allows reference to be made to an ASP, ALP, or PDU that is defined in the Constraints declaration part (§ D.8 – which also provides more information on encoding constraints).

Example D-32 – A constraint reference:

N-SAP? N-DATAind [UserData ~ CR-TPDU] N-DATAind (D1), CR-TPRU (CR1)

Where N-DATAind and CR-TPDU are defined in the ASP and Data Type Declarations sections of the test suite. The constraints D1 and CR1 are defined in the constraints section of the test suite.

If an event is qualified by a Boolean expression and also has a Constraints Reference, this shall be interpreted as *the event occurs if, and only if, both the Boolean expression and the constraint hold*.

If an event is followed by an assignment clause and also has either or both a Constraints Reference or a Boolean expression, this shall be interpreted as *the assignment is performed if, and only if, the event occurs*, according to the definition given above.

D.6.14 Verdicts

A test clause shall end with a verdict. This verdict is given with respect to the test purpose, and may be one of:

- a) Pass;
- b) Fail₁;
- c) Fail₂ (test purpose accomplished but failed subsequently)
- d) Fail₃ (protocol error but test purpose inconclusive);
- e) Incon(clusive).

A test event may have a verdict assigned to it by using the verdict column. Entries in the verdict column will either contain one of the following or be left blank (i.e. no verdict). This particular case is referred to in the following as the special value *none*.

D.6.14.1 Rules for applying verdicts

The following rules for verdict assignment apply:

- a) at any time during the execution of a test case, there is a “default verdict”;

Note – Not to be confused with any verdict that may appear in a default behaviour description.

- b) at the beginning of a test case the default verdict is *none*;

- c) when a tree is attached the default verdict for the attached tree becomes the result of applying Figure D-28/X.290, Part 2. “Current” is the current default verdict and “New” is the verdict assigned at the point of attachment.
- d) when an event (other than attach) has a verdict assigned to it which is not “none” (i.e. the verdict column is not blank) the test case ends with the verdict being the result of applying Figure D-28/X.290, Part 2 to the default verdict and the verdict assigned to that event;
- e) when the tip of a tree that is not the main tree is reached the test case terminates with the verdict being the default verdict.

Note – In this case there is no explicit verdict assigned to the last event of the tree (otherwise § D.6.14.1 d) would apply). This is an error.

Example D-33 – The use of a verdict attached to a subtree:

A-TREE

!CONreq

?CONcnf

+ Data Transfer (“Hello”)

[ok = “yes”]

+ bye Pass

[ok = “no”]

+ bye Fail

?DISind

Inconc

	New					
Current	none	Pass	Fail ₁	Fail ₂	Fail ₃	Inconc
none	none	Pass	Fail ₁	Fail ₂	Fail ₃	Inconc
Pass	Pass	Pass	Fail ₂	–	–	–
Fail ₁	Fail ₁	–	Fail ₁	–	–	–
Fail ₂	Fail ₂	–	Fail ₂	Fail ₂	–	–
Fail ₃	Fail ₃	–	Fail ₁	–	Fail ₃	–
Inconc	Inconc	–	Fail ₃	–	–	Inconc

FIGURE D-28/X.290, Part 2

Default Verdict Precedence

D.6.15 *Comments*

This column contains short remarks, or references to extended comments given at the bottom of the table.

D.6.16 *Defaults reference*

In order to emphasize the main path through a test, it is possible to specify, separately from the main behaviour description, default subsequent behaviours for any event which a tester may receive. The main behaviour descriptions of the dynamic part are completed by appending every set of alternatives with subsequent behaviour from a default behaviour description.

In order to ensure that a test case specification is complete, a TTCN test specification shall specify subsequent behaviour for every event possible, either in a main behaviour description, or by means of an associated default behaviour description.

Note – This can be simply done with ?OTHERWISE.

Default behaviours only apply where the relevant test event is possible.

Example D-34 – The following test case could be split into two behaviour descriptions as:

EXAMPLE-TREE

```
!CONreq
?CONcnf
!DATreq
?DATind
!DISreq
```

with a trivial default (having no subsequent behaviour associated with it) of:

DEFAULT-TREE

```
?DISind
```

The Defaults Reference is used to associate a set of defaults behaviours with a main behaviour description.

D.6.17 *The specification of defaults*

The default behaviour descriptions shall be presented in tables in the Default section of the Dynamic Part of a test suite. These tables shall be of the format shown in Figure D-29/X.290, Part 2.

Default Dynamic Behaviour				
Reference: <Test Case or Test Step Reference> Identifier: <Identifier> Purpose: <Purpose> Defaults Reference: <Defaults Reference>				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
<Behaviour Description>	<Label>	<Constraints Reference>	<Verdict>	<Comments>
Extended Comments (optional)				

FIGURE D-29/X.290, Part 2
Dynamic Behaviour Declaration Proforma

These tables shall be entirely analogous to the main behaviour descriptions except for the following differences:

- a) the default behaviour tree shall not have a name;
- b) it shall contain a single unnamed tree; (/ Rightarrow PCOs and parameters cannot be passed to defaults);
- c) it may not contain a Synchronization Requirements entry.

Note – References to default dynamic behaviours obey the same rules as for test steps (§ D.3.3.2).

Example D-35 – Figures D-30/X.290, Part 2, D-31/X.290, Part 2, and D-32/X.290, Part 2, present a slightly augmented version of EXAMPLE-TREE. The default behaviours in Figures D-31/X.290, Part 2 and D-32/X.290, Part 2, in combination, are exactly equivalent to the behaviour shown in Figure D-33/X.290, Part 2.

Dynamic Behaviour				
Reference: Examples/BiggerExample Tree Identifier: BET Purpose: To illustrate the use of defaults Defaults Reference: Examples/Defaults/Standard				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
BIGGER-EXAMPLE-TREE !CONreq ?CONcnf !DATreq !DATind !DISreq			Pass	Request Confirm Send Data Accept Data Disconnect

FIGURE D-30/X.290, Part 2
 Default Behaviour Example (Part 1 of 3)

Default Dynamic Behaviour				
Reference: Examples/Defaults/Standard Identifier: Std Purpose: First Level of Defaults Defaults Reference: Examples/Defaults/Default				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
?RSTind !DISreq			Inconc	Reset Finish

FIGURE D-31/X.290, Part 2
 Default Behaviour Example (Part 2 of 3)

Default Dynamic Behaviour				
Reference: Examples/Defaults/Default Identifier: Def Purpose: Second Level of Defaults Defaults Reference: None.				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
?DISind			Inconc	Premature

FIGURE D-32/X.290, Part 2
Default Behaviour Example (Part 3 of 3)

Default Dynamic Behaviour				
Reference: Examples/Defaults/Standard Identifier: Std Purpose: Combined Defaults Defaults Reference: None				
Behaviour Description	Label	Constraints Reference	Verdict	Comments
?RSTind				Reset
!DISreq			Inconc	Finish
?DISind			Inconc	Finish
?DISind			Inconc	Premature

FIGURE D-33/X.290, Part 2
Default Behaviour Example (Parts 2 and 3 combined)

D.7 *Interpreting trees*

This section describes how to interpret TTCN behaviour descriptions and how to assign a verdict to a test case. It consists of two parts intended for different audiences:

- a) the first part provides a set of rules together with a simple algorithm intended to aid the understanding of TTCN;
- b) the second part provides a more comprehensive algorithm (the rules have been integrated into the algorithm) intended to help implementing TTCN.

