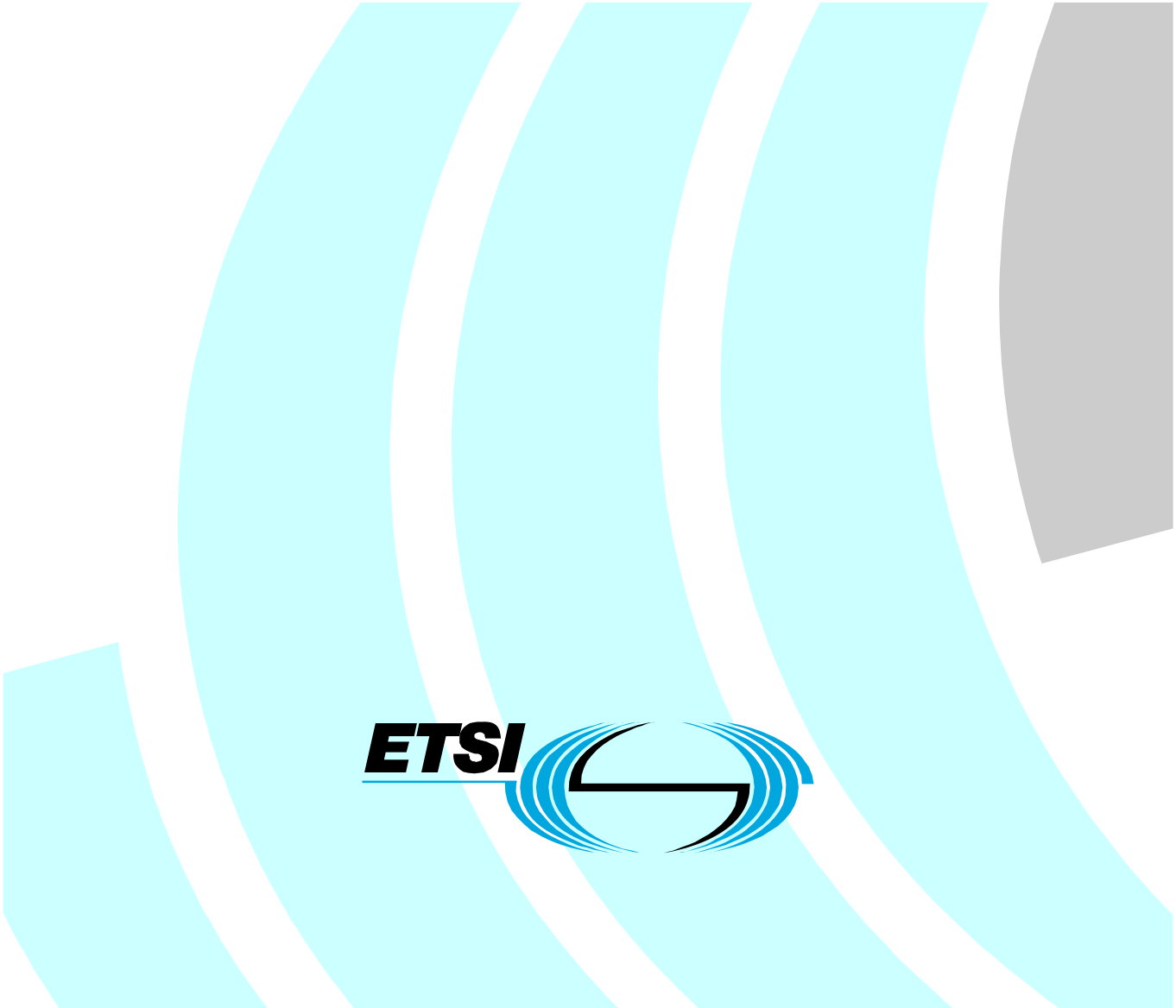
# ETSI ES 202 553 V1.2.1 (2009-06)

*ETSI Standard*

## Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes



Reference
RES/MTS-00100[2]-TPLan

Keywords
methodology, testing, TTCN

***ETSI***

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

***Important notice***

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

***Copyright Notification***

# Contents

# Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

# Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

# 1 Scope

The present document specifies the syntax and use of a notation for the definition of Test Purposes, TPLan. This notation provides a structure and a common set of English keywords for the specification of Test Purposes. The basic notation is oriented towards testing of reactive, black-box communication systems and uses terminology derived from ISO/IEC 9646-1 [3]. However, facilities are also included to allow users to extend the notation with application-specific keywords of their own.

The use of TPLan as the means of specifying Test Purposes is optional but, if it is used, the requirements specified in the present document shall be met.

# 2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.
- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:
  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;
  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

## 2.1 Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[2] ETSI EG 202 568 (V1.1.3): "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Testing: Methodology and Framework".

[3] ISO/IEC 9646-1: "Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 1: General concepts".

[4] ISO/IEC 9646-2: "Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 2: Abstract Test Suite specification".

## 2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Not applicable.

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

**event:** something observable (measurable) at a given place and time

NOTE: The cause of a stimulus or the result of a response.

**notation:** textual means of representing ideas

**programming language:** artificial language that can be used to control the behaviour of a machine

**test case:** specification of the actions required to achieve a specific test purpose, starting in a stable testing state, ending in a stable testing state and defined in either natural language for manual operation or in a machine-readable language (such as TTCN-3) for automatic execution

**test description:** systematic specification of the test steps (generally in tabulated text) that must be taken to reach a specific test verdict

**test purpose:** description of a well-defined objective of testing, focussing on a single interoperability requirement or a set of related interoperability requirements

**test suite structure:** logical grouping of test purposes or test cases which should be both relevant and convenient

## 3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

| | |
|---|---|
| EBNF | Extended Backus-Nauer Form |
| EUT | Equipment Under Test |
| ICS | Implementation Conformance Statement |
| IETF | International Engineering Task Force |
| IP | Internet Protocol |
| IPT | Internet Protocol Testing |
| IUT | Implementation Under Test |
| PICS | Protocol Implementation Conformance Statement |
| QE | Qualified Equipment |
| QSIG | Q interface Signalling protocol |
| RFC | Request For Comments |

NOTE: IETF terminology for a draft standard.

| | |
|---|---|
| RQ | ReQuirement |
| TC | Test Case |
| TD | Test Description |
| TP | Test Purpose |
| TPLan | Test Purpose Notation |
| TSS | Test Suite Structure |
| TTCN-3 | Testing and Test Control Notation edition 3 |

# 4 Introduction

## 4.1 TPLan: A formal notation for expressing test purposes

ISO/IEC 9646-1 [3] and ISO/IEC 9646-2 [4] recommend that test specifications include a concise and unambiguous description of each test which focuses on its purpose. These Test Purposes, or TPs, define **what** is to be tested rather than **how** the testing is performed. The TPs are based on the **requirements** identified in the relevant standard (or standards) from which the test specification is derived. The detailed coding of each Test Purpose is specified in a Test Case. Often Test Cases, or TCs, are written in a test specification language such as TTCN-3 [1]. The specification of Test Cases is outside the scope of the present document.

Generally, Test Purposes are written in prose (possibly displayed in a tabular format). There is considerable benefit to be gained by having all TPs written in a similar and consistent way. With this in mind, a simple, structured notation called TPLan has been developed for the expression of TPs. TPLan is defined with a minimal set of test-oriented keywords but with the capability that permits users to define extensions to the notation. is an example of how TPLan can be extended into a specific application area; in this case, telecommunications.

The benefits of using TPLan are:

- consistency in test purpose descriptions - less room for misinterpretation;
- clear identification of the TP pre-conditions, test body, and verdict criteria;
- automatic syntax checking and syntax highlighting in text editors;
- a basis for a TP transfer format and representation in tools.

## 4.2 Extensibility of TPLan

TPLan provides a framework for a consistent representation (format, layout, structure and logical ordering) and a consistent use of words and patterns of words for expressing TPs. This is achieved without unnecessarily restricting the expressive power of pure prose.

TPLan allows the use of keywords in combination with free-text strings (enclosed by single quotes). Thus, the TP writer has considerable freedom of expression in the use of unstructured text between the keywords.

The basic set of pre-defined TPLan keywords has been kept to a minimum. These keywords are mainly concerned with providing structure to the TPs. The intention is that this set of keywords is extended by the user for specific testing applications through the use of user-defined keywords (see clause 8) which can be checked by automatic tools for consistency and, to some extent, correctness.

## 4.3 The Test Suite Structure

Test Purposes should be grouped in a tree-like structure. This structure is known as the Test Suite Structure, or TSS. The combination of structure and Test Purposes is known as the TSS&TP [3].

The general composition of a TPLan TSS&TP is as follows:

```
TSS Header            -- title, author, version etc.
Cross References  -- references to base standards, configuration descriptions etc.
Definitions           -- user-defined words, events, test entities, conditions, headers etc.
TSS Groups            -- if any and possibly nested
Test Purposes         -- contained in the groups (if any)
```

## 4.4 Areas of application

TPLan is not specific to a particular type or area of testing. The fundamental set of predefined TPLan keywords is oriented towards conformance and interoperability testing (keywords such as **`IUT`**, **`TESTER`** and **`TD`**) but the extensibility of the language means that the user can adapt TPLan to a wide range of testing contexts.

## 4.5 Limitations of TPLan

The TPLan grammar provides limited syntax checking and an enhanced visual representation of the TP in, for example, a syntax sensitive text editor. However, in order to retain expressive power, TPLan is only loosely defined in that no strict relation between certain words (especially the user-defined words) is specified. Thus, it is possible to write nonsensical constructions if care is not taken. Of course, appropriate tools may be able to identify such constructions but there are no constructs for doing this explicitly in the notation.

# 5 TPLan keywords, comments and identifiers

## 5.1 TPLan keywords

Only those words listed in table 1 shall be considered to be valid TPLan keywords.

**Table 1: TPLan keywords**

| **TSS header keywords** | |
|---|---|
| author | Author |
| date | Date |
| title | Title |
| TSS | tss |
| version | Version |
| **Cross-references keywords** | |
| xref | Xref |
| **Definitions keywords** | |
| condition | Condition |
| context | Context |
| def | Def |
| entity | Entity |
| event | Event |
| header | Header |
| value | Value |
| unit | Unit |
| word | Word |
| **TP grouping keywords** | |
| end | End |
| group | Group |
| objective | Objective |
| **TP header keywords** | |
| config | Config |
| id | Id |
| ref | Ref |
| role | Role |
| RQ | rq |
| summary | Summary |
| TC | tc |
| TD | td |
| TP | tp |
| **TP body keywords** | |
| ensure | Ensure |
| that | That |
| with | With |
| when | When |
| then | Then |
| **Test entity keywords** | |
| IUT | iut |
| TESTER | tester |

| TPLan glue words | |
|---|---|
| a | A |
| an | An |
| as | As |
| in | In |
| is | Is |
| no | No |
| of | Of |
| the | The |
| **Logical words** | |
| and | And |
| not | Not |
| or | Or |
| **Stimulus and Response words** | |
| receives | Receives |
| sends | Sends |
| **Data-related words** | |
| containing | Containing |
| indicating | Indicating |
| **Direction-related words** | |
| from | From |
| to | To |
| **Time- and order-related words** | |
| after | After |
| before | Before |
| unordered | Unordered |
| within | Within |

## 5.2 Comments

Comments shall be introduced by the string "--" and terminated at the end of the same line.

## 5.3 TPLan identifiers

Letters, numbers and special characters may be used in a TPLan identifier, as follows:

- alphabetic:
  - a to z;
  - A to Z.
- Numeric:
  - 0 to 9.
- special characters:
  - |._-&%$*@?!></\#.

Typical TPlan identifiers are TP identifier, event names, cross reference identifiers and requirements identifiers. For example:

```
MyTSS&TP
TP_UMTS_0789_01
RQ_001_789
REQ3952.Arev2
CONF/HOST/INVALID/#75
CF_MOB_02
PICS_c.2
```

A TPLan identifier shall not contain any white space (e.g. tabs or spaces).

NOTE: In certain contexts it can be desirable to overlay TPLan with an additional level of checking related to a particular methodology or naming convention. Generally, such overlays are outside the scope of the present document. However, the ETSI IPT Testing Framework [2] includes naming rules for TPs and other identifiers. These conventions are summarized in annex B.

## 5.4 Uniqueness of identifiers

All user-defined words, headers, entities, conditions and events shall be unique in the scope of one TSS&TP. Parameter names of events shall be unique in the scope of the list in which they are declared.

No user-defined name shall be the same as any pre-defined TPLan keyword.

## 5.5 Including files

TPLan supports a limited form of file inclusion. This shall be indicated by the pre-processing directive, **`#include,`** followed by the location of the included file enclosed in quotes. TPLan does not define the form of this location reference. Typical locations may be a file path, reference to an ETSI standard, an object identifier or a URL. For example:

```
#include 'root/MyTPLan/MyDefs.txt'
#include 'root/MyTPLan/MyTPs.txt'
```

The include statement shall only be used to include definitions (see clause 8) or complete test purposes comprising the test purpose header and body (see clauses 10 and 11). Thus, the included file shall follow the syntax for definitions and test purposes as defined in annex A.

# 6 TSS Header

## 6.1 Standard TSS header entries

A TPLan specification (i.e. a TSS&TP) shall begin with the following headers (though not all entries are mandatory, see annex A):

- TSS identifier (mandatory):
  - The keyword **`TSS`** followed by the formal identifier of the TSS&TP.
- Title (optional):
  - The keyword **`title`** followed by the name of the TSS&TP as quoted free text.
- Version (optional):
  - The keyword **`version`** followed by the version number as any number of numeric values separated by dots (".").
- Date (optional):
  - The keyword **`date`** followed by three numeric values separated by dots (".), forward slash ("/") or dash ("-"). The following examples are all valid date entries:
    - 01-12-2007.
    - 1.12.07.
    - 31/5/2007.
    - 05-31-07.
- Author (optional):
  - The keyword **`author`** followed by the document author(s) as quoted free text.

Each TSS header keyword shall be followed by a colon (":").

A complete TSS&TP header:

```
TSS     : UMTS_TSS
title   : 'My TSS&TP as an example'
version : 1.0         -- other examples may be 1.0.0 or 11.01
date    : 29.11.2004  -- could also be written as 29/11/2004 or 29-11-2004
author  : 'ETSI PTCC'
```

## 6.2 User-defined TSS Header entries

Additional user-defined TSS&TP headers may be included with the standard headers. A user-defined TSS header may comprise one or more defined header words, followed by a list of one or more identifiers separated by commas, or a quoted string separated by a colon (see clause 8.2). For example:

```
TSS     : UMTS_TSS
title   : 'My TSS&TP as an example'
version : 1.0
date    : 29.11.2004
author  : 'ETSI PTCC'
status  : 'Public'      -- user-defined TSS header
```

A user-defined header may also use the predefined keyword **ref**. For example:

```
PICS ref  : 'Release 7'
```

A header (either predefined or user-defined) shall not appear more than once in a TSS.

# 7 External references

References to external sources of information shall be made using the **xref** keyword followed by an identifier and a list of the external sources. Examples of the use of the **xref** keyword are as follows:

- References to base standards or other sources from which the TPs have been derived. For example:

```
xref BaseStandards {TS123456-1, TS789345}
```

In the case where a PICS [3] is used the reference might be:

```
xref PICS {TS123456-1, TS123456-2}
```

The TPLan syntax allows reference to be made in any appropriate form such as Object identifiers or URLs. For example:

```
xref MyWebDocs {www.tplan.info}
```

- References to explicit test configurations. For example:

```
xref TestConfig1 {3GPPSpecXYZ_AnnexA_page10_fig1}
```

- The list of configuration sources may include references to file names as well as or instead of document information, thus:

```
xref TestConfig2 {OurProjectConfigs.pdf}
```

NOTE: The identifiers associated with **xref** statements could be used by software tools to check for consistency within the specifications of test purposes. For example, a tool could check that TPs refer only to configurations that have been previously declared in **xref** statements.

# 8 User definitions

## 8.1 User-defined words

TPLan may be extended by the user with the **def word** keywords followed by one or more identifiers separated by commas:

```
def word tunnels, forwards
```

For information to the reader, each word definition may be followed by a description of that word as a quoted string. For example:

```
def word forwards   ' receive a packet on one interface and transmit
                    that packet unchanged on another interface'
```

These words may be used in the **with**, **when** and **then** statements of the TP body (see clause 11).

Although the addition of new keywords can make a TPLan specification considerably easier to understand, care should be taken to avoid adding multiple words with almost identical meaning. Also, the **def context** construct (see clause 8.8) should be used wherever possible to limit the use of newly-defined keywords.

## 8.2 User-defined headers

The TSS and TP headers may be extended using the **def header** keywords followed by one or more identifiers separated by commas.

```
def header status
```

For information to the reader, each header definition may be followed by a description of the meaning of that header as a quoted string.

## 8.3 User-defined test entities

Explicit test entity names may be defined using the **def entity** keywords followed by one or more identifiers separated by commas.

```
def entity EUT, QE
```

For information to the reader, each entity definition may be followed by a description of the meaning of that entity as a quoted string.

These test entity names may be used in the **with**, **when** and **then** statements of the TP body (see clause 11).

## 8.4 User-defined events and parameters

Explicit event names may be defined using the **def event** keywords followed by one or more identifiers separated by commas. There is no strong definition of what an event is in TPLan, but, typically, these would be messages, timers or something else that is observable or, at least, measurable. In the present document the term "message" is generally used as an example of an event and, for clarity, event parameters are referred to as message fields, or fields for short. For example:

```
def event ICMP_Packet, SETUP
```

Optionally, the event name may be followed by a list of one or more parameter identifiers or user-defined values separated by commas. For example:

```
def event SETUP {f1, f2}
-- or
def event T1 {30sec}
```

For information to the reader, each event definition may be followed by a description of the meaning of that event as a quoted string.

These event and parameter names may be used in the **when** and **then** statements of the TP body (see clause 11).

NOTE: These are abstract definitions of events and are not intended to be, for example, records or similar constructs found in common programming languages.

## 8.5 User-defined values

Values may be defined using the **def value** keywords followed by one or more identifiers or literal values separated by commas.

```
def value T1, 3sec, FFFF, A76.4FF.321.255, ERRCODE, 200OK
```

For information to the reader, each value definition may be followed by a description of the meaning of that value as a quoted string. For example, it can be useful to define an error code as a literal value and identify its actual value in the quoted string, thus:

```
def value AUTHENTICATION_FAILED  '24'
```

These values may be used in the **with**, **when** and **then** statements of the TP body (see clause 11).

Integer values (e.g. 1, -653, 001, 0), real values of the form **number dot number** (e.g. 37,12, -1,5, 0,002) and exponentiated values of the form **real value e integer value** (e.g. 3,2e5, 0,65e3, 2,0e-3) are built-in and need not be explicitly defined.

Optionally, a value name may be followed by a list of one or more additional value names separated by commas. For example:

```
def value package1 {a, b, c}
-- or
def value area {length, width}
```

NOTE: The optional additional value names are used only to identify elements that should be included in the substructure of the defined value. They should not be interpreted as a specification of array parameters or possible enumerated values. As an example, in the definition of `area` above:

- `area` **can** be considered to comprise the values `length` and `width`;
- `area` **cannot** be considered to be:
  - a 2-dimensional array of size `length` × `width`;
  - an enumerated type which can only take the values `length` and `width`.

## 8.6 User-defined units

Units may be defined using the **def unit** keywords followed by one or more identifiers or literal values separated by commas.

```
def unit seconds, metres, volts, newtons, henrys
```

For information to the reader, each unit definition may be followed by a description of the units as a quoted string.

Care should be taken to ensure that, if required, both the singular and the plural version of each unit is defined. For example, using only the definition above, the expression `1 second` would be considered to be syntactically incorrect. If singular units are required within a TPLan specification, it is necessary that they be defined explicitly. The above example would then be extended as follows:

```
def unit second, seconds, metre, metres, volt, volts, newton, newtons, henry, henrys
```

Alternatively, units can be defined using their generally accepted abbreviations which are normally considered to be both singular and plural. To make such definitions clearer to the reader, the optional descriptive string should be used to provide the expanded unit description, thus:

```
def unit s        'seconds'
def unit m        'metres'
def unit V        'volts'
def unit N        'newtons'
def unit H        'henrys'
```

However, the TPLan syntax does not permit the use of symbolic unit abbreviations such as "ohm" (Ω) and "degree" (º) or prefixes such as "micro" (μ) or indices such as "squared" ($^2$). These abbreviations should be defined using plain textual constructs, such as:

```
def unit ohm
def unit degC     'degrees Celsius'
def unit micro_s  'micro-seconds'
def unit m2       'square metres'
```

## 8.7 User-defined conditions

Conditions or states may be defined using the **`def condition`** keywords followed by one or more identifiers or literal values separated by commas:

```
def condition Idle, Ready, SESSION_ESTABLISHED
```

For information to the reader each condition definition may be followed by a description of the meaning of that condition as a quoted string.

These conditions may be used in the **`with`**, **`when`** and **`then`** statements of the TP body (see clause 11).

## 8.8 Constraining keywords to specific contexts

If required, the use of certain keywords may be constrained to appear only in combination with other keywords. For example, a user may wish to define the word **`requested`** and restrict its use so that it can appear only in the context of **`is requested to`**.

Such restrictions shall be expressed by using the **`def context`** keywords followed by one or more predefined or user-defined keywords.

```
def context {is ~requested to}
```

The tilde(~) character shall be used to indicate that a particular user-defined word shall only be used in that context (constrained). Words not preceded by tilde may appear in any context (unconstrained). In a single TPLan specification, a TPLan word shall not be constrained by one **`def context`** statement and unconstrained by another **`def context`** statement. A tilde character shall not be considered to constrain the use of any predefined keyword immediately following it.

Keywords that are optional within the defined context shall be expressed by enclosing them in square brackets, for example:

```
def context {is [not] ~requested to}
-- which means that the word requested can only be used in the following contexts:
-- is requested to
-- is not requested to
```

NOTE: Although the **`def context`** construct is of benefit to human readers, it can also be used by tools to automatically include additional checking.