D.7.1 *Interpretation of TTCN (partial algorithm)*

- a) a TTCN tree has a number of levels of event hierarchy that is greater than or equal to one. Execution of the events in the tree will terminate at some level by:
 - 1) EITHER a verdict being assigned;
 - 2) OR the occurrence of a test case error (i.e. a tip with no verdict is reached or an unexpected event is received at a PCO);
- b) each level has a number of alternative events greater or equal to one;
- c) we assume here that:
 - 1) the tree expansion has already been done according to the general rules given in § D.6.5;
 - 2) the set of defaults to be applied has been computed according to the rules given in § D.6.14.1;
 - 3) the rules for default verdicts in the case of a subtree are given in § D.6.14.1;
- d) the algorithm to interpret what to do at a given level is as follows:
 - 1) append the defaults to the set of alternatives;
 - 2) use the procedure in § D.7.3 to determine whether a match occurs;
 - 3) if the procedure in § D.7.3 returns *no match* then TERMINATE (error);
 - 4) if a match is found then:
 - A) if the event which causes the match has a verdict assigned to it, or if it is the tip of a subtree with a default verdict, then terminate the test case and return the final verdict determined according to Figure D-29/X.290, Part 2;
 - B) otherwise determine the next level of indentation as the one following the event which caused the match and:
 - i) if that level designates a non-empty set of alternatives, then restart processing for that level;
 - ii) otherwise TERMINATE (error).

D.7.2 *Rules for default*

- a) Defaults of a subtree take precedence over (i.e. are to be considered before) those defaults of the tree which attached the subtree.

Example D-36 – Defaults in both main and attached tree:

MAIN-TREE

?A

+ Step-1

DEFAULT-TREE

?B

fail₁

?E

fail₁

Step-1

?C

?D

Default-Step-1

?B

pass

and the sequences:

?A ?C ?B => pass

?A ?C ?E => fail₁

- b) defaults of a subtree are considered only when the subtree is actually entered (i.e. at a level of indentation greater than those of the events causing entry into that subtree);

- c) defaults cease to apply at the end of a tree and are not to be appended to the “empty alternative set” which follows a tip of the tree. This is true for the main tree as well as the subtree.

Example D-37:

MAIN-TREE

```
?A
  ?B
  ?C                               fail1
?OTHERWISE                         pass
```

DEFAULT-TREE

```
?OTHERWISE                         inconc
```

and the sequence ?A ?B ?ANYTHING is not valid (i.e. the test case is incomplete);

- d) when defaults are used in a subtree and an event occurs which matches the default, if the sequence specified by the default does not have a verdict, the execution will continue in the main tree after a tip of the default is reached, except, of course, if the subtree has a default verdict!

Example D-38 – Default tree with verdicts:

MAIN-TREE

```
?A
  + Step-1
  ?X                               pass
?B                               fail1
```

Step-1

```
?D
  ?E
```

Default

```
?B
  !C
```

and the sequence ?A ?D ?B ?C ?X => pass

D.7.3 Matching events

The rules for determining if an event among a set of alternatives matches (i.e. evaluates as having occurred) are listed below for each possible type of TTCN event or pseudo-event:

- !(ASPid | ALPid | PDUid)** – for the SEND event to match, it must be possible to send the ASP, ALP or PDU (e.g. considering flow control). If a PCO has been specified this determination is made with consideration to that particular PCO. Additionally all qualifying Boolean expressions and/or constraints placed on the ASP, ALSP or PDU must hold;
- ?(ASPid | ALPid | PDUid)** – for the RECEIVE event to match the specified ASP, ALP or PDU must have been received and all specified constraint must hold. If a PCO has been specified the ASP, ALP or PDU must have been received at that particular PCO. Additionally all qualifying Boolean expressions and/or constraints placed on the ASP, ALSP or PDU must hold;
- ?TIMEOUT** – for the Timeout pseudo-event to match, a timer of the named type must have expired. If an optional timer identifier is specified, then the expiration of only that individual timer is considered;
- ELAPSE** – for the Elapse event to match, the timer which was implicitly started (by the specification of the Elapse) must have expired before any of the events prior to the Elapse have matched;
- ?OTHERWISE** – for the Otherwise event to match, some event must have been received which has not matched any previous alternatives to the Otherwise. If a PCO is specified then the event must have been received at that particular PCO;
- ATTACH** – this event will never be considered for matching because all tree attachments are assumed to have been fully expanded before attempting to match an alternative;
- REPEAT** – this event will never be considered for matching because it will have previously been expanded;
- GOTO** – this event always evaluates to TRUE, and therefore always matches;

- i) **BOOLEAN EXPRESSIONS** – a Boolean expression stated as an alternative matches if the expression evaluates to TRUE (see § D.6.7.2);
- j) **ASSIGNMENT CLAUSES** – an assignment clause always matched;
- k) **PARALLEL TREES** – the execution of parallel trees will never be considered for matching because these will have been fully expanded before attempting to match an alternative;
- l) **TIMER OPS Start, Cancel, Suspend and Resume** – all timer operations will always match.

Note – Any events specified following and at the same level of indentation as a goto, assignment clause or a timer operation can never be reached.

D.7.4 *Comprehensive algorithm*

For the purposes of this section:

- a) a verdict may be one of the following:
 - 1) pass, fail₁, fail₂, fail₃, inconc;
 - 2) none, which is used:
 - A) as a special value returned when a tip of a sub-tree is reached without a verdict;
 - B) as a default verdict value when attaching a sub-tree for which there is no default value;
 - C) as the verdict assigned to an event when the verdict column is empty;
 - 3) an ordered union of an ordered set is . . . (append?);
- b) let EVALUATE-DEFAULT (R) be a function:
 - 1) of a behaviour reference (R);
 - 2) which returns an ordered set of alternative events;
 - 3) this ordered set is defined as follows:
 - A) let d be the reference of the default behaviour associated with the behaviour description referenced to by R;
 - B) if there is no d return the empty set;
 - C) otherwise return the ordered union of:
 - i) all the events at the first level of indentation in the behaviour description referenced to by d;
 - ii) and evaluate EVALUATE-DEFAULT (d).
- c) let SELECT (E) be a function:
 - 1) of an ordered set E of alternative events;
 - 2) which returns an event of its input set, or the special value error;
 - 3) which is defined as follows:
 - A) let E' be a set of 2-uple {event, event} initialized as:

$$\{ \{ e_1, e_1 \}, \{ e_2, e_2 \} \dots \{ e_n, e_n \} \}$$
 where $e_1 \dots e_n$ are the elements of E;
 - B) replace any element of E' of the form:

$$\{ +tree, e \}, \dots \forall e \text{ by } \{ t_1, e \} \dots \{ t_n, e \}$$
 where $t_1, \dots t_n$ are the events at the first level of indentation of the tree;

Note 1 – This has to be done (possibly recursively) until no element is of the form { +tree, e } since a tree may attach a subtree at its first level of indentation.