A **def context** construct may include predefined TPLan keywords and user-defined notation extensions (e.g. words and entities) but should only be used to define the syntactical context associated with particular user-defined words. It should not be used to construct entity, event, condition or value identifiers which contain white space; i.e. in the following example, the definition of the away_from_home condition is semantically different from the result of the context definition which permits the use of the construct, **away from home**:

```
def condition away_from_home

def context {~away from ~home }
```

The inclusion of a user-defined TPLan word in more than one **def context** construct shall be interpreted as the specification of alternative constraints on the use of a word rather than combined constraints. The following example shows how two different contexts could be specified for the word "established":

```
def word established
def word having

def context {having ~established}
def context {~established as}
```

With the above context definitions, it is possible to use the word "established" either immediately after the word "having" or immediately before the word "as", for example:

```
IUT having established a connection
IUT established as default_router).
```

In this example, the word "established" cannot be used in any other immediate arrangement of predefined or user-defined words.

# 9 Groups

The TSS (Test Suite Structure) shall be expressed using the **group** keyword. Groups may be nested to provide sub-grouping. The contents of a group may be other groups (sub-groups) or TPs or both sub-groups and TPs. A TSS&TP does not have to be structured but, if it is, each group in that structure shall have the following group header:

- Begin group:
  - The keyword **group** denotes the start of a new group. This keyword shall be followed by at least one of the following:
    - the group number;
    - a string of free text.
  - The group number is any number of digits separated appropriately by dots ('.') e.g. 1 or 1.9 or 1.12.3.
- Group objective:
  - The keyword **objective** followed by a quoted free text description of the objective of the test group. This entry is optional.
- End group:
  - The keywords **end group** denote the end of a group. These keywords shall be followed by at least one of the following:
    - the group number which should be the same as the group number used at the start of the group;
    - a string of free text.

An example of one group and a sub group:

```
group 1 'Router(RT)' -- group number with optional free text
objective 'Test Purposes for Router'
  group 1.1 ' Router(RT)/Provide IPv6 Services(PS)' -- a sub-group
  objective 'Test Purposes for IPv6 Services'
             -- TPs and/or more subgroups can go here
  end group 1.1 'RT/PS'
             -- TPs and/or more subgroups can go here
end group 1 'RT'
```

# 10 TP Header

## 10.1 Standard TP Header entries

Each TP shall begin with a header construct as follows:

- TP id (mandatory):
  - The keywords **`TP id`** followed by the TP Identifier.
- TP summary (optional):
  - The keyword **`summary`** followed by a free text high-level description (overview) of the TP in quotes.
- Requirements reference (optional):
  - The keywords **`RQ ref`** followed by the reference identifier.
- Role (optional):
  - The keyword **`role`** followed by a list of one or more identifiers indication the role or roles of the object being tested by the TP (e.g. router or host).
- Configuration reference (optional):
  - The keyword **`config`** followed by a reference to the relevant testing configuration.
- The Test Case or Test Description reference (optional):
  - The keywords **`TC ref`** or **`TD ref`** followed by a reference to the corresponding Test Case or Test Description.

Header keywords may be followed by a colon (":").

For example:

```
TP id  : TP_COR_0001
summary  : 'Test for determining the correct use of the Pad1 option'
RQ ref   : RQ_COR_0001
role  : host
config  : CF_001_C
TC Ref  : TC_COR_0001
```

## 10.2 User-defined TP Header entries

Additional user-defined TP headers may follow the standard headers (see clause 8.2). A user-defined TP header may comprise one or more defined header words and the predefined keyword **ref** followed by a list of one or more identifiers separated by commas, or a quoted string (optionally separated by a colon). For example:

```
TP id  : TP_COR_0001
summary  : 'Test for determining the correct use of the Pad1 option'
RQ ref   : RQ_COR_0001
role  : host
config  : CF_001_C
TC ref  : TC_COR_0001
select  : 'Profile A'
PICS ref : PICS_001, PICS_345          -- where PICS has previously been defined as a header and
                                          the identifiers as xrefs
web ref : 'www.tplan.org/profileA.htm' -- where 'web' has previously been defined as a header
```

# 11 TP body

## 11.1 TP body structure

The body of the TP follows the header and it is here that the Test Purpose is described in detail. The TP is generally written from the viewpoint of the Implementation Under Test (IUT).

The general structure of a TP is:

```
Pre-conditions            -- optional initial conditions
TP behaviour description  -- comprising sequences of:
  Stimuli and Responses
```

Each TP behaviour description shall begin with the keywords **ensure that** followed by the remainder of the description enclosed in curly braces ('{' and '}').

For example:

```
ensure that {

  -- TP behaviour description goes here
}
```

The **when** and **then** statements describe stimuli and responses (interactions) as seen from the point of view of the IUT. Generally these are of the form:

```
ensure that {
              when { ... }   -- stimuli described from the viewpoint of the IUT.
              then { ... }   -- IUT responses and other behaviour
}
```

This pair of statements may be repeated any number of times to define a sequence of stimulus/response pairs, for example:

```
ensure that {
  when { ... }
  then { ... }
  when { ... }
  then { ... }
            }
```

## 11.2 TP pre-conditions

The **with** statement may be used to express the initial state or condition of the **IUT** from which the TP description begins. If used, the **with** statement shall precede the **ensure that** statement. The **with** statement does not define the steps or actions needed to reach the starting condition, only the condition itself. The conditions shall be expressed as free text. Multiple conditions shall be logically concatenated using the Boolean operators **and**, **or**, **not**. The general format of the **with** statement is:

```
with { IUT 'condition 1' and 'condition 2' and not ...etc...}
```

For example:

```
with { IUT 'in idle state' and 'port80 open' }
ensure that {
  when { ... }
  then { ... }
       }
```

Conditions may be defined as described in clause 8.7. In which case the condition above might be:

```
with { IUT in Idle_state and 'port80 open' }
```

## 11.3 TP stimuli

The **when** statement shall express some form of stimulus. In most cases such stimuli are caused by the tester and experienced by the IUT. Typically this will be a **receives** statement (i.e. the IUT has received a stimulus) with the name or description of the received event.

```
IUT receives 'a message'
```

In cases where there is more than one possible source of event (e.g. an incoming message) in the test configuration the keyword **from** may be added to the **receives** in order to identify that source.

```
IUT receives 'a message' from 'some interface'
```

A receive statement may include the keyword **containing** which shall be followed by either a valid event parameter name or a free text quoted string. This may, itself, be followed by the keyword **indicating** which shall be followed by a numeric value, a defined value name or a free text quoted string.

```
IUT receives 'a message' containing 'description of a field'
                         indicating 'expected value of a field'
```

Using defined message names rather than strings allows for consistency checking of message names throughout the TSS&TP. For example:

```
def event AMessage {f1}
. . .
. . .
when { IUT receives AMessage containing f1
                            indicating 'expected value of a field' }
```

Further consistency checking can be achieved by defining the source of an event as an entity, thus:

```
def entity Router1
. . .
. . .
when { IUT receives MyMessage from Router1 }
```

The keywords **and, or** and **not** may be used to concatenate and qualify actions and conditions within the **when** statement. For example:

```
when {   IUT receives 'a message' from 'node 1'
           containing 'field 1' indicating 'any valid value'
       and containing 'field 2' indicating 'a know source node'
       and not containing 'field 3' indicating 'an error in transmission'
}
```

## 11.4 TP responses

The **then** statement shall express the expected response to the previous **when** statement. In most cases the response is performed by the IUT and observed by the tester. Typically this will be a **sends** statement followed by the name or description (expressed as free text) of the sent message.

```
IUT sends 'a message'
```

In cases where there is more than one possible message destination in the test configuration the keyword **to** may be added to the **sends** in order to identify that destination. For example:

```
IUT sends 'a message' to 'some interface'
```

The syntax of the contents of sent messages is the same as that for the received messages. For example:

```
IUT sends 'a message' containing 'description of a field' indicating 'expected value of a field'
```

The keywords **and, or** and **not** may be used to concatenate and qualify responses and conditions within the **then** statement. For example:

```
then {      IUT sends 'another message' to 'node 1'
          containing 'field 3' indicating 'any valid value'
       and containing 'field 4' indicating 'any valid value'
       and IUT sends 'yet another message'
}
```

As with the **receives** statement, user-defined messages and field names may be used in place of quoted strings.

## 11.5 Precedence of TPLan statements

In cases where successive logical operations are used in the TP body, it may not be clear what the intended order of evaluation may be. Parentheses shall be used to resolve such ambiguities. For example:

```
with { IUT ('condition 1' and 'condition 2') or 'condition 3'} -- is not the same as:
with { IUT 'condition 1' and ('condition 2' or 'condition 3')}
```

The assumed order of precedence is governed by the following rules:

- the basic order of precedence is from left to right;
- logical words within brackets before those that are not;
- where nested brackets are used, logical words within the innermost brackets are evaluated before those in outer brackets.

## 11.6 Temporal ordering of TPLan statements

If the strict sequence of TPLan behavioural statements is important, this shall be expressed using the pre-defined words **before** and **after**. For example:

```
when {IUT receives message1
      before IUT receives message2 }
then { ....}
```

Statements may also be enclosed by parentheses to make the intended sub-ordering clear. For example:

```
when {IUT receives message1
     (before IUT receives message2
             before IUT receives message3)}
then { ....}
```

By default, sequential TPLan statements shall be evaluated in the order that they appear. For example, the following TPLan:

```
when {     IUT receives A
       and IUT receives B }
```

means that the IUT shall respond to the receipt of message A followed by message B.

The keyword **unordered** shall be used if it is required to express that a set of events may not evaluate in a sequential manner.

```
when { unordered (IUT receives A
                  and IUT receives B)}
```

means that the IUT shall respond to the receipt of either message A followed by message B or message B followed by message A. Again, parentheses may be used to clearly show what the scope of the ordering should be.

The keywords **before**, **within** and **after** may also be used to express ordering, especially in the context of timers. For example:

```
before 'timer T1 expires'
-- or
within 'two minutes'
-- or
after '15 seconds'
```

In the following example, note also the use of the defined value "15s" in place of the string "15 seconds"):

```
then {   IUT sends 'a message' to 'Node 1' within 15s
}
```

## 11.7 Using user-defined test entities, conditions and words

In some cases the pre-defined TPLan keywords may not be adequate. In such cases users may define additional keywords suited to particular needs (see clause 8.3). For example, it would be beneficial to define the **entities EUT** (Equipment Under test) and **QE** (Qualified Equipment) and the **word forwards** in order to make an interoperability TP clearer, thus:

```
then {     EUT accepts 'an incoming IPv6 Packet'
       and EUT forwards 'the packet' to 'Node 1' within 15s
}
```

A timer may be defined as a test entity (see clause 8.3), for example:

```
when { T1 expires ... } -- this example assumes the defined word 'expires'
```

Defined conditions (see clause 8.7) may be used to express states instead of quoted strings, for example:

```
IUT changes from IDLE to ACTIVE -- assumes the defined word 'changes' as
                                -- well as the definition of the ACTIVE
                                -- and IDLE conditions
```

## 11.8 Glue words and readability

To aid readability, TPLan allows the use of "glue" words such as **a**, **an** and **the**. For example:

```
then {     the EUT accepts an 'incoming IPv6 Packet'
       and the EUT forwards a 'message' to 'Node 1' within '15 seconds'
}
```

Syntax highlighting (i.e. use of multiple colours) can also aid readability:

```
then {     the EUT accepts an IPv6_Packet
       and the EUT forwards an ICMP_Packet to RouterA within '15 seconds'
}
```

# Annex A (normative): The TPLan Grammar

## A.1 Syntactic Rules

This annex defines the TPLan grammar in Extended Backus-Nauer Form (EBNF). This can be used either as a reference or as input to parser generator tools. Table A.1 defines the syntactic conventions that should be used when reading the TPLan EBNF.

**Table A.1: The syntactic metanotation**

| | |
|---|---|
| ::= | is defined to be |
| abc | the non-terminal symbol abc |
| abc xyz | abc followed by xyz |
| abc \| xyz | alternative (abc or xyz) |
| [abc] | 0 or 1 instances of abc |
| {abc} | 1 or more instances of abc |
| [{abc}] | 0 or more instances of abc |
| a - z | all characters from a to z |
| '...' | denotes a regular expression |
| (...) | denotes a textual grouping |
| "abc" | the terminal symbol abc |
| ; | production terminator |
| \ | the escape character |

## A.2 TPLan EBNF Productions

```
// BNF grammar for TSS & TP language (TPLan)
// Version: 2.6
// Date: 08.10.2007
// Author: ETSI CTI


// TSS header
tss_header          ::= KWD_tss DELIM ext_tss_id
                    [tss_title]
                    [tss_version]
                    [tss_date]
                    [tss_author]
                    [{user_tss_header}]
                    tss_body;

tss_title           ::= KWD_title DELIM
                    qstring;

tss_version         ::= KWD_version DELIM
                    numeric [{DOT numeric}];

tss_date            ::= KWD_date DELIM
                      '[0-9][0-9]' DOT     '[0-9][0-9]' DOT     '[0-9][0-9][0-9][0-9]'
                    | '[0-9][0-9]' F_SLASH '[0-9][0-9]' F_SLASH '[0-9][0-9][0-9][0-9]'
                    | '[0-9][0-9]' DASH    '[0-9][0-9]' DASH    '[0-9][0-9][0-9][0-9]';

tss_author          ::= KWD_author DELIM
                    qstring;

user_tss_header     ::= {header_id | TSS_header_KWDS} DELIM
                    (user_header_list | qstring);

user_header_list    ::=   extended_id [{SEPARATOR extended_id}];
```

```
// TSS body
tss_body            ::= [{xrefs}]
                    [{definitions}]
                    {group | tp};

// References and definitions
xrefs               ::= KWD_xref
                    xref_id
                    L_BRACE extended_id [{SEPARATOR extended_id}]R_BRACE;

definitions         ::= KWD_def
                    (define_word
                    | define_header
                    | define_event
                    | define_entity
                    | define_unit
                    | define_value
                    | define_condition
                    | define_context) [qstring]
                    | includes;

includes            ::= "#include" qstring;

define_word         ::= KWD_word
                    word_id [{SEPARATOR word_id}];

define_header       ::= KWD_header
                    header_id [{SEPARATOR header_id}];

define_event        ::= KWD_event
                    event_id [field_list] [{SEPARATOR event_id [field_list]}];


field_list          ::= L_BRACE field_id | value_id [{SEPARATOR field_id | value_id}] R_BRACE;
// STATIC SEMANTICS 1: field_id shall be unique in the field list

define_entity       ::= KWD_entity entity_id [{SEPARATOR entity_id}];

define_unit         ::= KWD_unit unit_id [{SEPARATOR unit_id}];

define_value        ::= KWD_value value_id [field_list] [{SEPARATOR value_id [field_list]}];

define_condition    ::= KWD_condition condition_id [{SEPARATOR condition_id}];

define_context      ::= KWD_context L_BRACE {context} R_BRACE;

context             ::= [L_BRACKET] context_id [R_BRACKET];

// STATIC SEMANTICS 2: If used, each and every L_BRACKET shall be paired with a corresponding
R_BRACKET

// Grouping
group               ::= group_header
                    [group_objective]
                    [{group | tp}]
                    [group_num][qstring]

group_header        ::= KWD_group
                    [group_num]
                    [qstring];

group_objective     ::= KWD_objective DELIM
                    [qstring];

tp                  ::= (tp_header
                    tp_body)
                    | includes;

// TP Header
tp_header           ::= tp_identifier
                    [summary]
                    [req_ref]
                    [role]
                    [config_ref]
                    [tc_or_td_ref]
                    [{user_tp_header}];

tp_identifier       ::= KWD_tp KWD_id DELIM
```

```
                    TP_id;

summary             ::= KWD_tp_summary DELIM
                    [qstring];

req_ref             ::= KWD_req KWD_ref DELIM
                    [cat_ref_list];

role                ::= KWD_role DELIM
                    [role_ref_list];

config_ref          ::= KWD_config DELIM
                    [CF_id];

tc_or_td_ref        ::= tc_ref | td_ref;

tc_ref              ::= KWD_tc KWD_ref DELIM
                    TC_id;

td_ref              ::= KWD_td KWD_ref DELIM
                    TD_id;

cat_ref_list        ::= RQ_id [{SEPARATOR RQ_id}];

role_ref_list       ::= role_id [{SEPARATOR role_id}];

user_tp_header      ::= {TPLan_Hid | TP_header_KWDS} DELIM
                    (user_header_list | qstring);

// TP body
tp_body             ::= [preconditions]
                    KWD_ensure KWD_that
                    begin_tp
                        {[stimuli] responses}
                    end_tp;

preconditions       ::= KWD_precondition
                    begin_conditions
                        [ precondition [{KWD_logical precondition}]]
                    end_conditions;


precondition        ::= [test_object] mixed_text;

stimuli             ::= KWD_stimulus
                    begin_stimuli
                        [stimulus [{KWD_logical stimulus}]]
                    end_stimuli;

stimulus            ::= [test_object] mixed_text;

responses           ::= KWD_response
                    begin_responses
                        [response [{KWD_logical response}]]
                    end_responses;

response            ::= [test_object] mixed_text;

TPLan_word          ::= predefined_words
                    | num_id
                    | TPLan_id;

test_object         ::= TPLan_Eid | KWD_IUT | KWD_TESTER;

mixed_text          ::= TPLAN_word
                    | qstring
                    | (mixed_text mixed_text)
                    |(L_PAREN mixed_text R_PAREN);


// TPLan identifiers
// STATIC SEMANTICS 2: no identifier of any kind shall be the same as any
// other predefined or user-defined TPLan keyword or identifier

// TSS and TP related identifiers
ext_tss_id          ::= extended_id;
ext_xref_id         ::= extended_id;
ext_RQ_id           ::= extended_id;
```

```
ext_CF_id           ::= extended_id;
ext_TC_id           ::= extended_id;
ext_TD_id           ::= extended_id;
ext_TP_id           ::= extended_id;
role_id             ::= extended_id;
// Header identifiers
header_id           ::= extended_id;
TPLan_Hid           ::= extended_id;
// Test entity identifiers
entity_id           ::= extended_id;
TPLan_Eid           ::= extended_id;
// Event (Message) and field identifiers)
field_id            ::= extended_id;
event_id            ::= extended_id;
// Unit identifiers
value_id            ::= extended_id;
// Value identifiers
unit_id             ::= extended_id;
// Word identifiers
word_id             ::= extended_id;
TPLan_id            ::= extended_id;
// Condition identifiers
condition_id        ::= extended_id;

extended_id         ::= '[a-zA-Z0-9|._&%$*@%?></\#!-]+';
context_id          ::= ["~"]extended_id

// Numbering
group_num           ::= numeric [{DOT numeric }];

numeric             ::= '[0-9]+';

num_id              ::= '[0-9.eE]+';


// STATIC SEMANTICS 3: Table 1 of this present document shows alternative forms of
// case sensitivity for the TPLan keywords.
// For simplicity the keywords shown in this BNF correspond to the
// left-hand column of Table 1. The alternatives in column 2 are assumed.

// TSS header keywords

TSS_header_KWDS     ::= KWD_author | KWD_date | KWD_title | KWD_tss | KWD_version;
KWD_author          ::= "author";
KWD_date            ::= "date";
KWD_title           ::= "title";
KWD_tss             ::= "TSS";
KWD_version         ::= "version";

// Reference and definition keywords
KWD_xref            ::= "xref";

KWD_condition       ::= "condition";
KWD_context         ::= "context";
KWD_def             ::= "def";
KWD_entity          ::= "entity";
KWD_event           ::= "event";
KWD_header          ::= "header";
KWD_value           ::= "value";
KWD_unit            ::= "unit";
KWD_word            ::= "word";

// Group keywords
KWD_end             ::= "end";
KWD_group           ::= "group";
KWD_objective       ::= "objective";

//TP header keywords
TP_header_KWDS      ::= KWD_config | KWD_id |KWD_ref |KWD_role |KWD_req
                    |KWD_tp_summary |KWD_TC |KWD_TD |KWD_TP;

KWD_config          ::= "config";
KWD_id              ::= "id";
KWD_ref             ::= "ref";
KWD_role            ::= "role";
KWD_req             ::= "RQ";
KWD_tp_summary      ::= "summary";
KWD_TC              ::= "TC";
```

```
KWD_TD              ::= "TD";
KWD_TP              ::= "TP";

//TP body (structure) keywords
KWD_ensure          ::= "ensure";
KWD_that            ::= "that";
KWD_response        ::= "then";
KWD_stimulus        ::= "when";
KWD_precondition    ::= "with";
KWD_logical         ::= "and"|"or"|"and not"|"or not";

//Test entity keywords
KWD_IUT             ::= "IUT";
KWD_TESTER          ::= "TESTER";

//Predefined words
predefined_words    ::=
                    // glue words
                          "a"
                        | "an"
                        | "as"
                        | "in"
                        | "is"
                        | "no"
                        | "of"
                        | "the"
                    // logical words
                        | "and"
                        | "not"
                        | "or"
                    // stimulus and response words
                        | "receives"
                        | "sends"
                    // data-related words
                        | "containing"
                        | "indicating"
                    //direction words
                        | "from"
                        | "to"
                    // time- or order-related words
                        | "after"
                        | "before"
                        | "unorderd"
                        | "within";


// Begin/End symbols
begin_stimuli       ::= L_BRACE;
end_stimuli         ::= R_BRACE;
begin_conditions    ::= L_BRACE;
end_conditions      ::= R_BRACE;
begin_responses     ::= L_BRACE;
end_responses       ::= R_BRACE;
begin_tp            ::= L_BRACE;
end_tp              ::= R_BRACE;

// Delimiters, separators etc.
DASH                ::= "-";
DELIM               ::= ":";
DOT                 ::= ".";
F_SLASH             ::= "/";
L_BRACE             ::= "{";
R_BRACE             ::= "}";
L_BRACKET           ::= "[";
R_BRACKET           ::= "]";
L_PAREN             ::= "(";
R_PAREN             ::= ")";
LT                  ::= "<";
RT                  ::= ">";
SEPARATOR           ::= ",";
U_SCORE             ::= "_";

qstring             ::= "'" *("'") "'";
```

# Annex B (informative): Use of the IPT Testing Framework

## B.1 IPT naming conventions

### B.1.1 IPT identifiers

The IP Testing Framework [2] naming conventions provide traceability to other components of a complete test specification. For example, to the base standards, requirements catalogue or PICS, configuration descriptions or Test Cases. These conventions are defined in EG 202 568 [2] but are repeated here for convenience.