Note 2 – The second element of the 2-uple keeps track of the original event name in the ??? of ???, SELECT will return this element.

Note 3 – E' is an ordered set.
 - C) if there is any **ELAPSE** among the alternatives then compute T = time-unit (elapse value) otherwise set T = infinity;
 - D) for each element in { x, e } in E' do the following:

IF the following holds:

 - i) IF there is a Boolean expression and it holds

AND

- EITHER x is PCO? Otherwise and any ASP event is ready at that PCO;
- OR x is PCO? ASP and the ASP is ready and any Boolean expression using parameters of this ASP holds and all the constraints (if any) are satisfied;
- OR x is PCO! ASP and ASP can be sent at this PCO (i.e. flow control permits sending) and all constraints (if any) are satisfied;
- OR x is neither ? nor !.

THEN SELECT terminates and returns the event.

- E) if there is any ASP ready at any PCO and this ASP cannot be flow-controlled, or there is an elapsed timer, SELECT terminates and returns “error”;
- F) otherwise
- i) if a period of time more than T has elapsed since c(3)D was entered for the first time, then select the event “elapse” which was used to compute T;
 - ii) otherwise restart in c(3)D;
- i.e. wait for the next event (busy loop);
- d) let EXECUTE(x) be a function
- 1) of an event x
 - 2) which terminates without returning a value
 - 3) is defined as follows:
 - A) if x is ? OTHERWISE, then discard the actual event (?ASP) which caused the SELECT function to return, and then;
 - B) if x is PCO? ASP, take the ASP from the PCO, bind any related parameters or free variables to the received values; and then;
 - C) bind any unbound global variables so that the Boolean expression holds (if any); and then;
 - D) if x is PCO! ASP, bind the parameters to values so that the Boolean expression holds and issue the ASP at the PCO; and then;
 - E) perform any assignments (including any timer operation);
- e) let EVAL-TREE be a function:
- 1) of the following:
 - the current level of indentation (L) which designates a set of alternative events in a behaviour description;
 - the current default verdict (V) which indicates in the case of tree attachment whether a verdict was assigned at the point of attachment of the tree.
Note – none means no verdict.
 - the ordered set of default event (D) to be used to interpret the tree;
 - 2) EVAL-TREE returns a value as defined in a;
 - 3) EVAL-TREE is defined as:
 - A) evaluate K = set of events as L;
 - B) invokes SELECT(K U D);
 - C) if e(3)B returns error, then EVAL-TREE TERMINATES(error);
 - D) if e(3)B returns an event of the form + tree then:
 - i) compute V = EVAL-TREE (L', V', D') where
 - L' designates the first level of indentation in the attached tree
 - V' is the result of applying Figure D-29/X.290, Part 2 and the verdict assigned to the tree
 - D' is the ordered union of EVAL-DEFAULT (tree-ref) and D
 - ii) if V ≠ none, then return V
 - E) otherwise:
 - invoke EXECUTE(e) where e is the event as returned by e(3)B;
 - if e has verdict compute V as the result of applying Figure D-29/X.290, Part 2 and the verdict assigned to the event, return V;
 - F) compute L = next level of indentation, taking into account any GOTO;

- G) if L designates an empty set then terminate and return V, otherwise continue with e(3)A.
- 4) the interpretation of a test case is given by EVAL-TREE (L, none, D) where:
 - L = the first level of indentation of the first tree in the behaviour description of the test case;
 - D = EVALUATE-DEFAULT (test-case-reference);
 - A) if EVAL-TREE returns “error” then the specification is incomplete;
 - B) if result is “none” then the test case specification is incomplete;
 - C) otherwise EVAL-TREE returns the verdict.

D.8 *Constraints declaration part*

It is necessary to describe, in detail, the coding of parameters in PDUs and ASPs. The parameter coding is described using either a tabular method (§ D.8.1.1), or a method which takes advantage of the ASN-1 notation (§ D.8.3). Reference to particular values is made in the Constraints Reference column of the tables used in the Dynamic Part (§ D.6.13).

D.8.1 *Parameter encodings*

An ASP or PDU can be considered as a list of parameters. However, because of the importance of PDU encoding in testing, it may be more appropriate not to abstract away from PDU encoding and to consider a PDU as a list of fields such as Length, Parameter Type, Parameter Value.

D.8.1.1 *Tabular method*

If an actual value is assigned to each field, the actual PDU or ASP can be represented by a list of values. Secondly, a set of lists can be built, each element of this set being a possible list of values from all the combinations. Then, for each PDU or ASP, this set can be represented by a table. The PDU constraints declaration table shall have the proforma shown in Figure D-34/X.290, Part 2.

PDU Constraint		
PDU Name: <PDU Name>	Constraint Name: <Constraint Name>	
Field Name	Value	Comments
.	.	.
.	.	.
.	.	.
<Field Name>	<Value>	<Comment>
.	.	.
.	.	.
.	.	.

FIGURE D-34/X.290, Part 2
PDU Constraint Proforma

Note – In the following text there is a corresponding ASP or ALP table for every occurrence of a PDU table. The ASP or ALP proforma is similar in every detail except that every occurrence of PDU in the table fields substitute the word ASP or ALP.

Each field entry in the field name column shall have been declared in the PDU (or ASP, ALP) declaration. Values assigned to each field shall be of the type specified in the PDU (or ASP, ALP) declaration.

In cases where a constraint contains only a few fields, or when there are only a small number of constraints, the compacted version of the constraints table proforma shown in Figure D-35/X.290, Part 2 may be used.

PDU Constraints				
PDU Name: <Name>				
Constraint Name	Field Name			Comments
	<Field Name> ₁	...	<Field Name> _n	
<Constraint Name> ₁	<Value> _{1,1}	...	<Value> _{1,n}	<Comment> ₁
<Constraint Name> ₂	<Value> _{2,1}	...	<Value> _{2,n}	<Comment> ₂
		...		
<Constraint Name> _m	<Value> _{m,1}	...	<Value> _{m,n}	<Comment> _m

FIGURE D-35/X.290, Part 2
Compact Constraints Table Proforma

Example D-39 – Shows field names across the top of the table, and different instances of the PDU in rows within the table. However, where there are too many fields to fit conveniently across a page, it is permitted to transpose the table.

Example D-39 – Given a PDU X, with two fields P1 and P2, if the possible values for P1 and P2 are 0 and 1, then we have the table shown in Figure D-36/X.290, Part 2.

The Constraints Reference columns of in the Dynamic Part might then contain entries such as X(S1) and X(S4).

D.8.1.2 *Generic values*

Constraints may be parameterized using generic values. These shall be presented in the format shown in Figure D-37/X.290, Part 2. Again, an alternative compact version of the format may be used, this is given in Figure D-38/X.290, Part 2.

PDU Constraints			
PDU Name: X			
List Name	Field Name		Comments
	P1	P2	
S1	0	0	
S2	0	1	
S3	1	0	
S4	1	1	

FIGURE D-36/X.290, Part 2
Two field PDU example

Generic Value		
PDU Name: <PDU Name>		Generic Name: <Generic Name>
Field Name	Value	Comments
.	.	.
.	.	.
.	.	.
<Field Name>	<Value>	<Comment>
.	.	.
.	.	.
.	.	.