If the TPLan user wishes to follow these conventions then the syntax defined in [2] should be used.

### B.1.2 The Requirements Identifier

The Requirements Identifier is of the form **RQ_nnn_mmmm** where "nnn" is a 3-digit number and "mmmm" is a 4-digit number. The Requirements Identifier uniquely identifies a requirement in the corresponding Requirements Catalogue, derived from the relevant base standards. For example:

```
RQ_201_1001
```

In cases where a PICS (or profile PICS) is used rather than a Requirements Catalogue then the reference to the relevant PICS entry will depend on the naming conventions followed by the relevant PICS. A typical example might be:

```
PICS_101_Table1.item3
```

### B.1.3 The Configuration Identifier

The Configuration Identifier is of the form **CF_aa..a_nn** where "aa..a" is an alphanumeric string of length 1 to 8 and "nn" is a 2-digit number (see also clause 7.2). The alphanumeric string may be the same as the TSS identifier in the TSS header (often it will be the same, but not necessarily). The Configuration Identifier uniquely identifies a specific test configuration (if any). For example:

```
CF_MOBILITY_03
```

### B.1.4 The Test Purpose Identifier

The Test Purpose Identifier is of the form **TP_aa..a_nnnn_mm** where "aa..a" is an alphanumeric string of length 1 to 8, "nnnn" is a 4-digit number and "mm" is a 2-digit number. The alphanumeric string should be the same as the TSS identifier in the TSS header. The Test Purpose Identifier uniquely identifies the TP.

```
TP_MOBILITY_0001_99
```

### B.1.5 The Test Case Identifier

The Test Case Identifier is of the form **TC_aa..a_nnnn_mm** where "aa..a" is an alphanumeric string of length 1 to 8, "nnnn" is a 4-digit number and "mm" is a 2-digit number. The alphanumeric string should be the same as the TSS identifier in the TSS header. The Test Case Identifier uniquely identifies a corresponding (TTCN-3) test case (if any).

```
TC_MOBILITY_0001_99
```

### B.1.6 The Test Description Identifier

The Test description Identifier is of the form **TC_aa..a_nnnn_mm** where "aa..a" is an alphanumeric string of length 1 to 8, "nnnn" is a 4-digit number and "mm" is a 2-digit number. The alphanumeric string should be the same as the TSS identifier in the TSS header. The Test Description Identifier uniquely identifies a test description (if any).

```
TD_MOBILITY_0001_99
```

## B.2 IPT cross references

### B.2.1 References to the Requirements Catalogue

When the IPT Testing Framework [2] is used each TP should refer to one or more requirements defined in the relevant Requirements Catalogue. Each requirement is uniquely identified as described in clause B.1. The first three digits in the requirement reference identify the source documents from which a particular set of requirements are derived. This is specified using the **xref** keyword, followed by a list of one or more references to base standards and/or profiles and relevant requirements catalogue.

```
xref RQ_001 {TS123456-1, RFC1234}
.
.
RQ ref: RQ_001_0728

--In this example the requirement RQ_001_0728 identifies requirement 0728
-- extracted from the base standards TS123 456-1 and RFC 1234.
```

### B.2.2 References to test configurations

References to explicit IPT test configurations are made using the keyword **xref** followed by a configuration identifier as defined in clause B.1 followed by a list of one or more references to where the description (e.g. prose and/or drawing) can be found. For example:

```
xref CF_UMTS_007 {3GPPSpecXYZ_AnnexA_page10_fig1}
```

# Annex C (informative): A guide to using TPLan in a communications testing environment

## C.1 General considerations

### C.1.1 Introduction

TPLan is specified in the present document as a generic notation which has only a limited semantic model and has a number of powerful capabilities which make it adaptable to most testing environments. However, this power and its flexibility, if misused, can result in unreadable and meaningless TP specifications. Consequently, it is important to follow some practical guidelines when writing TPLan. This annex offers some guidelines on using TPLan in the production of communications test specifications.

### C.1.2 Structure of a TPLan specification

A complete TPLan specification comprises a Header section followed by the Test Purposes (TPs) themselves.

The TPLan Header section contains the following items:

- The TSS Header:
  - TSS Identifier;
  - TSS Title;
  - version number;
  - date;
  - author.
- Cross-references:
  - to requirements sources;
  - to configuration (abstract architectures) information.
- User-defined extensions to TPLan:
  - header fields;
  - entities;
  - events;
  - values;
  - units;
  - keywords;
  - conditions;
  - syntactical context.

Test Purposes can be grouped together to reflect the Test Suite Structure (TSS) and each TP is specified using the following elements:

- TP Header:
  - TP identifier;
  - a summary of the test;
  - references to the requirements covered by the TP;
  - the role of test subject (IUT or EUT);
  - an identification of the abstract architecture upon which the TP is based;
  - a reference to the Test Case (TC) or Test Description (TD) derived from the TP.
- TP Body:
  - test preconditions;
  - stimulus;
  - response.

## C.1.3 Choosing a suitable text editor

TPLan depends quite heavily on the use of colour to distinguish between different types of keyword. Consequently, if TPs are being developed outside a specific TPLan tool, it is important to choose a text editor that can support user-defined context-sensitive highlighting. Many such editors exist and the one that is best suited to the particular project should be selected. Apart from the ability to use colour highlighting, other selection criteria may include current availability as an installed product, price, support and additional functionality.

Whichever text editor is chosen, it should be configured to provide the colour scheme shown in table C.1 for TPLan specifications, if possible.

**Table C.1: TPLan colour highlighting conventions**

| TPLan element | Font colour | Font weight | Example |
|---|---|---|---|
| TSS Header keywords | purple | bold | Date |
| Definition keywords | purple | bold | def entity |
| Grouping keywords | purple | bold | Group . . . . End Group |
| TP Header keywords | blue | bold | RQ ref |
| TP Body keywords | blue | bold | when |
| Entities | dark red | normal | IUT |
| Events | dark red | normal | Call_Proceeding |
| Event parameters | dark red | normal | source_address |
| Values | dark red | normal | prefix_lifetime |
| Units | dark red | normal | 300 msec |
| Conditions | dark red | normal | away_from_home |
| Numbers | dark red | normal | 1234 |
| Strings | dark grey | normal | 'this is a string' |
| Comments | dark green | normal | -- this is a comment |
| All other text | black | normal | TP_SEC_2345_04 |

# C.2 The TPLan header

## C.2.1 TSS Header

Most of the information specified in the TSS Header is included for the management and control of the TPLan specification and should be maintained conscientiously. This means that:

a) the title field should accurately reflect the contents of the specification;

b) the date and version fields should, together, identify the revision status of the specification; and

c) the author field should identify the group or individual responsible for writing the specification.

However, the purpose of the TSS field is to declare the short string (3 to 8 characters) that should be used in the construction of each TP identifier. For example, if the TSS field is declared as "ABCDE" then all TP identifiers should be of the form "TP_ABCDE_nnnn_mm".

The following example shows a valid TSS Header:

```
TSS     : SEC
Title   : 'IPv6 Security TSS and TP'
Version : 1.1.6
Date    : 25.10.2006
Author  : 'STF276-II'
```

## C.2.2 Cross-references

### C.2.2.1 Requirement sources

Cross references to requirements sources are included in a TPLan specification purely for information and have no semantic meaning within the notation. They provide an opportunity to identify the sources of specific groups of requirements referenced within the TPs. Thus, in the following example, all requirements with identifiers of the form RQ_040_nnnn are derived from the source documents, RFC 1234 and RFC 4567:

```
xref RQ_040 { RFC1234, RFC4567 }
```

The list of sources should be as complete as possible and should not be limited to publicly-available documents. Any that are relevant and from which requirements have been extracted should be included.

### C.2.2.2 Configurations

As with the cross-references to requirements sources, the configuration cross-references are for information only. They provide convenient pointers to files or documents that specify the various abstract architectures upon which the TPs are based. The following example identifies that the configurations CF_SEC_01, CF_SEC_02 and CF_SEC_03 can all be found in the document, Config_IOP_SEC.pdf:

```
xref CF_MOB_02 {Configs_IOP_SEC.pdf}
xref CF_MOB_03 {Configs_IOP_SEC.pdf}
xref CF_MOB_04 {Configs_IOP_SEC.pdf}
```

Again, the list of configurations should be as complete as possible, particularly for interoperability TPs where the abstract architectures are an integral part of the specification.

## C.2.3 User-defined extensions to TPLan

### C.2.3.1 General layout of user definitions

The TPLan notation requires that all user-defined extensions are specified as part of the TPLan Header, i.e. before any TPs are specified, but, within this part of the Header, there is no strict order specified. However, the use of TPLan comments to group similar types of definition together can make the specification easier to read. The structure of the user-defined extension is likely to be dependent upon the project to which the TPLan is related but, as an example, the following list shows how the extensions could be organized:

1. Cross references:
   - Requirements.
   - Configurations.
2. Entities:
   - Test entities (e.g. EUT, QE).
   - Network entities (e.g. destination_node, connection).
   - Addressing entities (e.g. multicast_group, port_21).
3. Events:
   - Messages (e.g. SETUP, IPv6Packet).
   - Timeouts (e.g. max_response_time).
   - User-interface stimuli (e.g. escape_key, Go_command).
   - Procedural events (e.g. transport_mode, connection_establishment).
   - Generic events (e.g. request, response).
4. Conditions:
   - Pre-conditions (e.g. powered-up).
   - States (e.g. idle, away_from_home).
5. Values:
   - Event-related values (e.g. packet headers, payload contents).
   - Literal constants (e.g. status codes, error codes, message types).
   - Counters and timers.
6. Units:
   - Simple measurement (e.g. metres, mille-seconds).
   - Quantitative (e.g. octets, errors).

7. Keywords:
   - Comparators (e.g. equal, more).
   - Qualifiers (e.g. acceptable, modified).
   - Functions (e.g. plus, times).
   - General "glue" words (e.g. at, this).
   - Keywords related to the "**with**" statement (e.g. having, established).
   - Keywords related to the "**when**" statement (e.g. requested, expires).
   - Keywords related to the "**then**" statement (e.g. accepts, resends).

### C.2.3.2 Header fields

TPlan permits the definition of new fields to be used in the TSS Header and the TP Headers. This facility should be used sparingly to add fields that are of particular relevance to a project. In the TSS Header it could be used to add, for example, a **status** field or a **Work Item** reference as shown below:

```
TSS      : SEC
Title    : 'IPv6 Security TSS and TP'
WI ref   : 'DTS/MTS-IPT-010-IPv6-SecTCSS'
Version  : 1.1.6
Status   : 'Draft'
Date     : 25.10.2006
Author   : 'STF276-II'
. . . .
. . . .
def header WI
def header Status
```

In those projects that use an Implementation Conformance Statement (ICS) as a reference document for the requirements rather than or as well as a requirements catalogue, it is convenient to define a new field for the TP Header, **PICS**, for example. This can then be used in place of the **RQ ref** field, thus:

```
def header PICS
. . . .
. . . .
TP id    : TP_SEC_2009_01
summary  : 'Test reaction on multicast IPv6 packets for unknown
            multicast_group SA'
PICS ref: B.5, D.23
role     : Ipsec_host
config   : CF_SEC_01
TC ref   : TC_SEC_2009_01
```

### C.2.3.3 Entities

Although the primary purpose of the **def entity** construct is enable the identification of test entities such as the EUT and QEs, it is also useful for defining other architectural and addressing items. Examples of possible entities of this type are shown in the following list:

1. Architectural:
   - destination_node;
   - B_Channel.
2. Addressing:
   - multicast_group;
   - UDP_port_500.

### C.2.3.4 Events

The concept of an event within TPLan is not restricted. It could, for example, be a protocol message, a timeout or a procedure invocation. Generally, they can be considered to be associated with stimuli and responses and usually require the presence of additional keywords to describe a complete action. For example, tests often involve message events which need to be associated with a **send** or **receive** keyword. The following examples show the different ways in which events can be used:

```
  when { IUT receives SETUP . . . . }
  then { IUT sends CALL_PROCEEDING . . . . }

  when { sanity_timer expires in the IUT. . . }

  when { EUT is requested to establish a call to QE1 . . . }
```

TPLan events may have parameters associated with them. The purpose of this is to make it possible to identify fields within messages and other events. The use of such parameters can improve the readability of a TPLan specification quite considerably, as the following example shows:

```
def event SETUP
          { source_address,
            destination_address,
            A_flag }
 . . . .
  when { IUT receives SETUP
              containing source_address indicating an external_user
            and containing A_flag set to 1 }
  then { . . . . }
```

In those cases where a parameter, itself, contains additional fields (for example, a packet header), these fields should be identified in a **def value** statement (see clause C.2.3.6), as follows:

```
def event IPv6Packet
          { IPv6_Header,
            Routing_header,
            payload }

def value IPv6_header
          { Version,
            Traffic_Class,
            Payload_Length,
            source_address,
            destination_address }
 . . . .
  when { IUT receives an IPv6Packet
              containing an IPv6_header
                          containing source_address indicating a link_local_address
  then { . . . . }
```

The parameter field can also be useful in identifying the length of a timeout event, as follows:

```
def event response_time {100mSec}
```

### C.2.3.5 Conditions

The **def condition** statement in TPLan makes it possible to identify the various states that a test entity can reach. A condition identifier can either be used within the TP Body in conjunction with a user-defined **state** keyword or in the preconditions (**with** statement) without the **state** keyword, as follows:

```
def condition away_from_home
def condition idle
def word state
 . . . .
  with { EUT away_from_home }
  ensure that
       { when {. . . . . }
         then {     EUT . . . .
                  and EUT enters the idle state }
       }
```

### C.2.3.6 Values

Within TPLan, a value can be a literal, a constant (e.g. `Invalid_Format`, `Avogadros_Number`), a value identifier (e.g. `repeat_count`, `message_ID`) or any other value-related item.

When defining the literal constants which are often associated with protocol status or error codes, for example, it is useful to include the numerical value of the constant as a comment, thus:

```
--** Configuration Types
def value CFG_REQUEST                    -- 1
def value CFG_REPLY                      -- 2

--** Notify Message Types
def value UNSUPPORTED_CRITICAL_PAYLOAD   -- 1
def value INVALID_IKE_SPI                -- 4
def value INVALID_MAJOR_VERSION          -- 5
def value INVALID_SYNTAX                 -- 7
def value INVALID_MESSAGE_ID             -- 9

def value ADDITIONAL_TS_POSSIBLE         -- 16386
def value IPCOMP_SUPPORTED               -- 16387

def value generic_payload_header
          { next_payload,
            Critical_flag,
            payload_length }
```

### C.2.3.7 Units

Although TPLan allows specific combinations of numbers and units to be defined as **`values`** (e.g. `30 sec`) this approach is not convenient in all cases. In those instances where a TPLan specification includes many different numeric values associated with the same units then these units should be defined using the **`def unit`** construct as follows:

```
def unit msec      'mille-seconds'
. . . . .
  then { IUT sends CALL_PROCEEDING after 100 msec }
```

### C.2.3.8 Keywords

Although the TPLan notation standard includes a base set of useful keywords, it is quite likely that each TP specification will require the definition extra keywords. There are generally two reasons for adding new TPLan keywords:

1. to extend the functional capabilities of TPLan, for example:
   - **`starts`**;
   - **`established`**;
   - **`registered`**.
2. to add words that improve readability, for example:
   - **`at`**;
   - **`for`**;
   - **`this`**.

Although the addition of new keywords can make the TPLan specification considerably easier to understand, care should be taken to avoid adding multiple words with almost identical meaning. Also, the **`def context`** construct (see clause 8.8) should be used wherever possible to limit the use of newly-defined keywords.

### C.2.3.9 Syntactical context

In order to avoid the use of newly-defined keywords in meaningless combinations, TPLan has a facility for defining the specific syntactical context(s) in which a keyword may be used. This capability should be used extensively to avoid the misuse of user-defined extensions. Within a **def context** statement, square brackets around a word indicates that it is optional within the defined context, a preceding tilde (~) character indicates that the word may only be used in this context (it is, however, possible to include the same word in more than one context). The following example shows how **def context** statements can be constructed:

```
def word established
  . . .
def context {~established as }
def context {[not]~having ~established }
```

The result of these statements is that the keyword established can only be used in the following constructs:

```
  . . . . established as
  . . . . not having established
  . . . . having established
```

NOTE: The **def context** construct should only be used to define the syntactical context associated with particular user-defined words. It should not be used to construct entity, event, condition or value identifiers which contain white space; i.e. in the following example, the definition of the away_from_home condition is semantically different from the result of the context definition which permits the use of the construct, **away from home**:

```
def condition away_from_home

def context {~away from ~home }
```

## C.3 Test Purposes

### C.3.1 Grouping TPs

Each TP in a test specification is usually allocated to one or other of the groups in the Test Suite Structure (TSS). TPLan allows this grouping to be expressed using its **group** and **end group** statements.

Each group of TPs should be given a unique number and a title which accurately reflects the nature of the grouping. group numbers should be in the "legal" form (i.e. 1, 1.1, 1.1.1 etc.) as shown in the following example:

```
Group 2 'Basic communications functions'
  . . . .
Group 2.1 'Sending SETUP'
  . . . .
End Group 2.1
Group 2.2 'Receiving SETUP'
  . . . .
Group 2.2.1 'Sending CALL_PROCEEDING'
  . . . .
End Group 2.2.1
End Group 2.2
End Group 2
```

If the more traditional approach to naming TSS groups is taken (i.e. the group title is based upon the test path) EG 202 568 [2] where there is less functional information in the group title, the optional **objective** statement should be used to add a more meaningful title, as follows:

```
Group 2 'Basic Call (BC)
Objective 'Basic communications functions'
  . . . .
Group 2.1 'Basic Call (BC) / Originating Exchange (OE)
Objective 'Sending SETUP'
  . . . .
End Group 2.1
Group 2.2 'Basic Call (BC) / Terminating Exchange (TC)
Objective 'Receiving SETUP'
  . . . .
Group 2.2.1 'Basic Call (BC) / Terminating Exchange (TC) / Response to SETUP (RS)
```

```
Objective 'Sending CALL_PROCEEDING'
  . . . .
End Group 2.2.1
End Group 2.2
End Group 2
```

## C.3.1.1 TP header

Each TPLan Test Purpose begins with a header which should contain all of the following information:

- TP Identifier:

  The TP identifier should conform to the guidelines specified in EG 202 568 [2] and should include the TSS identifier as shown in the following example:

  ```
  TSS       : DEMO
  . . . . . . . . .
  TP id     : TP_DEMO_1234_03
  . . . . . . . . .
  ```

- Summary of the TP:

  The TP summary is a string which should briefly describe the basis for the test. The summary in each TP should be unique within the TSS so that even in those cases where a number of TPs are derived from a single set of requirements, the summaries help to highlight the differences between each TP. For example:

  ```
  TP id     : TP_DEMO_1234_03
  Summary   : 'Test the response of a host device to something (unicast address)'
  . . . . . . . . .
  TP id     : TP_DEMO_1234_04
  Summary   : 'Test the response of a host device to something (multicast address)'
  . . . . . . . . .
  ```

- Identification of the requirement source:

  It is important to know which base requirements each TP aims to test so it is essential that all relevant requirements are listed, as shown in the following example:

  ```
  TP id     : TP_DEMO_1234_03
  Summary   : 'Test the response of a host device to something (unicast address)'
  RQ ref    : RQ_204_3001, RQ_204_3002, RQ_316_0593, RQ_316_0619, RQ_316_0620,
              RQ_450_5261, RQ_450_6372
  . . . . . . . . .
  ```

- The role of the IUT or EUT:

  As TPs are generally expressed in terms of the IUT or EUT, this field is necessary in order to identify what functional role the IUT or EUT is expected to play in the test. If the test applies to more than one role, then all applicable roles should be listed, as shown in the following example:

  ```
  TP id     : TP_DEMO_1234_03
  Summary   : 'Test the response of a host device to something (unicast address)'
  RQ ref    : RQ_204_3001, RQ_204_3002, RQ_316_0593, RQ_316_0619, RQ_316_0620,
              RQ_450_5261, RQ_450_6372
  Role      : Host, Router
  . . . . . . . . .
  ```

  The role should not be confused with the physical entity playing the role. In the example above, a personal computer (physical entity) could be configured to operate as either a Host or a Router (functional role).

- The abstract architecture associated with the TP:

  The relevant configuration from those specified in the cross-reference statements in the TPLan Header (clause C.2.2.2) should be identified here, as follows:

```
TP id     : TP_DEMO_1234_03
Summary   : 'Test the response of a host device to something (unicast address)'
RQ ref    : RQ_204_3001, RQ_204_3002, RQ_316_0593, RQ_316_0619, RQ_316_0620,
            RQ_450_5261, RQ_450_6372
Role      : Host, Router
Config    : CF_DEMO_17
. . . . . .
```

- The identifier of the Test Case or Test Description which implements the TP:

  Each TP is likely to have a Test Case and/or a Test Description derived from it. The TC or TD identifier, if it exists, should be identified here. If both a TD and a TC exist for a particular TP then only the TD reference should be included:

```
TP id     : TP_DEMO_1234_03
Summary   : 'Test the response of a host device to something (unicast address)'
RQ ref    : RQ_204_3001, RQ_204_3002, RQ_316_0593, RQ_316_0619, RQ_316_0620,
            RQ_450_5261, RQ_450_6372
Role      : Host, Router
Config    : CF_DEMO_17
TD ref    : TD_DEMO_1234_03
. . . . . .
```

## C.3.1.2 TP Body

### C.3.1.2.1 Preconditions

In most TPs there will be conditions that need to be defined before the test itself is specified. These preconditions are specified in the TPLan **with** statement and should, in general, be status-related rather than dynamic, as shown in the following examples:

```
with { EUT away_from_home }

with { IUT having sent SETUP }

with { EUT configured 'to perform route optimization' }
```

It is possible to express multiple and complex conditions by using the logical **and**, **not** and **or** functions, thus:

```
with {     EUT away_from_home
       and EUT registered to QE4 }

with {     IUT having sent SETUP
       and IUT not having received CALL_PROCEEDING }
```

User-defined keywords and text within string-quotes should be chosen to fit grammatically in the **with** statement so that it is easy to read as a correct English phrase. As an example, the following statement is not acceptable:

```
with { IUT establishes a Security_Association }
```

However, the following similar statement is acceptable:

```
with { IUT established in a Security_Association }
```

### C.3.1.2.2 Stimulus and response

#### C.3.1.2.2.1 The **when** and **then** construct

Tests are usually specified as a combination of a stimulus followed by an expected response. In a TPLan specification, these are represented by the **when** and the **then** statements, respectively. Consequently, although it is possible to have more than one **when** and/or **then**, it is not possible to have a **when** statement without a corresponding **then** statement.