FIGURE D-37/X.290, Part 2
PDU Constraint Proforma

Generic Value				
PDU Name: <PDU Name>		Generic Name: <Generic Name>		
List Name	Field Name			Comments
	<Field Name> ₁	...	<Field Name> _n	
<Constraint Name> ₁	<Value> _{1,1}	...	<Value> _{1,n}	<Comment> ₁
<Constraint Name> ₂	<Value> _{2,1}	...	<Value> _{2,n}	<Comment> ₂
		...		
<Constraint Name> _m	<Value> _{m,1}	...	<Value> _{m,n}	<Comment> _m

FIGURE D-38/X.290, Part 2
Compact Generic Constraint Proforma

These concepts are illustrated in the Examples D-39 and D-40.

Example D-40 – The invocation of the PDU X (Figure D-39/X.290, Part 2) in a test step may be made as follows: X(S1), X(S2), X(S3), X(S4), X(S5(0)), X(S5(1)) or X(S5(Var)), where Var is either a variable (see § D.5.6 and D.8.2) or a field name of a received PDU (§ D.5.9) or an expression.

PDU Constraints			
PDU Name: X			
List Name	Field Name		Comments
	P1	P2	
S1	0	0	
S2	0	1	
S3	1	0	
S4	1	1	
S5(A)	1	A	

FIGURE D-39/X.290, Part 2
Simple Parameterized Constraint example

Example D-41 – The way of defining generic values illustrated in Example D-40 can be extended to a group of fields:

Given the PDU Y with three fields Q1, Q2 and Q3 with 0 and 1 as possible values, the table can be represented as shown in Figures D-40/X.290, Part 2, D-41/X.290, Part 2, D-42/X.290, Part 2.

PDU Constraint				
PDU Name: Y				
List Name	Field Name			Comments
	Q1	Q2	Q3	
T1	0	0	0	
T2(B1)	1	B1		
T3(B2)	B2		0	

FIGURE D-40/X.290, Part 2
Parameterized Constraint example (Part 1 of 3)

Generic Value			
PDU Name: Y		Generic Name: B1	
Constraint Name	Field Name		Comments
	Q3	Q4	
C1	0	0	
C2	0	1	

FIGURE D-41/X.290, Part 2
Parameterized Constraint example (Part 2 of 3)

Valeur générique			
Nom de la PDU: Y	Nom générique: B2		
Nom de la contrainte	Nom du champ		Commentaire
	Q1	Q2	
D1	0	0	
D2	1	1	
D3	1	0	

FIGURE D-42/X.290, partie 2
Exemple de contraintes paramétrées (partie 3 de 3)

The invocation of the PDU Y in a test step is then made as follows: Y(T1), Y(T2(C1)), (T2(C2)), (Y(T2(D1))), etc.

Tables defining generic values shall not be referred to directly in the main behaviour part, but only via a parameterized constraint name (e.g., B1 is used as Y(T2(B1)) in Example D-41).

D.8.2 Free variables

D.8.2.1 Introduction

In many cases it is necessary to use the values received in a PDU (e.g., SRC-REF in a CC) for filling a corresponding field when sending another PDU (e.g., DST-REF in DT). Although this can be achieved by keeping track of the value in a variable from the dynamic part, it may also be convenient to represent this in a more concise way by using a Free Variable in the constraints part. Such a variable shall be declared at the beginning of the constraints part when declaring constraints. Examples D-40 and D-41 are not examples of free variables. In this case, one can represent that value by a free variable which is considered to be global to the Constraints Part.

These free variables shall be declared using the proforma shown in Figure D-43/X.290, Part 2.

A variable is bound to a value when a PDU in which it appears occurs as a parameter to a received ASP.

D.8.2.3 Variables in constraints

Whenever a constraint applies, any variable appearing in the field of a constraint either has, or takes on, the actual value sent or received.

- a) If the variable is already bound to a value, either
 - 1) that value is sent; or
 - 2) that value must be the value received in that field if the constraint is to apply. (Otherwise the constraint does not apply.)
- b) If the variable is unbound, either
 - 1) any value of the appropriate type may be sent; or
 - 2) the variable becomes bound to the received value as a result of applying the constraint.

A typical application of this feature might be to preserve addresses, or connection references from PDU to PDU. These may not be of interest in most test cases, but it is important that the same value is present in several different PDUs.

Free Variables		
Name	Type	Comments
.	.	.
.	.	.
.	.	.
<Name>	<Type>	<Comments>
.	.	.
.	.	.
.	.	.

FIGURE D-43/X.290, Part 2
Free Variables Proforma

D.8.2.4 Conventions

The conventions shown in Table D-1/X.290, Part 2 apply to constraint table entries.

TABLE D-1/X.290, Part 2
Constraint Table Entry Conventions

Direction	Field or Parameter Value	
Initiated by Tester (!)	—	?
Received by Tester (?)	—	?

Note -- A test case writer should be aware that if an ASN-1 parameter has a default value, then according to Recommendation X.209, an IUT is entitled to encode the default value explicitly, or to omit the parameter from the encoding.

D.8.2.4.1 Pattern matching in a Character String

Inside a character string, a ? in place of a character means that any single character is accepted. When the ? symbol itself is needed within the character string, this shall be indicated by preceding the ? with the special character \. The character \ itself is written \.

Inside a character string a * means that none, or any number, of characters is acceptable. This * shall match the longest sequence of characters possible, according to the pattern as specified by the symbols surrounding the *. (When the * symbol itself is needed within the character string, this shall be indicated by preceding the * with the special character \.)

D.8.3 ASN-1 modular method

D.8.3.1 Scope of method

This method assumes that ASN-1 descriptions of the PDUs or ASP parameters are available, and takes them as a convenient basis for constraint description.

D.8.3.2 Description of the ASN-1 modular method

When a PDU or an ASP parameter is defined in the declaration part using ASN-1, the ASN-1 modular method is applicable. This method provides the following major facilities:

- a) **Specification:** the (partial) specification of ASN-1 values. Typically, there will be correct PDUs, but arbitrary ASN-1 values may be constructed and used.
Note – ASN-1 requires explicit values, whereas the modular method allows “don't care”.
- b) **Naming:** these values may be named, so that they can be referenced from the dynamic part.
- c) **Annotation:** values may be annotated with an extended form of ASN-1 type notation. Value encodings may be specified so that two identical values encoded differently may be distinguished. Constraints may be placed on components of values through the use of free variables, or the – and ? conventions, just as constraints can be placed on field values in the tabular method.
- d) **Replacement:** new values may be constructed from old ones by replacing components of existing values with new values.

D.8.3.3 Value specification

When specifying an ASN-1 value the following auxiliary information shall be provided:

- a) A name.

This is required to identify the value.

When specifying large, complex ASN-1 values, it is sometimes convenient to be able to place constraints on selected components of a value only. Such components can be defined by tracing a path through the ASN-1 type declaration corresponding to the value, identifying elements at each level by name and using a period to separate each element name.

- b) The ASN-1 type of the value.

This is required for documentation purposes so that the ASN-1 value can be related to an appropriate ASN-1 type definition to be found in the relevant protocol standard.

In order to make an ASN-1 type definition suitable for documenting ASN-1 value definitions, a number of abbreviations and restrictions to the ASN-1 type notation are required. These are enumerated below. Some of these represent changes to the grammar to be found in (Recommendation X.208, Annex F).

The following notation is used:

$\langle NonTerminal \rangle ::=$

...
| $\langle NewProds \rangle$

where $\langle NonTerminal \rangle$ is defined in (Recommendation X.208, Annex F) and ... represents that definition. $\langle NewProds \rangle$ represents the TTCN extensions.

- 1) It is permitted to omit the $\langle Type \rangle$ from an $\langle identifier \rangle \langle Type \rangle$ combination. The following productions are added to (Recommendation X.208, Annex F):

$\langle ElementType \rangle ::=$

...
| $\langle TTCNElementType \rangle$

$\langle TTCNElementType \rangle ::=$
 $\langle TTCNNamedType \rangle$

$\langle TTCNNamedType \rangle ::=$
 $\langle identifier \rangle$
| $\langle NamedType \rangle$

- 2) In a *<ChoiceType>*, only the type of the choice made shall be used. However, in order to remind the reader that a choice has been made, the CHOICE keyword may be retained. The following production replaces the production for *<ChoiceType>* in (Recommendation X.208, Annex F):

```
<ChoiceType> ::=
    <TTCNNamedType>
    | CHOICE <TTCNNamedType>
```

- 3) The *<SelectionType>* shall not be used. Instead, the actual type selected shall be used. *<SelectionType>* is removed from the list of possible *<BuiltinTypes>* in (Recommendation X.208, Annex F).
- 4) The *<SequenceOfType>* is changed to permit the naming of the actual values to be sent out. The following production replaces the production for *<SequenceOfType>* in (Recommendation X.208, Annex F):

```
<SequenceOfType> ::= SEQUENCE OF
    { <SequenceOfTypeList> }
<SequenceOfTypeList> ::=
    ...
    | <SequenceOfTypeList>
    <SequenceOfTypeType>
<SequenceOfTypeType> ::=
    <Type>.<number>
```

All the Types appearing in a *<SequenceOfTypeList>* shall be identical. The numbers in a *<SequenceOfTypeList>* shall be distinct. They are used to index the Types.

- c) A specification of encoding constraints on the value.

If any legal encoding of the value is acceptable, the specification of encoding constraints for the value may be omitted. Otherwise, the encoding specification shall consist of two parts:

- 1) A specification of the *<Identifier>* appearing in the encoding of the value. The *<Identifier>* shall consist of a whole number of octets.

```
<Identifier> ::= [ <IdentifierValue> ]
<IdentifierValue> ::=
    <hstring>
    | <bstring>
    | <variable>
```

It is not necessary for the *<IdentifierValue>* to be correct with respect to the ASN-1 type specification. This means that incorrectly encoded PDUs may be specified.

- 2) A specification of the *<Length>*, to be used in the encoding of the value. The *<Length>* shall consist of a whole number of octets.

```
<Length> ::= [ <LengthValue> ]
<LengthValue> ::=
    | <hstring>
    | <bstring>
    | <variable>
    | LI
    | SD
    | LD
    | LD <number>
    | IN
```

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

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

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

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

3) A specification of the value itself.

```

<Value> ::= [ <TTCNNamedValue> ]
<TTCNNamedValue> ::=
    | <NamedValue>
    | <valuereference>
    | ?
    | -
    | <in> <identifier>1
      REPLACE <identifier>2
      BY <TTCNNamedValue>
<in> ::= IN | <empty>
<BuiltinValue> ::=
    ...
    | <TTCNNamedValue>

```

Values are specified using the standard ASN-1 value notation with the following extensions:

- A) Values defined in either the Test Suite Parameters § D.5.4, or elsewhere in the Constraints Part may be referenced.
- B) <CharacterString> values may be denoted by enclosing the string in double quotes. Within those quotes a pair of consecutive double quotes shall be used to denote the double quote itself.
- C) A ? is used to specify “don't care” values. A ? appearing in a value that is to be sent by the tester may take any value that is legal in that context (according to the relevant service or protocol standard). A tester need make no verification of received values specified as ?.
- D) A – is used to specify that the encoding of the value shall be absent.
- E) The replacement operation provides the ability to substitute ASN-1 values (and their corresponding type and encoding constraints) for components of existing values, thereby producing new values, with the meaning *replace the component named <identifier>₁ of the value <identifier>₁ with the <TTCNNamedValue>*.

This information shall be provided in the format shown in Figure D-44/X.290, Part 2. The drawing of the box lines is optional in this case.

ASN-1 Value Constraint Declaration				
ASN-1 Type	Identifier	Value	Value	Comments
.
.
.
<ASN-1 Value Constraint Name> <ASN-1 type>	<Identifier>	<Length>	<Value>	<Comments>
.
.
.

FIGURE D-44/X.290, Part 2
ASN-1 Value Constraint Declaration Proforma

Example D-42 – This example shows how to reference PDU values from the dynamic part. We define a PDU called APDU:

```
APDU ::= SEQUENCE {  
    INTEGER  
    BOOLEAN  
}
```

and the global variable N. An instance of APDU might be:

```
Expected PDU ::= SEQUENCE {  
    INTEGER [10] [LI] N  
    BOOLEAN [ID] [LI] TRUE  
}
```

Then in the behaviour description we can write:

```
?DATAind [(UserData ExpectedPDU) AND (N = 27)]
```

APPENDIX I

(to Recommendation X.290, Part 1)

Applicability of the test methods to OSI* protocols

I.1 *Introduction*

Physical layer and media access control protocols are outside the scope of this Recommendation. None of the four methods defined in this Recommendation is intended to be applicable to these protocols.

I.2 *Data link protocols*

For testing data link protocols, the following points should be considered:

- a) The local single-layer test method is applicable only if the physical layer boundary of the IUT is accessible. In practice this is unlikely to be the case, except when testing a prototype or the design of an algorithm before it is put into hardware or firmware.
- b) Since the physical service is not end-to-end, the distributed, coordinated and remote single-layer test methods are applicable only when the external lower tester is connected to the SUT by a single link rather than across a network.
- c) The distributed, coordinated and remote single-layer test methods are applicable only if a lower tester can be realized with control over physical service primitives (or perhaps more realistically physical and data link PDUs). This may be difficult for some types of subnetwork.
- d) Most protocols for real subnetworks were defined before physical or data link services were thought of. Hence, service boundaries below the network service are generally either unavailable or not well-defined. Hence, it is generally felt that the control and observation for data link layer testing needs to be interpreted in terms of PDUs rather than ASPs. However, there may be some systems which do provide a data link service boundary or similar technology-dependent interface.

If single-layer testing of a data link protocol is not possible, multi-layer or single-layer embedded methods should be considered.

I.3 *Network protocols*

For network protocols, the test methods to be used are dependent upon whether the IUT is an end-system or an open relay system.

Since services below the network service are generally either unavailable or not well-defined, it is generally felt that the control and observation for network layer testing needs to be interpreted in terms of PDUs rather than ASPs.