TPLan specifications can be written with only a few user-defined extensions (see clause C.2.3) by making extensive use of quoted strings. Such specifications are generally not difficult to read but the opportunities for automatically checking the specification and, possibly, for transposing it into a test case specification language such as TTCN-3, are limited. It is, therefore, beneficial to define sufficient notation extensions to make the use of quoted strings the exception rather than the rule.

All TPs should be written in terms of the IUT or EUT and the other test entities (i.e. the TESTER or the QEs). The following example shows the specification of a simple stimulus and response:

```
ensure that {
               when { IUT receives SETUP }
               then { IUT sends CALL_PROCEEDING }
             }
```

#### C.3.1.2.2.2 Identifying the contents of message events

In most instance, test stimuli (and, to a lesser extent, responses) are based on the status of the parameters of an event rather than the event alone. The **containing** and **indicating** keywords are used for this purpose, as follows:

```
when { IUT receives SETUP
            containing source_address
            indicating external_caller }
. . . . . .
```

If it is necessary to be more specific about the contents of an event parameter, the readability of the specification can be improved by substituting the **indicating** keyword with a **set to** construct which is defined and used thus:

```
def word set
def context { ~set to }
. . . . . .
when { IUT receives an IPv6packet
            containing Hop_Limit set to 0 }
. . . . . .
```

A similar approach can be used in a response:

```
. . . . . .
then { IUT sends an ICMPv6packet
            containing an Error_Message
                   set to PACKET_TOO_BIG }
. . . . . .
```

Logical operators can be used to build stimuli and responses from multiple parameters values as the following example shows:

```
when { IUT receives SETUP
            containing source_address
            indicating external_caller
       and containing destination_address
       not indicating the address of the IUT }
. . . . . .
```

It is possible that message event parameters may, themselves, contain parameters (see clause C.2.3.4). This is particularly true in packet-based protocols such as IPv6 where, in addition to a number of simple parameters, a packet may contain:

- headers;
- payloads; and/or
- encapsulated (tunnelled) packets.

In these instances, round braces and suitable indentation should be used to improve the clarity of the TPLan specification. For example:

```
ensure that
     { when { IUT receives CREATE_CHILD_SA_request
                 containing (Traffic_Selector_payload
                             containing more than 1 Traffic_Selector) }
       then { IUT sends CREATE_CHILD_SA_response
                 containing (Notify_payload
                             containing Notify_Message_Type
                                 set to SINGLE_PAIR_REQUIRED) }
     }
```

#### C.3.1.2.2.3 Interactions with the user

It is not unusual for an event to be stimulated by the notional user of the IUT or EUT or for a response to involve a report to the user. These are highlighted in the following examples which both require the addition of some user-defined extensions to TPLan:

- User-initiated stimulus:

```
def word requested
def word send
def context { is ~requested to ~send }
. . . . . .
  when { the EUT is requested to send Echo_Request }
. . . . . .
```

- Report-to-user response:

```
def word indicates
def word receipt
def context { ~indicates ~receipt of }
. . . . . .
  then { the EUT indicates receipt of the packet }
. . . . . .
```

#### C.3.1.2.2.4 Establishing the order of a sequence of events

Although logically associated multiple events are treated by TPLan as ordered by default, it is possible to emphasize a strict sequence of events within a stimulus or response by using **`before`** and **`after, particularly`** if the sequence comprises only a small number of message events (no more than 3), as follows:

```
when {        IUT receives message_1
       before IUT receives message_2 }
```

Using the **`after`** keyword to link a number of message events together in a sequence can be confusing (because the messages appear in reverse order) but it is very useful for associating a message event with a timer event, thus:

```
then {       IUT sends message_1
       after timer_3 expires }
```

When there are a larger number of events in a sequence, these should be specified as a list of events which are, by default, ordered. Although this results in poorer English phrasing, it is considerably less complex than a long series of events linked together with either **`before`** or **`after`**:

```
when {     IUT receives message_1
       and IUT receives message_2
       and IUT receives message_3
       and IUT receives message_4
       and IUT receives message_5
     }
```

Round braces should always be used to indicate the extent of the ordered sequence. For example, the **when** statement above specifies a stimulus which is not the same as the stimulus specified in the following TPLan:

```
when { (     IUT receives message_1
         and IUT receives message_2
         and IUT receives message_3
         and IUT receives message_4
       )
       and IUT receives message_5
     }
```

#### C.3.1.2.2.5 The "do nothing" response

When the expected response to a particular stimulus is to do nothing, user-defined extensions should be specified so that a clear statement can be made, thus:

```
. . . . . .
def event response
def context { sends no ~response }
. . . . . .
. . . . . .
  then { IUT sends no response }
```

# Annex D (informative): Some communications testing examples

## D.1 IPv6 Interoperability Test Purposes

```
TSS      : COR
Title    : 'RFC2460 IPv6 Core Specification'
Version  : 1.0.1
Date     : 05.10.2006
Author   : 'Steve Randall (ETSI TC-MTS)'

-- Cross references
xref RQ_001 {RFC2460, RFC2461}
xref CF_COR_11 {ETSI_TS_102_517_Annex_B}
xref CF_COR_21 {ETSI_TS_102_517_Annex_B}
xref CF_COR_23 {ETSI_TS_102_517_Annex_B}

-- Definitions
-- Entities
def entity EUT
def entity QE1
def entity QE2

-- Messages
def event data{packet_length}
def event ICMP_error_message {parameter_problem}
def event packet
          { source_address,
            destination_address,
            routing_header,
            hop_by_hop_options,
            Hop_Limit,
            flow_label,
            Type_0_routing_header,
            EUT_address,
            request_for_response}

-- Values
def value Path_MTU
def value Type_0_routing_header
          { Next_Header,
            Header_Extension_Length,
            Routing_Type_0,
            Segments_Left,
            Addresses }

-- Units
def unit octets

-- Keywords - Pre-conditions
def word configured

-- Keywords - Stimuli
def word indicates
def word requested
def word requiring
def word send
def context {is ~requested to}

-- Keywords - Responses
def word decrements
def word discards
def word receipt
def word response
def word unchanged
def context {sends no ~response}
def context {sends a valid ~response}

-- Keywords - Glue
def word between
def word exactly
```

```
def word greater
def word less
def word same
def word than
def word valid

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

Group 1 'RFC2460'
Group 1.1 'Process IPv6 Packet'
Group 1.1.1 'Process IPv6 Header'

TP id   : TP_COR_1097_01
Summary : 'EUT processes a packet with its size equals to its link MTU'
RQ ref  : RQ_001_1097
Config  : CF_COR_11
TD ref  : TD_COR_1097_01

with { QE1 configured 'with a unique global unicast address '
   and EUT configured 'with a unique global unicast address'
   and EUT 'having a link MTU smaller than the link MTU of QE1'
     }
ensure that {
  when { EUT receives a packet 'with its size equal to link MTU of EUT'
           containing QE1 as the source_address
       and containing EUT as the destination_address
       and containing a request_for_response }
  then { EUT sends a valid response to QE1 }
           }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1097_02
Summary : 'EUT processes a traversed packet with its size equals to its
           incoming link MTU'
RQ ref  : RQ_001_1097
Config  : CF_COR_21
TD ref  : TD_COR_1097_02

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the link
                       connecting QE1 and EUT, and, the link connecting QE2
                       and EUT, respectively'
   and QE1 'having larger link MTU than EUT'
   and EUT 'having larger or equivalent link MTU than QE2'
     }
ensure that {
  when { EUT receives a packet 'with its size equal to its
                                incoming link MTU'
           containing QE1 as the source_address
       and containing QE2 as the destination_address }
  then { EUT sends the packet to QE2 }
           }

Group 1.1.1.1 'Process Hop Limit'

TP id   : TP_COR_1002_01
Summary : 'EUT decreases the Hop Limit field of a traversed IPv6 packet and
           forwards it'
RQ ref  : RQ_001_1002
Config  : CF_COR_21
TD ref  : TD_COR_1002_01

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the
                       link connecting QE1 and EUT, and the link connecting
                       QE2 and EUT, respectively'
     }
ensure that {
  when { EUT receives a packet
           containing QE1 as the source_address
       and containing QE2 as the destination_address
       and containing a Hop_Limit greater than 1 }
  then { EUT decrements the Hop_Limit
     and EUT sends the packet to QE2 }
          }
```

```
--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1002_02
Summary : 'EUT drops a traversed IPv6 packets with a zero Hop Limit and
           returns an ICMP error message to the source'
RQ ref  : RQ_001_1002
Config  : CF_COR_21
TD ref  : TD_COR_1002_02

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the
                       link connecting QE1 and EUT, and on the link connecting
                       QE2 and EUT, respectively'
     }
ensure that {
  when { EUT receives a packet
           containing QE1 as the source_address
       and containing QE2 as the destination_address
       and containing a Hop_Limit of 0 }
  then { EUT discards the packet
     and EUT sends an ICMP_error_message to QE1 }
           }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1058_01
Summary : 'EUT drops a packet with a Type 0 routing header and Hop Limit<=1
           and returns an ICMP error message to the source'
RQ ref  : RQ_001_1058
Config  : CF_COR_21
TD ref  : TD_COR_1058_01

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the
                       link connecting QE1 and EUT, and on the link connecting
                       QE2 and EUT, respectively'
     }
ensure that {
  when { EUT receives a packet
           containing QE1 as source_address
       and containing QE2 as destination_address
       and containing a Type_0_routing_header
       and containing a Hop_Limit less than 2 }
  then { EUT discards the packet
     and EUT sends an ICMP_error_message to QE1 }
           }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1059_01
Summary : 'EUT forwards a traversed packet with a Type 0 routing header
           and Hop Limit > 1'
RQ ref  : RQ_001_1059
Config  : CF_COR_21
TD ref  : TD_COR_1059_01

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the
                       link connecting QE1 and EUT, and on the link connecting
                       QE2 and EUT, respectively'
     }
ensure that {
  when { EUT receives a packet
           containing QE1 as the source_address
       and containing QE2 as the destination_address
       and containing a Type_0_routing_header
                      containing Addresses indicating the EUT_address
       and containing Hop_Limit greater than 1 }
  then { EUT decrements the Hop_Limit
     and EUT sends the packet to QE2 }
           }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--
```

```
End Group 1.1.1.1

Group 1.1.1.2 'Process Flow Label'

TP id   : TP_COR_1130_01
Summary : 'EUT detects two packets with different hop-by-hop option contents
           but the same source and destination addresses in the flow label'
RQ ref  : RQ_001_1130
Config  : CF_COR_21
TD ref  : TD_COR_1130_01

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on the link
                       connecting QE1 and EUT and, the link connecting QE2 and
                       EUT, respectively'
     }
ensure that {
  when { EUT receives packet 1
           containing QE1 as the source_address in the flow_label
       and containing QE2 as the destination_address in the flow_label
     and EUT receives packet 2
           containing hop_by_hop_options not the same as in packet 1
       and containing QE1 as the source_address in the flow_label
       and containing QE2 as the destination_address in the flow_label }
  then { EUT sends an ICMP_error_message
           indicating a parameter_problem to QE1
     and EUT discards packet 1
     and EUT discards packet 2 }
            }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1130_02
Summary : 'EUT detects two packets with different routing header contents but
           the same source and destination addresses in the flow label'
RQ ref  : RQ_001_1130
Config  : CF_COR_21
TD ref  : TD_COR_1130_02

with { QE1 configured 'with a unique global unicast address '
   and QE2 configured 'with a unique global unicast address'
   and EUT configured 'with two unique global unicast addresses on
                       the link connecting QE1 and EUT and
                       the link connecting QE2 and EUT, respectively'
     }
ensure that {
  when { EUT receives packet 1
           containing QE1 as the source_address in the flow_label
       and containing QE2 as the destination_address in the flow_label
     and EUT receives packet 2
           containing a routing_header not the same as in packet 1
       and containing QE1 as the source_address in the flow_label
       and containing QE2 as the destination_address in the flow_label }
  then { EUT sends an ICMP_error_message
           indicating a parameter_problem to QE1
     and EUT discards packet 1
     and EUT discards packet 2 }
            }

End Group 1.1.1.2
End Group 1.1.1
End Group 1.1

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

Group 1.2 'Generate Extension Headers'
Group 1.2.1 'Generate Fragmented Packets'

TP id   : TP_COR_1064_01
Summary : 'EUT fragments a packet larger than the available Path MTU before
           sending it'
RQ ref  : RQ_001_1064
Config  : CF_COR_23
TD ref  : TD_COR_1064_01

ensure that {
        when { EUT is requested to send data requiring a packet_length
```

```
                    greater than the Path_MTU to QE1 }
          then { QE2 indicates receipt of the same data unchanged }
               }

End Group 1.2.1

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

Group 1.2.2 'Process Fragmented Packets'

TP id   : TP_COR_1100_01
Summary : 'EUT reassembles a fragmented packet of an original length less
           than 1500 octets'
RQ ref  : RQ_001_1100
Config  : CF_COR_23
TD ref  : TD_COR_1100_01

with { 'the MTU on the path from QE1 towards the EUT set at 1280 octets' }
ensure that {
       when { QE1 is requested to send data requiring a packet_length
                  of between 1288 octets and 1492 octets to the EUT }
       then { EUT indicates receipt of the same data unchanged }
            }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1100_02
Summary : 'EUT reassembles a fragmented packet of an original length equal
           to 1500 octets'
RQ ref  : RQ_001_1100
Config  : CF_COR_11
TD ref  : TD_COR_1100_02

with { 'the MTU on the path from QE1 towards the EUT set at 1280 octets' }
ensure that {
       when { QE1 is requested to send data requiring a packet_length
                  of exactly 1500 octets to EUT }
       then { EUT indicates receipt of the same data unchanged }
            }

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : TP_COR_1101_01
Summary : 'EUT reassembles a fragmented packet of an original length
           greater than 1500 octets'
RQ ref  : RQ_001_1101
Config  : CF_COR_11
TD ref  : TD_COR_1101_01

with { 'the MTU on the path from QE1 towards the EUT set at 1280 octets' }
ensure that {
       when { QE1 is requested to send data requiring a packet_length
                  of greater than 1500 octets to EUT }
       then { EUT indicates receipt of same data unchanged }
            }
End Group 1.2.2
End Group 1.2
End Group 1
```

# D.2 QSIG Interoperability Test Purposes

```
TSS     : BS
Title   : 'QSIG Interoperability: Basic Service with Call Forward,
           Call Transfer and Call Completion Supplementary Services'
Version : 1.2
Date    : 15.08.2001
Author  : 'Steve Randall'

-- Cross References

xref PICS_01 {ECMA_143}
xref CF_BS_01 {PQM_BS_IOTv1b}

-- Definitions
-- Header fields
```

```
def header PICS

-- Entities
def entity user_A       -- Connected to PINX Equipment Under Test
def entity user_B       -- Connected to Qualified Equipment PINX
def entity user_C

-- Values
def value Line_Identity

-- Keywords - Pre-conditions
def word configured

-- Keywords - Stimuli
def word call           -- causing a call setup to be sent
def word requested
def context {is ~requested to}

-- Keywords - Responses
def word answers
def word communicate
def word presents

-- Keywords - Glue
def word can

--xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Group 1 'QSIG Interoperability Tests'
Group 1.1 'Basic Service'
Group 1.1.1 'Simple call set-up'

TP id     : TP_BS_001
Summary   : 'EUT PINX supports outgoing call establishment with en-bloc sending'
PICS ref  : PICS_01.B6
Config    : CF_BS_01
TD ref    : TD_BS_001

with {  user_A configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_A configured 'to send address information in en-bloc
                           sending mode'
    and user_B configured 'to receive address information in en-bloc
                           sending mode'
     }
ensure that {
  when  { user_A is requested to call user_B
      and user_B answers }
  then  { user_A and user_B can communicate }
        }

--xxxxxxxxxxxxxxxxxxxxxxx

TP id     : TP_BS_002
Summary   : 'EUT PINX supports incoming call establishment with en-bloc sending'
PICS ref  : PICS_01.B6
Config    : CF_BS_01
TD ref    : TD_BS_001

with {  user_A configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'to send address information in en-bloc
                           sending mode'
    and user_A configured 'to receive address information in en-bloc
                           sending mode'
     }
ensure that {
  when  { user_B is requested to call user_A
      and user_A answers }
  then  { user_A and user_B can communicate }
        }

--xxxxxxxxxxxxxxxxxxxxxxx

TP id     : TP_BS_003
Summary   : 'EUT PINX supports outgoing call establishment with
             overlap sending'
PICS ref  : PICS_01.B6
Config    : CF_BS_01
```

```
TD ref    : TD_BS_001

with {  user_A configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_A configured 'to send address information in overlap
                           sending mode'
    and user_B configured 'to receive address information in overlap
                           sending mode'
     }
ensure that {
  when  { user_A is requested to call user_B
      and user_B answers }
  then  { user_A and user_B can communicate }
        }

--xxxxxxxxxxxxxxxxxxxxxx

TP id     : TP_BS_004
Summary   : 'EUT PINX supports incoming call establishment with
             overlap sending'
PICS ref  : PICS_01.B6
Config    : CF_BS_01
TD ref    : TD_BS_001

with {  user_A configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'to send address information in overlap
                           sending mode'
    and user_A configured 'to receive address information in overlap
                           sending mode'
     }
ensure that {
  when  { user_B is requested to call user_A
      and user_A answers }
  then  { user_A and user_B can communicate }
        }

End Group 1.1.1

--xxxxxxxxxxxxxxxxxxxxxx

Group 1.1.2 'Call set-up with line identities'

TP id     : TP_BS_005
Summary   : 'EUT PINX supports incoming call establishment with
             Connected Line identity'
PICS ref  : PICS_01.J8
Config    : CF_BS_01
TD ref    : TD_BS_005

with {  user_A configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_B configured 'with Bearer Capability set to "Speech, 64kbit/s" '
    and user_A configured 'to present the Connected Line Identity
                           on connection'
    and user_B configured 'to allow the presentation of its Line Identity
                           on connection'
     }
ensure that {
  when  { user_A is requested to call user_B
      and user_B answers }
  then  { user_A presents the Line_Identity from user_B }
        }

End Group 1.1.2
End Group 1.1
End Group 1
```

# D.3 ISDN Conformance Test Purposes

```
TSS     : CW
Title   : 'ISDN DSS1 Call Waiting Supplementary Service'
Version : 1.1
Date    : 05.10.2006
Author  : 'ETSI STC-SPS5'

--***Cross references***
xref CW_U {ETS_300_058_1}

--***Definitions***

-- Messages
def event SETUP {Channel_identification_IE}
def event ALERTING
def event RELEASE_COMPLETE {Cause_IE}

-- Values
def value no_B_channel_available
def value no_circuit_or_channel_available

-- Conditions (ISDN states)
def condition Busy      'ISDN defined Busy state'
def condition Null      'ISDN defined NULL state'
def condition U00       'Sub-state of NULL'
def condition information_channel_control   'Any call establishment state'

-- Keywords

def word compatible
def word valid
def word state

--xxxxxxxxxxxxxxxxxxxxxxxxx--


Group 1 'User (S/T)'
Group 1.1 'Valid behaviour'

TP id   : CW_U01_001
Summary : 'A busy IUT with an available B-Channel responds to an incoming SETUP'
RQ ref  : 9.5.1
Role    : user

with {      IUT in the Busy state
       and 'at least one B-Channel free to the IUT'
     }
ensure that
   {
     when { the IUT receives a valid and compatible SETUP from the TESTER}
     then { the IUT sends ALERTING to the TESTER }
   }

--xxxxxxxxxxxxxxxxxxxxxxxxx--

TP id   : CW_U01_002
Summary : 'A busy IUT with information channel control but no B-Channel responds
           to an incoming SETUP'
RQ ref  : 9.5.1
Role    : user

with {      IUT in an information_channel_control state
       and 'no B-Channel free to the IUT'
     }
ensure that
   {
     when { the IUT receives a valid and compatible SETUP from the TESTER
                    containing a Channel_identification_IE
                    indicating no_B_channel_available }
     then { the IUT sends ALERTING to the TESTER }
   }

--xxxxxxxxxxxxxxxxxxxxxxxxx--
```

```
TP id   : CW_U01_003
Summary : 'A free IUT with no resources available responds to an incoming SETUP'
RQ ref  : 9.5.2
Role    : user

with {      the IUT in the Null state U00
       and the IUT 'having all resources in use'
     }
ensure that
   {
     when { the IUT receives a valid and compatible SETUP from the TESTER
                    containing a Channel_identification_IE
                    indicating no_B_channel_available }
     then { the IUT sends a RELEASE_COMPLETE to the TESTER
                    containing a Cause_IE
                    indicating no_circuit_or_channel_available }
   }
End Group 1.1
End Group 1
```

# Annex E (informative): Bibliography


- ETSI EG 202 237: "Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic approach to interoperability testing".

# History

| Document history | | |
|---|---|---|
| V1.1.1 | February 2008 | Publication |
| V1.2.1 | March 2009 | Membership Approval Procedure MV 20090529: 2009-03-31 to 2009-05-29 |
| V1.2.1 | June 2009 | Publication |
| | | |
| | | |