It should be recognized that with some subnetwork technologies there are more than three protocols required to provide the network service. Each of these protocols may be tested separately or in any combination of adjacent protocols.

Considering the layer as a whole, both network and data link abstract service primitives are controllable and observable. Thus, for end-systems, all four single-layer (non-embedded) test methods are applicable, but since the data link service is not end-to-end, the lower tester has to be connected to the SUT over a single link for the distributed, coordinated and remote single-layer test methods.

Both the loop-back and transvers test methods are applicable to testing network relay systems.

I.4 *Transport protocol*

All the abstract test methods defined in this Recommendation are applicable to transport protocol conformance testing.

I.5 *Session protocol*

All the abstract test methods defined in this Recommendation are applicable to session protocol conformance testing.

For a large group of systems it will be appropriate to test the session protocol in combination with presentation and application protocols. Testing of session protocol should, therefore, normally be done in one of the two following ways:

- a) as a single-layer implementation or in combination with underlying protocols, in order to test the provision of a general purpose session service capable of supporting several different applications; the distributed or coordinated single-layer test methods are likely to be appropriate;
- b) in combination with presentation and application protocols, in order to test it in a specific application context; the remote or distributed single-layer embedded test methods are likely to be appropriate.

I.6 *Presentation and application protocols*

I.6.1 *General comments*

Conformance tests can be specified abstractly in terms of service primitives, irrespective of whether there is any notion of a service access point associated with them. Thus, provided that there is some mapping between application service primitives and real effects which can be observed and/or controlled, then tests can be specified in terms of those application service primitives. The observation and control of the service primitives may be indirect because of the nature of the mapping onto real effects, but as long as that mapping is possible then tests specified in these terms can be run. It is hard to find an example of an application service primitive which can never have a realization that can be observed and controlled; indeed if there were any, it seems likely that they ought not to be defined as primitives at all.

It is accepted that, in some circumstances, application protocol standards or recommendations may specify requirements on real effects which have to be achieved as the result of protocol exchanges (in particular this is evident in Job Transfer and Manipulation (JTM)). However, these requirements on real effects should be kept quite distinct from the normal protocol conformance requirements, possibly even in separate (perhaps functional) standards or recommendations. Some of these “non-protocol” conformance requirements might be testable by means of the general purpose abstract test methods defined in this Recommendation, but in general they will require application-specific test methods which fall outside the scope of this Recommendation.

I.6.2 *Association control*

Association control is unusual in that it involves 3 phases, the second of which is defined by another ASE. Association control service primitives could be observed and controlled in some systems. If so, it would be possible to test the association control protocol (possibly in combination with the presentation protocol) in isolation from any other ASE. The isolated testing of association control would have to use a dummy ASE for testing purposes. Any of the test methods could be used, including the coordinated method with a test management protocol defined as a dummy ASE. However, such isolated testing is of limited value as it could only test the protocol machine, leaving untested the mapping between abstract and transfer syntaxes. This aspect can only be tested for a particular ASE. The use of a dummy ASE in testing association control would not in general involve the syntax mappings that other real ASEs would. Therefore, priority should be given to testing association control in combination with the presentation protocol and

another real ASE, using the remote or distributed single-layer embedded test methods. In other words, the emphasis should be on testing application context operations.

The association control protocol has no non-protocol conformance requirements.

I.6.3 *Virtual terminal (VT)*

Virtual terminal protocol can be used in three ways: terminal to VT host, terminal to terminal, and VT host to VT host. In all three cases the VT service primitives could be observable and controllable in a large enough proportion of systems to make their use in test specification realistic. VT hosts may provide VT service interfaces in order to permit implementation of arbitrary user processes, so such an interface can be used by a realization of an upper tester. Terminals, on the other hand, are most likely to provide control and observation of VT service primitives only indirectly through a user interface which may be very different from the VT service. Some terminals might, in addition, provide direct access to a VT service interface.

Hence, the remote and distributed single-layer test methods are applicable to VT protocol testing.

I.6.4 *File transfer access and management (FTAM)*

In practice, observation and control of service primitives is different for FTAM initiators and responders. In general, they can be observed and controlled when testing initiators, but not when testing responders. The problem with FTAM responders is that there will be real effects associated with their service primitives, but the observation and control of these will be highly system specific. This is because although the “user” of the FTAM service in a responder is the virtual filestore, within the system under test effects on the VFS can only be seen through the real filestore. Thus, the remote single-layer test method is applicable to both FTAM initiators and responders, but the distributed single-layer test method is applicable only to FTAM initiators.

The testing of file management can be carried out on its own if necessary and does not depend on which of the read and write functional units are implemented. The testing of file transfer and access does, however, depend on which of these functional units are implemented.

The testing of file transfer and access for read-only systems or read/write systems is relatively straightforward. Read/write systems can be tested by transferring files to the system and then reading them back again. Although there is in general nothing to prevent local modification of such files between reading and writing, such local modification could be prevented for the duration of the test suite execution, by keeping other users off the system. Read-only systems can be tested by pre-loading the required set of files by local means and then running tests to read them using FTAM. In the testing of write-only systems, the assignment of verdicts to outcomes has to rely on a highly system specific (possibly subjective) interpretation of what the outcome (i.e. detailed record of observed test events) is.

Another way of viewing this situation is to recognize that FTAM has non-protocol conformance requirements concerned with the real effects which should occur as the result of PDU exchanges. These requirements can be tested by general test methods provided that the methods involve observation of FTAM service primitives (i.e. the test execution involves observation of the real effects associated with such primitives). In order for the real effects to be observed during testing, the test laboratory will need to be supplied with a detailed description of the mapping of service primitives onto real effects. This amounts to requiring the whole real filestore definition. This information should therefore be referenced by the PIXIT, being more information than one would want to include explicitly in the PIXIT or in the PICS.

There is a conformance requirement that, for the purposes of testing, an FTAM implementation shall be usable without the private use or legal qualification attributes. Nevertheless, there are other attributes which are difficult or practically impossible to test. For example, data protection and security attributes come in this category, because the information needed to test them will probably not be available. Also the ability to use the storage attributes to lock a file against non-OSI access will probably be impossible to verify, because the fact that the test laboratory cannot access such a file says nothing about the capabilities of a user with greater access permissions.

I.6.5 *Job transfer and manipulation (JTM)*

The situation regarding observation and control of service primitives is similar to that for FTAM. For JTM initiators, observation and control of service primitives is likely to be possible, and thus both the remote and distributed single-layer test methods are applicable. For JTM responders, the distributed single-layer test method can be used only if the real effects associated with the service primitives can be observed. This will involve knowing a mapping which may be complex. However, the JTM protocol standard places requirements on the implementor to state what these effects are and to make certain information human readable. Typically these effects are concerned with various documents being created and moved around. Thus, it may be possible to use the distributed single-layer test method for JTM initiators as well as the remote single-layer test method.

Status/enquiry primitives are easily testable. Reports on progress of a job can be observed.

A multi-system test method is needed to test major aspects of JTM. All but one system could be (lower) testers, and several logically distinct (lower) testers could be realized in the same real system. Nevertheless, there needs to be more than one external (lower) tester from an abstract point of view. For example, tests of failure situations need to cover the situation in which one system crashes and the remaining systems are expected to recover. It may also be desirable to test a collection of systems rather than a single system. Thus, at a minimum, there could be two new sets of test methods, one involving a single SUT and 2 (lower) testers, the other involving 2 SUTs and a single (lower) tester. Such multi-system test methods are outside the scope of this Recommendation.

Note – A similar requirement for a multi-system test method exists for MHS.

I.6.6 *Message handling protocols*

There are CCITT test suites for three protocols, the inter-personal message service (IPMS) (P2 protocol), the message transfer service (MTS) (P1 protocol), and reliable transfer service (RTS). There are also two basic system configurations that require consideration.

The first is the “end-system test configuration” which can be used to test P2, P1 and RTS. The second is the “relaying message transfer agent test configuration” which can be used to test the relaying aspects of the P1 protocol.

The P2 protocol test suite uses one external PCO at the boundary between the user agent layer and the MTS and one PCO at the upper boundary of the IUT. However, the P2 protocol does not include definitions of ASPs, so it has therefore been necessary to construct “hypothetical” ASPs in order to describe the abstract test suite in a formal way.

The P1 protocol test suite for end-system testing uses one external PCO at the boundary between the message transfer layer and the RTS and one PCO at the upper boundary of the IUT. However, testing of multiple destination delivery requires more than one user agent, which in turn requires more than one PCO at the upper boundary of the IUT. The P1 protocol test suite for relay testing uses two external PCOs at the boundary between the message transfer layer and the RTS.

The RTS test suite uses one external PCO at the boundary between the reliable transfer layer and the session layer and one PCO at the upper boundary of the user agent layer within the SUT. The RTS test suite also includes a third service access point in the description of the test cases; this is between the message transfer and the RTS within the SUT; it is used for clarification only and is not a PCO.

Therefore, the distributed single-layer test method is used for end-system testing of the P1 and P2 protocols; the distributed single-layer embedded test method is used for testing RTS; and the transverse test method is used for testing the relaying aspects of the P1 protocol. In all cases of end-system testing, the remote single-layer test method is also applicable.

A further point is that the abstract test suites for P2 and P1 protocols include X.409 encoding and decoding tests.

Note – The test suites referred to here are concerned with testing implementations of the 1984 versions of the CCITT X.400-Series Recommendations.

I.6.7 *Commitment concurrency and recovery (CCR)*

For reasons similar to those given in I.6.2 for association control, CCR should be tested as part of an application context, such as that required by JTM. Thus, the remote and distributed single-layer embedded test methods are applicable.

CCR contains a requirement to preserve secure data beyond crashes, although it recognizes that this is not possible to implement in practice because some error situations (e.g. bomb on the installation) are too catastrophic. In order to test this requirement to preserve secure data beyond crashes, it is necessary for the implementor to state in either the PICs or PIXIT which error situations would not affect the preservation of secure data.

I.6.8 *Presentation*

The service primitives are potentially observable and controllable to the same extent as for lower layers. Thus, all four single-layer (non-embedded) test methods are theoretically applicable. However, as discussed above in relation to association control, the testing of presentation protocol in isolation from an ASE is of limited value, because it could only test the protocol machine, leaving untested the more interesting aspect of the presentation layer, namely the mapping between abstract and transfer syntaxes. Therefore, the testing of presentation protocol embedded under association control and another ASE is preferred. Thus, the relevant applicable test methods are the remote and distributed single-layer embedded ones.

I.6.9 *Transfer syntaxes*

Transfer syntaxes (e.g. ASN-1 or X.409) are rather different from the OSI* protocol Recommendations* with respect to conformance. In general, there would not be conformance testing of the encoding rules of a transfer syntax independent of the application protocol using those rules.

It is noted that there are, for instance, conformance requirements in the ASN-1 basic encoding rules Recommendation. The requirement that where alternative encodings are provided as a sender's option, conforming receivers shall support all alternatives does require ASN-1 specific conformance testing. Nevertheless, the ASN-1 basic encoding rules will always be tested with the presentation protocol and the test methods appropriate to that protocol will be chosen.

I.7 *Management protocols*

Since system management and application management reside in the application layer, the general comments above which apply to application protocols, also apply to system and application management protocols. In particular, test cases for such management protocols can be specified in terms of management service primitives, provided that there is some mapping between the management service primitives and real effects which can be observed and/or controlled. If such service primitives and such a mapping exist, then the testing can be performed by the distributed test method. Otherwise, the testing can be performed by using the remote test method.

Also, as with other application protocols, the exchange of PDUs of a system management protocol can be tested by using the test methods described in this Recommendation. The application of conformance testing to other aspects of application and system management services and related "non-protocol" activities of management entities is, however, outside the scope of this Recommendation.

Thus, because of the "non-protocol" conformance requirements associated with system management protocols, in particular, the testing of conformance to them cannot fully be achieved by means of the methods described in this Recommendation. For example, the correct operation of the system management function of modifying the information in a directory cannot be tested merely by exchanging PDUs.

Furthermore, system management functions are not only related to the application layer but also to the operations of the underlying protocols. Therefore, the testing of the system management protocol needs to be carried out using an implementation of the underlying protocols which has already been tested.

The testing of layer management protocols is dependent upon the existence of the relevant protocols but, given that these exist, it should be possible to test them by means of the test methods described in this Recommendation.

I.8 *Connectionless protocols*

Since each test method described in this Recommendation is defined in terms of observation and control of ASPs and PDUs, and not in terms of connections, then all methods are applicable to the testing of connectionless protocols, taking into account the restrictions applying to each layer.

APPENDIX II

(to Recommendation X.290, Part 1)

Index of definitions of terms

<i>Term</i>	<i>Section</i>	<i>Term</i>	<i>Section</i>
Abstract local primitive	3.8.5	Parameterized executable test case	3.6.23
Abstract (N)-service primitive	3.8.4	Parameterized executable test suite	3.6.25
Abstract test case	3.6.3	Pass verdict	3.7.12
Abstract testing methodology	3.6.2	Passive testing	3.5.2
Abstract test method	3.6.1	PCO	3.8.1
Abstract test suite	3.6.16	PCTR	3.7.8
Active testing	3.5.1	PICS	3.4.6
ALP	3.8.5	PICS proforma	3.4.7
ASP	3.8.4	PIXIT	3.4.8
Basic interconnection testing	3.5.5	PIXIT proforma	3.4.9
Basic interconnection test suite	3.6.19	Point of control and observation	3.8.1
Behaviour testing	3.5.8	Postamble	3.6.9
Capabilities of IUT	3.4.5	Preamble	3.6.7
Capability testing	3.5.6	Protocol conformance test report	3.7.8
Client	3.4.12	Protocol implementation conformance statement	3.4.6
Comparability (of results)	3.7.2	Real tester	3.8.13
Conformance assessment process	3.5.10	Remote test method	3.8.12
Conformance log	3.7.15	Repeatability (of results)	3.7.1
Conformance testing	3.5.9	SCTR	3.7.7
Conformance test suite	3.6.18	Selected abstract test suite	3.6.20
Conforming implementation	3.4.10	Selected executable test suite	3.6.21
Coordinated test method	3.8.11	Static conformance requirements	3.4.4
Distributed test method	3.8.10	Static conformance review	3.5.7
Dynamic conformance requirements	3.4.3	SUT	3.4.2
Embedded testing	3.5.4	Syntactically invalid test event	3.7.10
Executable test case	3.6.4	System conformance statement	3.4.11
Executable test suite	3.6.17	System conformance test report	3.7.7
External test method	3.8.9	System under test	3.4.2
Fail verdict	3.7.13	Test body	3.6.8
Foreseen outcome	3.7.4	Test case	3.6.13
Generic test case	3.6.6	Test coordination procedures	3.8.6
Generic test suite	3.6.15	Test event	3.6.11
Implementation under test	3.4.1	Test group	3.6.14
Inconclusive verdict	3.7.14	Test laboratory	3.4.13
Inopportune test event	3.7.11	Test management protocol	3.8.7
IUT	3.4.1	Test purpose	3.6.5
Local test methods	3.8.8	Test realizer	3.8.14
Lower tester	3.8.2	Test step	3.6.10
Multi-layer testing	3.5.3	Test suite	3.6.12
(N)-ASP	3.8.4	Upper tester	3.8.3
Outcome	3.7.3	Unforeseen outcome	3.7.5
Parameterized abstract test case	3.6.22	Valid test event	3.7.9
Parameterized abstract test suite	3.6.24	Verdict	3.7.6

APPENDIX III

(of Recommendation X.290, Part 2)

Examples for guidance for PICS proforma specifiers

III.1 *Abbreviations*

The following examples use the abbreviations: “m” for mandatory “c” for conditional, “o” for optional, “n” for negotiable and “–” for not applicable, as defined in § 7.1.7.

III.2 *PDU example*

PDU NAME	CLAUSE REFERENCE	SUPPORT	
		TRANSMIT	RECEIVE
PDU-1	a.b.c	m	m
PDU-2	a.b.d	o	m
PDU-3	a.b.e	c	–

III.3 *Parameter example*

PARAMETER	CLAUSE REFERENCE	PERMITTED RANGE OF VALUES	VALUES SUPPORTED	SUPPORT STATUS
PARAM-1	x,y & p,q	unlimited		n
PARAM-2	f,g,h	1 → 10		o
PARAM-3	f,i	4		c
PARAM-4	w,x,y,z	type-A		m

A variant of this example would be a timer example which might include an extra column for units.

III.4 *Supported services example*

OPTION	Clause	SUPPORTED ON			
		2 Cans and Wet String	Carrier Pigeon	ExtraSensory Perception	UltraSound
Whisper	4	m	—	o	m
Echo	5	n	o	c	n
Whistle	6	o	—	c	m

APPENDIX IV

(to Recommendation X.290, Part 2)

Example of choice of abstract test methods

IV.1 *Rationale for the transport layer*

Objective : test a transport entity (single-layer IUT) in a real open system.

Assumption 1: the real open system is used for several applications.

Assumption 1.1: all applications use the OSI transport service via a locally defined and accessible interface.

Assumption 1.2: all applications use the OSI transport service via a locally defined and accessible session service interface, the transport service boundary being inaccessible.

Assumption 1.3: all applications use the OSI transport service via the OSI session service, but neither the transport service boundary nor the session service boundary are accessible.

Assumption 2: the real open system is used for one application only.

Assumption 2.1: all layer boundaries are accessible via locally defined interfaces.

Assumption 2.2: no layer boundary is accessible – the system does not provide interfaces except for the end-user (this is the case for Teletex and MHS monolithic products).

Assumption 2.3: no layer boundary is accessible – the system does not provide interfaces – even the end-user cannot access the boundary between the OSI and non-OSI of the application process.

The external test methods apply to these assumptions as follows:

- | | | | | | | |
|-----|-------------------------------|-----|----|-----|----|----|
| 1.1 | directly | CS | or | DS | | |
| 1.2 | embedded via session layer | CSE | or | DSE | | |
| 1.3 | | RS | | | | |
| 2.1 | directly | CS | or | DS | or | LS |
| 2.2 | embedded via all upper layers | CSE | or | DSE | | |
| 2.3 | | RS | | | | |

Conclusions: CS, CSE, DS, DSE, RS and LS test methods all apply to the transport layer. Embedded test methods can be used via the session layer on its own or via all the upper layers.

Priority: assumptions 1.1, 1.2 and 2.2 appear to be the most common cases (respectively, the direct access method, session access method, and Teletex and MHS monolithic products). Therefore, test methods CS, CSE and DSE should be given the highest priority.

IV.2 Main test methods to be used for the transport layer

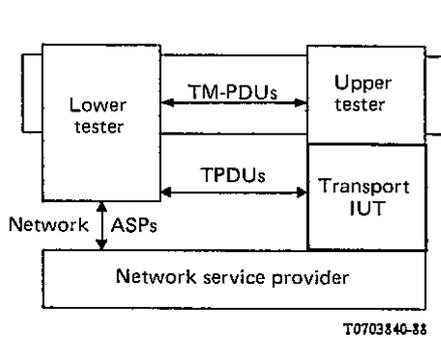


FIGURE IV-1/X.290, Part 2

The CS test method

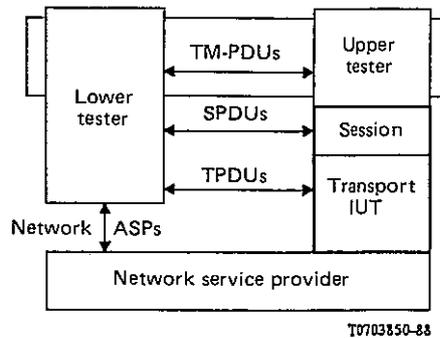


FIGURE IV-2/X.290, Part 2

The CSE test method

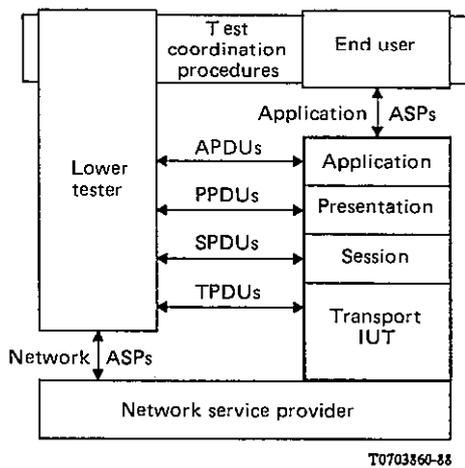


FIGURE IV-3/X.290, Part 2

The DSE test method

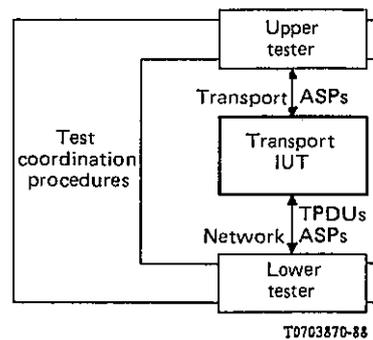


FIGURE IV-4/X.290, Part 2

The LS test method

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems