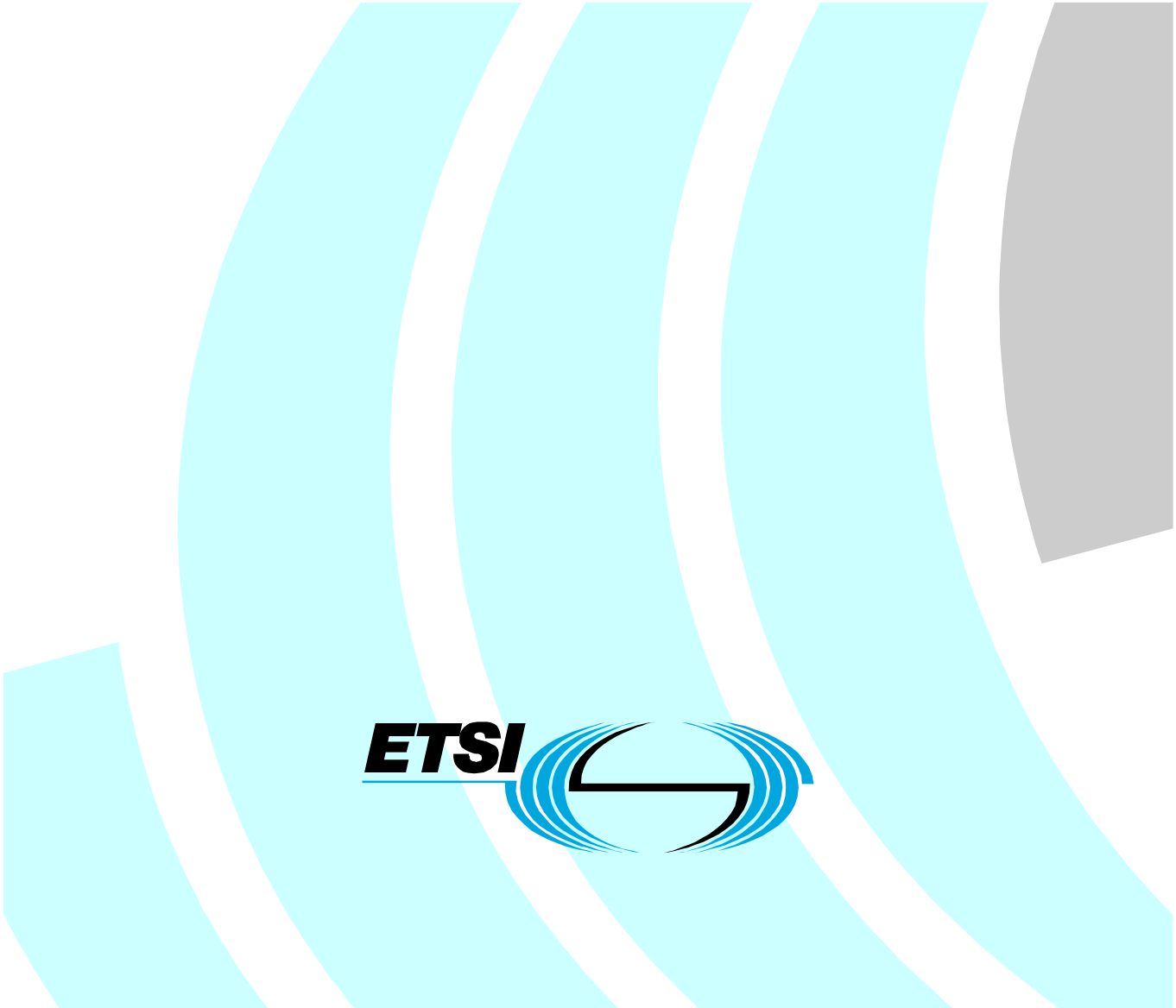
ETSI TS 102 351 V1.1.1 (2004-09)

Technical Specification

Methods for Testing and Specification (MTS);
IP Testing;
TTCN-3 IPv6 Test Specification Toolkit



Reference
DTS/MTS-00092

Keywords
IP, interoperability, 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

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

1 Scope

The purpose of the present document is to provide broad guidelines on the use of a common method for developing test specifications for IPv6. This method is applicable to all IPv6 categories including the core specification, mobility, security and transitioning to IPv6 from IPv4.

The underlying method is based on the methodologies specified in ISO/IEC 9646-1 [4] for conformance tests and ETSI TS 102 237-1 [1] for interoperability tests. It provides guidance on the development and use of the following key elements of the method:

- a Requirements Catalogue (RC);
- a Test Suite Structure (TSS) and Test Purposes (TP);
- Test Descriptions (TD) - interoperability;
- a TTCN-3 library of data types and values, templates and functions;
- an Abstract Test Suite (ATS) - conformance.

The method also offers guidance on a naming convention and other style-related issues.

Although the present document has been developed primarily for use in the testing of IPv6 standards, it could equally be used in other areas of protocol test specification.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- 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.
- For a non-specific reference, the latest version applies.

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

[1] ETSI TS 102 237-1 (2003): "Telecommunications and Internet Protocol Harmonization Over Networks (TIPHON) Release 4; Interoperability test methods and approaches; Part 1: Generic approach to interoperability testing".

[2] ETSI EG 202 106 (2003): "Methods for Testing and Specification (MTS); Guidelines for the use of formal SDL as a descriptive tool".

[3] ETSI ES 201 873-6 (2003): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)".

[4] ISO/IEC 9646-1 (1992): "Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 1: General concepts".

[5] IETF RFC 2460 (1998): "Internet Protocol, Version 6 (IPv6) Specification".

[6] IETF RFC 1035 (1997): "Domain names - implementation and specification".

3 Definitions and abbreviations

3.1 Definitions

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

behavioural function: a TTCN-3 function which specifies actions which result in the sending of messages to one or more observed interface

computational function: a TTCN-3 function which specifies actions which modifies data values but does not result in the sending of messages to one or more observed interface

Equipment Under Test (EUT): grouping of one or more devices which has not been previously shown to interoperate with previously Qualified Equipment (QE)

Qualified Equipment (QE): grouping of one or more devices that has been shown, by rigorous and well-defined testing, to interoperate with other equipment

NOTE: Once an EUT has been successfully tested against a QE, it may be considered to be a QE, itself.

3.2 Abbreviations

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

3GPP	3rd Generation mobile Partnership Project
API	Application Programming Interface
ATS	Abstract Test Suite
EUT	Equipment Under Test
IETF	Internet Engineering Task Force
IFS	Interoperable Functions Statement
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IUT	Implementation Under Test
MTC	Main Test Component
NGN	Next Generation Network
PICS	Protocol Implementation Conformance Statement
PTC	Parallel Test Component
QE	Qualified Equipment
RC	Requirements Catalogue
RFC	Request For Comments (IETF terminology for a draft standard)
RQ	Requirement
SUT	System Under Test
TISPAN	ETSI technical body with responsibility for NGN standardization
TC	Test Case
TCI	TTCN-3 Control Interface
TD	Test Description
TE	Test Equipment
TP	Test Purpose
TSS	Test Suite Structure
TTCN-3	Testing and Test Control Notation edition 3
UDP	User Datagram Protocol
UT	Upper Tester
UTP	Upper Tester Protocol

4 The TTCN-3 Framework

ETSI test specifications are usually developed for a single base protocol standard or for a coherent set of standards. As such, it is possible to follow the methodology specified for conformance test development in ISO/IEC 9646-1 [4] without much difficulty. However, the requirements of IPv6 are distributed across a wide range of documents and a different approach to test development is necessary. It is this approach that is referred to as the "TTCN-3 Framework".

As its name implies, the framework is oriented towards the production of Abstract Test Suites (ATS) in the Testing and Test Control Notation edition 3 (TTCN-3). The TTCN-3 Framework comprises:

- a documentation structure;
  - catalogue of requirements;
  - Test Suite Structure (TSS);
  - Test Purposes (TP);
    - conformance;
    - interoperability;
- Abstract Test Suite (ATS);
  - Test Cases (TC) in TTCN-3 for conformance tests;
  - Test Descriptions (TD) in tabulated English for interoperability tests;
- library of TTCN-3 building blocks;
  - data types and values;
  - templates;
  - general computational functions;
  - TP functions (see clause 7.2.1);
- a methodology linking the individual documentation, library and ATS elements together;
  - style guidelines and examples;
  - naming conventions;
  - guidelines on the use and extension of the TTCN-3 library;
  - a structured notation for TPs.

The TTCN-3 Framework, particularly the methodology, draws heavily on the tried and tested ISO/IEC 9646-1 [4] but modifies it to suit the particular case of IPv6 testing. It also incorporates guidelines on interoperability testing taken from TS 102 237-1 [1].

5 The IPv6 test development process

The process to be followed when developing IPv6 test specifications is shown in figure 1.

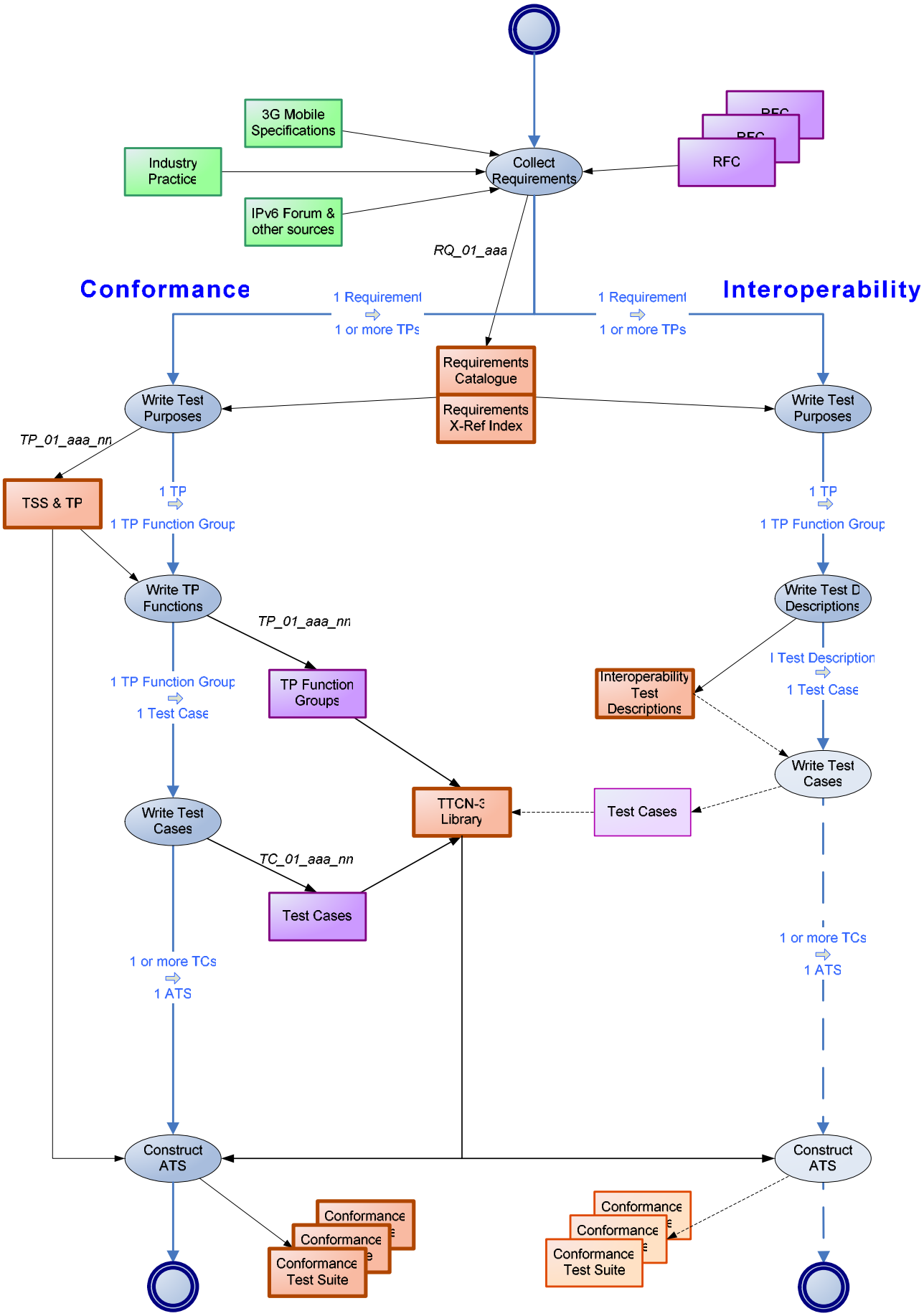


Figure 1: IPv6 test development process

The process begins with the analysis of the IETF RFCs related to IPv6 and a range of secondary inputs which include:

- current industry practice;
- existing test documentation from the IPv6 Forum and other established sources;
- specifications related to the use of IPv6 in 3rd Generation mobile networks.

The result of this analysis is the identification and classification of a full range of IPv6 requirements which is recorded in the Requirements Catalogue and used as the basis for both conformance and interoperability test specifications.

5.1 Conformance testing methodology

Any conformance test specification should be produced following the methodology described in ISO/IEC 9646-1 [4]. In summary, this methodology begins with the collation and categorization of the requirements to be tested into a tabular form which is normally referred to as the "Protocol Implementation Conformance Statement" (PICS). Each PICS relates to a specific protocol standard. As the requirements of IPv6 are distributed across a large number of documents, there would be very little benefit in producing a PICS for each document. Consequently, the IPv6 requirements will be collected together and categorized in a single document, the Requirements Catalogue.

For each requirement in the catalogue, one or more tests should be identified and classified into a number of groups which will provide a structure to the overall test suite (TSS). A brief Test Purpose (TP) should then be written for each identified test and this should make it clear what is to be tested but not how this should be done. Finally, a detailed Test Case (TC) is written for each TP. In the interests of test automation, TCs are usually combined into an Abstract Test Suite (ATS) using a specific testing language such as TTCN-3.

5.2 Interoperability testing methodology

For a certification (or branding or logo) scheme to be meaningful, it is necessary that interoperability testing is carried out in addition to conformance testing and that this is done in accordance with a comprehensive and structured suite of tests. In the context of the present document, it is exactly this type of testing which is referred to as "Interoperability Testing". The purpose of interoperability testing is to prove that the end-to-end functionality between (at least) two communicating systems is as required by the standard(s) on which those systems are based. A methodology for developing such interoperability test specification is described in TS 102 237-1 [1] and this should be used as a guide when developing IPv6 test suites. This methodology is based extensively on ISO/IEC 9646-1 [4] but with some modifications to make it more suitable for interoperability testing.

In TS 102 237-1 [1], the Interoperable Functions Statement (IFS) replaces the PICS and is a statement of which functions supported by the protocol have been implemented. However, in this framework these functions should be clearly identified in the Requirements Catalogue.

6 The Requirements Catalogue

The requirements which collectively specify and characterize IPv6 are taken from a wide range of specifications and other documentation. Building a coherent set of test specifications from these disperse requirements can be made simpler by gathering the requirements together into a single catalogue. The Requirements Catalogue lists IPv6 requirements from the various sources and organises them in a tree structure.

An example of the Requirements Catalogue in a tabulated form is shown in table 1 and table 2. The root of the catalogue consists of the base IPv6 functions/requirements (table 1), each of which can be specified further in sub-tables. The example in table 1 indicates that there is a requirement "provide something" which is elaborated in table 2.

Table 1: Provide IPv6 Services (example)

Requirement Group: Root of IPv6 Requirements		Source Reference	Requirement Type	IPv6 Label	Requirement Dependencies
1	Provide something	RFC XYZ, 1.7	MUST	MUST	None
2	[Provide something else] (table 2)			N/A	None
3	Limit the packet's something	RFC XYZ, 3.3	MAY	MUST	None
4	Provide the packet's something	RFC XYZ, 3.4	MUST	MUST	if R3 then MUST else N/A
5	Do something completely different	v6F 1234, 22.1	MUST	N/A	None
...	...	...	...	...	...

Table 2: Provide something (example)

Requirement Group: Provide something else		Source Reference	Requirement Type	IPv6 Label	Requirement Dependencies
32	Provide sub-something	RFC XYZ, 1.8	SHALL	MUST	None
33	Provide sub-something else	RFC XYZ, 2.3	SHOULD	N/A	None
34	Provide another sub-something else	RFC XYZ, 3.4	MAY	MUST	None
...	...	...	...	...	...

NOTE 1: This information may be presented in various forms, ideally as hyperlinked web pages. The intention of tables 1 and 2 is to show the extent of the information that needs to be collected.

NOTE 2: The grey columns indicate information that is supplied by a specific organization (the IPv6 Forum in this example) and is not supplied during the development of the test specifications as defined in this framework.

6.1 Entries in the Requirements Catalogue

For each requirement in the catalogue the following information should be present:

- requirement group which is either the root of the Requirements Catalogue or a sub group (another table) of requirements;
- a requirement identification number (clause 9.3). For easy use of the Requirements Catalogue, a sequential numbering system should be used in the table with a separate and complete cross-reference list linking the sequential numbers in the tables and the requirement identification numbers;
- the requirement in text:
  - a requirement in square brackets [] indicates a functional category created for structuring purposes. The category may not be specifically mentioned in any of the sources and is derived during development of the catalogue;
- reference to the source of the requirement (for example, an RFC number, or industry practice as may be defined in documentation specific to the IPv6 Forum or 3GPP). The reference should be precise and unambiguous, for example "RFC XYZ, 1.7" indicates that the reference text is from clause 1.7 of the RFC that is this requirement's source.
- the type of requirement (MUST, SHALL, SHOULD, MAY) which is useful in determining whether a requirement is optional or mandatory:
  - the keyword N/A is used to indicate that a requirement is not applicable;

- some requirements may be "negative" requirements, for example "... MUST not do something ...". In such cases the requirement type should be indicated as MUST_NOT (SHALL_NOT, SHOULD_NOT, MAY_NOT);
- for various reasons the language used in a particular standard may not always follow the relevant drafting rules and words such as "can/ought/will/could/etc." may be used to express a requirement. In such cases, the Requirement Type is assumed from the text and qualified with the word 'implied' in parentheses, for example MUST (implied);

- the organization requirement type. To be filled in by a particular organization which may wish to redefine the type of a requirement by, for example, stating a MAY requirement to be a MUST, or excluding a particular requirement. Typical cases of this would be the needs of the IPv6 Forum (v6Ready label requirement), 3GPP or ETSI TISPAN (NGN);
- the requirement dependencies. In many cases the implementation of one requirement will depend on the implementation of another. For example, a requirement of the kind "*... An IPv6 host MAY support <R3>. If it does then the host MUST also support <R4>...*". These dependencies are described as a Boolean expression (similar to those used in a PICS). In table 1 for R4 this is written as "***If** R3 **then** MUST **else** N/A*". These Boolean expressions are linked to Boolean module parameters that is used in the control part of the relevant TTCN-3 module to switch in or out individual test cases and/or groups of test cases. Note that changes to the Requirement Type as stated in the Organisation Requirement Type column may impact on the corresponding entries in the Dependencies column.

7 Developing test suites

7.1 Test Suite Structure (TSS) and Test Purposes (TP)

7.1.1 TSS

TSS groups should be chosen according to natural divisions in the base specification(s) and/or the architecture of the testing configuration. Examples might be, "Normal behaviour" and "Exceptional behaviour" or "Uni-cast operations" and "Multi-cast operations".

Interoperability test groups can be identified, for instance, according to the functionalities specified in the Requirements Catalogue.

7.1.2 TP Contents

A Test Purpose (TP) should be written for each potential test of an IPv6 requirement remembering that a requirement may need more than one TP to ensure that it is fully tested. As well as describing what is to be tested, the TP should identify the initial conditions to be established before testing can take place, the required status of the Implementation Under Test (IUT) or Equipment Under Test (EUT) from which testing can proceed and the criteria upon which verdicts can be assigned.

The contents of a TP should be limited to a description of what is to be tested rather than how that testing is to be carried out.

7.1.3 TP checklist

The TP checklist maps a specific requirement to one or more Test Purposes for that requirement. The numbering and naming conventions of clause 9 ensure consistency between requirements numbering and Test Purpose numbering. This checklist (table 3) serves as a useful summary as well as acting as a checklist for a particular IUT to indicate what requirements are supported. The IUT Support column is filled in at the time of testing (hence shown in grey).

Table 3: TP checklist

Organization: IPv6 Label		
Requirement	**TP**	**IUT Support**
1	TP1_1, TP1_2	Yes
2	TP2	No
3	TP3_1, TP3_2, TP3_3	Yes
...	...	...

7.1.4 Using the TP Language

There is considerable benefit to be gained by having all TPs written in a similar way. Readers of the TPs will find them easier to understand and harder to misinterpret. With this in mind, a simple, structured specification language has been developed for the expression of TPs. This is described fully in annex A. The use of this notation simplifies the identification of initial conditions, preamble, test description and postamble. Examples of the use of the language to express TPs are shown below.

EXAMPLE 1:

```
tp id TP_40147
title aligning PadN option
rc ref RC_61255
config ref CF_01
ensure that
    when {  IUT receives Echo Request from TN1
            containing Hop-by-Hop Options Header
            indicating Header Ext Length field ZERO
                and PadN option containing Opt Data Len field set to 4
                                and Option Data aligning the Hop-by-Hop Options Header
                                    to a multiple of 8 octets }
    then {  IUT sends Echo Request to TN2}
```

EXAMPLE 2:

```
tp id TP_40147
title not aligning PadN option
RC ref RC_61256
config ref CF_01
ensure that
    when {  RUT receives invalid Echo Request from TN1
            containing Hop-by-Hop Options Header
            indicating Header Ext Length field ZERO
                    and PadN option containing Opt Data Len field set to 3
                                        and Option Data not aligning the Hop-by-Hop
                                            Options Header to a multiple of 8 octets }
    then { RUT sends PARAMETER PROBLEM to TN1
                containing the Code field indicating code value 2
                    and the Pointer field indicating pointer value }
```

7.2 Test suite development

7.2.1 TP function groups

For each TP, a TP function group is specified in TTCN-3 and this should contain one function per test component. These functions should not be complete TCs with preambles and postambles but should contain only the TTCN-3 necessary to carry out the actions that the TP requires. A TP function should not set the test component verdict but should return a value which the calling TC can use to determine the verdict. Each of these TP functions will be entered into the project TTCN-3 library for future use in the development of TCs.

7.2.2 Test cases

A TC can be developed in TTCN-3 from the appropriate TP function(s) by adding:

- a preamble:
  - the actions required to place the IUT or EUT into the status required by the TP function;
- synchronization code:
  - the actions required to ensure that the main test component and any parallel test components are established in a known, coordinated status prior to the start of the test itself;
- the evaluation of a final verdict:

an assessment of the overall performance of the IUT or EUT based on information returned from TP Function calls, to determine whether it can be considered to have passed the test or not;

- a postamble:

  the actions required to return the IUT or EUT to a known quiescent state after completing the test.

7.2.3 Test case selection

The conformance test suite is a TTCN-3 module comprising all the test cases relevant to a particular area of IPv6, for example, Core IPv6. This set of test cases may be subset by the requirement needs of a particular organization (e.g. IPv6 Forum or 3GPP) as defined in the requirements catalogue for the organization.

In the control part of the test suite module each Test case should be preceded by a selection statement as shown in the following example:

```
If (RQ_01_407==true){
execute (f_TC_01_407_35_IPv6Router())
    }
```

In this example RQ_01_407 is a module parameter of type Boolean whose value is set by the entries in the IUT column of table 3 (Yes≡**true**, No≡**false**).

7.2.4 Test suite parameterization

It is often necessary to parameterise a test suite so that values not known at the time of writing the test cases can be used in testing. These values may depend on the IUT or the test system on which the test suite is being run.

NOTE: Test suite parameter values correspond to values normally found in a PICS or PIXIT.

Table 4 shows an example of how test suite parameters could be documented. The IUT Value column is completed at the time of testing.

Table 4: Module (test suite) parameters

Organization: IPv6 Label

Parameter Name	Description	Reference	Type	IUT Value
R_HOST	IP address for remote host	N/A	IPAddress	
T1	Response timer	RFC XYZ, 3.2	integer	
...				

7.3 Test description development

Test Descriptions (TDs) specify the detailed steps that must be followed in order to achieve the stated purpose of each interoperability test. These steps should be specified in a clear and unambiguous way but without placing unreasonable restrictions on how the step is performed. TDs written in a structured and tabulated natural language are ideal when the tests themselves are to be performed manually. If, however, tests are to be automated, test cases should be written in TTCN-3. The development of TTCN-3 test cases does not mean that TDs should not also be produced because they have significant value as higher-level designs of the test cases.

NOTE: TDs should only be used in the specification of interoperability tests and not for conformance tests.

8 The TTCN-3 library

In order to facilitate the rapid and consistent production of both abstract and executable test suites, a library of reusable TTCN-3 elements will be maintained. This library will be made freely available so that manufacturers, operators, testing organizations and other standards bodies can make use of it in constructing IPv6 test suites specific to their needs.

Once the library is established, facilities will be introduced for the controlled addition of new TTCN-3 elements and the modification of existing ones.

NOTE: The following clauses specify a number of rules as strong recommendations ("should") because they are considered to be based on good test programming practice. However, any TTCN-3 segments submitted for inclusion in the IPv6 TTCN-3 Library will be expected to comply with these recommendations as if they were mandatory.

8.1 Library structure overview

Although TTCN-3 does not mandate the use of any structure in a library, the elements will be grouped logically into a number of modules, thus:

- types and values which include:
  - data elements;
  - ports;
  - components;
  - module parameters;
  - templates;
- functions:
  - verdict handling functions;
  - synchronization functions;
  - test case functions;
  - TP functions;
  - other functions.

8.1.1 Data types and values

Commonly used subtypes for fields (e.g. subtypes for different encoding of integer, and octetstring values) are held within a single module. The TTCN-3 encode attribute is used here to provide additional information to codecs because the TCI [3] currently does not support access to subtyping information. This module also contains IPv6 library module parameter definitions.

```
type UInt2 integer (0..3) with { encode "2 bits" }
type Octet2 octetstring length(2) with { encode "2 octets" }
```

Types for data elements are organized in one module per RFC. A special role plays the root RFC which imports all other RFC modules. The latter also defines the union types for all IP headers and payloads. In general the type and value specification is modularized as follows:

- Common Library (CommonLib_TypesAndValues):

  Useful types which can be used in other projects;

  Example: CommonLib_TypesAndValues.UInt8

NOTE: Throughout the present document, examples indicate where relevant TTCN-3 code can be found in the electronic attachment in annex B

- IPv6 Library (Ipv6Lib_Common_TypesAndValues):

  Types that are used by more than one RFC are defined here;

  Example: Ipv6Lib_Common_TypesAndValues.MtuOption

- PIXIT parameters and Constants which are useful for multiple RFCs are grouped and defined here;

  Example: Ipv6Lib_Common_TypesAndValues.PX_LLA_ADDR_TN

- RFC 2460 [5] Root Library (Ipv6Lib_Rfc2460Root_TypesAndValues):

  models the IPv6 packet and therefore imports all the RFC type modules

- RFC-Specific Library (Rfc_Specific_TypesAndValues):

  - Each RFC module models a specific RFC;

EXAMPLE 1: Ipv6Lib_Rfc2461NeighborDiscovery_TypesAndValues

  - Each RFC module should only import the Common Library and the IPv6 Lib;
  - RFC specific PIXIT parameters and Constants should be grouped and defined here.

The rules used to define the types in each module are:

- If an RFC modifies an information element of another RFC then a separate type should be created in the RFC modules;

EXAMPLE 2: Ipv6Lib_Rfc2461NeighborDiscovery_TypesAndValues.PrefixInfo
and Ipv6Lib_RfcXXXXMipv6_TypesAndValues.MipPrefixInfo

- If an RFC module needs to use a type which is already defined in another RFC module then this type should be moved to the IPv6 Library;

EXAMPLE 3: Ipv6Lib_Common_TypesAndValues.SrcLinkLayerAddress used in
Ipv6Lib_Rfc2461NeighborDiscovery_TypesAndValues and
Ipv6Lib_RfcXXXXMipv6_TypesAndValues

- Any field of basic type in a user defined type should use the subtypes defined in the Common Library module;

EXAMPLE 4: Ipv6Lib_Rfc2894RouterRenumbering_TypesAndValues.RrMatchPrefix

- Field names of user defined types as well as type names should not be abbreviated but be written in full;
- List type identifiers should use the postfix "List". They should use length restrictions in their type definition, e.g. a lower bound of one list element;
- IPv6 packet structure is defined in RFCs using tables in which the encoding is specified. In some cases these tables allow a group of information elements to occur in arbitrary order. Here the following approaches should be taken in deriving their type structure:
- If this group consists only of information elements which can occur only once then a set type should be used to model that group. Elements which are not required in all packets should be reflected as optional set fields;

EXAMPLE 5: Ipv6Lib_Rfc2461NeighborDiscovery_TypesAndValues.RtAdvOptions

If some or all information elements in a group are able to occur more than once consecutively in a packet then a set type should also be used to model the group. List types should model set fields which can occur multiple times;

EXAMPLE 6: Ipv6Lib_Rfc2461NeighborDiscovery_TypesAndValues.PrefixInfoList

If the group allows some of its informational elements to occur more than once but in any order then a set of union type structure should be used. This type structure should also be used to model frequently extended groups such as IP headers and IP packet payload.

EXAMPLE 7: Ipv6Lib_Rfc2460Root_TypesAndValues.ExtensionHeaderList

Component and port types are specified in a separate module. The following rules apply to component types:

- There should be one general component type per protocol. This type is intended for use in the *runs on* statement of functions which define behaviour that can be used in any test component, irrespective of its role in the test case;

EXAMPLE 8: CommonLib_Synchronization.MtcComp

- There may be additional component definitions for specific roles. These should take the definitions of the general type as a basis and may extend them with additional port, timer, or variable definitions. These types should be used in the *runs on* statement of functions which define behaviour which can only make sense in the context of a specific role, e.g. in a test case function.

EXAMPLE 9: Ipv6Ats_Core_TestSystem.Master.

8.1.2 Templates

Conceptually, template definitions follow the same RFC based modularization as the types for data elements. The following rules apply to template specification:

- Templates should be identified with names rather than numbers;
- Templates should not modify other modified templates. Base templates which are modified must be identified in their naming;
- Templates should be specified separately for use in sending and receiving operations. Postfixes (clause 9) should be appended to clarify their use;

EXAMPLE: Ipv6Lib_Rfc2463Icmpv6_Templates.m_echoRequest_noData_snd
and Ipv6Lib_Rfc2463Icmpv6_Templates.m_echoReply_noData_rcv

- Template definitions should avoid using matching attributes such as "*" or "?" for complete structured values, e.g. record or set of values;
- PIXIT parameter values (table 4) should be passed as parameters into templates.

8.1.3 Test cases

Every test case should be selectable by having a test case selection function (see clause 7.2.3). This applies even to those test cases that test mandatory requirements from the base specification.

Where the test configuration involves more than one test component, the test case is coordinated by the MTC which:

- establishes the test configuration by creating, starting, mapping and connecting PTCs;
- starts the TC Function of each PTC;
- synchronizes the PTCs;
- shuts down the test configuration by unmapping and disconnecting PTCs.

If the test configuration involves only one test component, the test case is implemented by the MTC as a TC Function which also maps and unmaps required ports.

8.1.4 Functions

The IPv6 library differentiates between synchronization functions, verdict handling functions, TC functions, TP functions and other functions. Each type of function is implemented in a separate module, although there may be multiple modules for each function type. As an example, TP functions related to IPv6 core package could be implemented in a different module from the TP functions related to IPv6 security. The following general rules apply:

- Functions should use the *runs on* statement wherever this is possible;
- Each function should provide a return value. It is recommended to use the return value enumeration defined in the Common Library module;

EXAMPLE: CommonLib_TypesAndValues.FncRetCode

- If a selection switch is used then the associated *if* statement body should contain only a function call;
- The *stop* statement should be used with care in functions (controlled test component shutdown should be always insured).

8.1.4.1 Verdict handling functions

The following guidelines apply to functions which handle verdicts:

- Test verdict functions should only be used in the TC function or in the test case itself;
- Test verdict functions use the return value from a function to determine a test verdict.

8.1.4.2 Synchronization functions

The following guidelines apply to functions handling the synchronization of multiple, parallel test components:

- Synchronization should be invoked by the MTC at least after the preamble and before the postamble. The MTC may also invoke synchronization at other appropriate times;
- A PTC should synchronize after setting a verdict. This is to ensure that the verdict is always set prior to a PTC shutdown;
- Synchronization should use "named" synchronization as implemented in the ETSI TTCN-3 synchronization common library module:
  - Named synchronization uses a different synchronization message for each synchronization in order to avoid confusion where multiple synchronizations are required. The message is constructed from a synchronization string (chosen by the TTCN writer) concatenated with the string "`-READY`".

- Synchronization of test termination should use the stop message which is the character string "STOP";
- To terminate test execution a PTC should send the stop message to the MTC and wait for the corresponding STOP-notification from the MTC;
- If an MTC receives the stop message then it should send stop messages to all PTCs;
- To terminate test execution an MTC should send the stop message to all PTCs and wait for them to cease execution;
- If a PTC receives the stop message then it should execute the appropriate postamble. This could be implemented as default behaviour. As this notification may occur at any point of the PTC execution, the postamble should take its current state into account.

8.1.4.3 TC functions

The following guidelines apply to TC functions:

- TC Functions should only be necessary where there is more than one test component in the test architecture;
- Each PTC should have one TC Function defined for it;
- A TC Function is invoked in the "start test component" operation of a test case;
- TC Functions should be grouped with their associated test case;
- A TC Function should implement behaviour by invoking other functions rather than by expressing it directly. Any behaviour implemented directly in a TC Function would not be reusable in other test cases or functions;
- The name of a TC Function should include the role as well as the test case identifier as shown in the following example:
  - f_TC_01_051_12_Ipv6Host

8.1.4.4 TP functions

The following guidelines apply to TP functions:

- A TP Function should implement the test purpose for one component only;
- If there is more than one test component identified in the architecture associated with a TP, there should be one TP function for each of these components;
- If there is only one test component identified in the test architecture, there should be only one TP function for each TP;
- The name of a TP function should include the role as well as the test purpose identifier as shown in the following example:
  - f_TP_01_051_12_Ipv6Host();
- A TP function should not call other behavioural functions although computational functions can be called;
- TP functions should contain neither invocation nor implementation of test configuration management, preamble, or postamble aspects;
- A TP function should not set a verdict but use the return value to pass information with which the calling TC can determine a verdict. This allows the reuse of TP functions in preambles or postambles.

8.1.4.5 Other functions

Other function types should be collected into modules using grouping criteria appropriate to the particular application or project. Examples of such functions include test configuration management, preambles, postambles and algorithms.

The following guidelines apply to functions in this category:

- Other functions should not set verdicts but should use return values in the same way as TP functions;
- Other functions should never call the *stop* operation as this prevents execution of the postamble by TC functions;
- Other functions may call TP functions if they match the requirements of a preamble or postamble.

8.2 Adding modules to the library

Users or organizations may submit their own modules for addition to the ETSI IPv6 TTCN-3 module library. Such modules should be submitted to ETSI Technical Committee MTS for review. Details of the submission process can be obtained from the ETSI Secretariat at mtssupport@etsi.org.

9 Naming conventions

9.1 General guidelines

The IPv6 TTCN-3 library will be publicly available for test developers to use and, in a controlled way, extend. It is, therefore, desirable to specify a naming convention to cover each of the TTCN-3 elements which require an identifier.

The naming convention is based on the following underlying principles:

- when constructing meaningful identifiers, the general guidelines specified for naming in clause 6 of EG 202 106 [2] should be followed.
- in most cases, identifiers should be prefixed with a short alphabetic string (specified in table 5) indicating the type of TTCN-3 element it represents;
- suffixes should not be used except in those specific cases identified in table 5;
- prefixes and suffixes should be separated from the body of the identifier with an underscore ("_"):

EXAMPLES: `c_sixteen, t_wait_max`;

- only module names, data type names and module parameters should begin with an upper-case letter. All other names (i.e. the part of the identifier following the prefix) should begin with a lower-case letter:
- the start of second and subsequent words in an identifier should be indicated by capitalizing the first character. Underscores should not be used for this purpose:

EXAMPLE 2: `f_authenticateUser`;

Table 5 specifies the naming guidelines for each element of the TTCN-3 language indicating the recommended prefix, suffixes (if any) and capitalization.

Table 5: IPv6 TTCN-3 naming convention

Language element	Naming convention	Prefix	Suffix	Example	Notes
Module	Use upper-case initial letter	*none*	*none*	IPv6Templates	
TSS grouping	Use all upper-case letters as specified in clause 9.2	*none*	*none*	TP_RT_PS_TR	
Item group within a module	Use lower-case initial letter	*none*	*none*	messageGroup	
Data type	Use upper-case initial letter	*none*	*none*	SetupContents	
Data template	Use lower-case initial letter	m_	_snd _rcv	m_setupInit_snd m_setupBasic_rcv	Notes 1 and 2
Port instance	Use lower-case initial letter	*none*	*none*	signallingPort	
Test component ref	Use lower-case initial letter	*none*	*none*	userTerminal	
Signature	Use lower-case initial letter	s_	*none*	s_callSignature	
External function	Use lower-case initial letter	xf_	*none*	xf_calculateLength()	
Constant	Use lower-case initial letter	c_	*none*	c_maxRetransmission	
Function	Use lower-case initial letter	f_	*none*	f_authentication()	Note 6
Altstep	Use lower-case initial letter	a_	*none*	a_receiveSetup()	
Altstep (Default)	Use lower-case initial letter	d_	*none*	d_receiveOtherMessages()	
Test case	Use numbering as specified in clause 9.4	TC_	*none*	TC01_009_47	
Variable	Use lower-case initial letter	v_	_gbl	v_macId v_systemName_gbl	Note 3
Timer	Use lower-case initial letter	t_	_min _max	t_wait t_auth_min	Note 4
Module parameter	Use all upper case letters	*none*	*none*	PX_MAC_ID	Note 5
External constant	Use lower-case initial letter	xc_	*none*	xc_macId	
Parameterization	Use lower-case initial letter	p_	*none*	p_macId	
Enumerated Value	Use lower-case initial letter	e_	*none*	e_synCpk	

NOTE 1: If different templates based on the same types are introduced, the name part (not prefix or suffix) of each identifier should give further information about the template's purpose (e.g. m_setupInit_snd, m_setupBcapFax_snd).
NOTE 2: If no suffix is used, the template is considered to be bidirectional.
NOTE 3: Local variables have no suffix but if global variables are used, the suffix "_gbl" should be appended.
NOTE 4: If a time window is needed, the suffixes "_min" and "_max" should be appended.
NOTE 5: In this case it is acceptable to use underscore as a word delimiter.
NOTE 6: The naming of TP functions follows the convention described in clause 9.4 for TPs

9.2 Naming IPv6 test groups

TP groups have a short name (or identifier) and a longer, more readable title. The short name is derived from the longer title (i.e. it is a two or three letter abbreviation of the longer title name). For example, if the long title is "Router", the short name could be: "RT". It is recommended that the title is followed by the short name in parentheses, for example: "Router (RT)"In the case of subgroups both the title and the short name should reflect the sub structuring, essentially making them path names. The group delimiter in the case of the title is "/". The delimiter in the case of the short name is: "_". As a further example, the group "Provide IPv6 Services (PS)" which is a sub group of the "Router (RT)" group, has the title:

```
Router(TR)/Provide IPv6 Services(PS)
```

and the short name:

```
RT_PS
```

9.3 Naming IPv6 requirements

Although individual requirements will not need to be identified in the TTCN-3 code, it will still be necessary to provide a unique name for each requirement in the catalogue. Each requirement name will begin with "RQ_" followed by two digits indicating which area of the IPv6 specification it refers to and a three-digit identifier, as follows:

- RQ_01_nnn IPv6 Core requirements (example: RQ_01_254)
- RQ_02_nnn IPv6 Security requirements (example: RQ_02_037)
- RQ_03_nnn IPv6 Mobility requirements (example: RQ_03_198)
- RQ_04_nnn IPv4 to IPv6 Transitioning requirements (example: RQ_04_471)

9.4 Naming IPv6 TPs and TCs

As there will be a one-to-one relationship between TPs and TCs, they will share a common numbering scheme with a prefix to distinguish between them. The prefixes will, naturally, be "TP" for test purposes and "TC" for test cases which will be followed by the five-digit sequence number taken from the requirement it corresponds to, a two digit sequence number and finally a character string indicating the architectural role with which it is associated, thus:

- TP_01_nnn_mm_aaaa/TC_01_nnn_mm_aaaa IPv6 Core TPs and TCs

EXAMPLE 1: TP_01_147_04_IPv6Host, TC_01_147_04_IPv6Host

- TP_02_nnn_mm_aaaa/TC_02_nnn_mm_aaaa IPv6 Security TPs and TCs

EXAMPLE 2: TP_02_109_17_IPv6Router, TC_02_109_17_IPv6Router

- TP_03_nnn_mm_aaaa /TC_02_nnn_mm_aaaa IPv6 Mobility TPs and TCs

EXAMPLE 3: TP_03_033_05_IPv6Terminal, TC_03_033_05_IPv6Terminal

- TP_04_nnn_mm_aaaa /TC_04_nnn_mm_aaaa IPv4 to IPv6 Transitioning TPs and TCs

EXAMPLE 4: TP_04_006_32_IPv6Server, TC_04_006_32_IPv6Server)

10 Specifying an upper tester

In order to completely automate conformance and interoperability testing, the upper interface or API of the IUT needs to be accessible to TTCN-3 test cases. The specification of this upper interface is not standardized by IPv6 RFCs and so there are no primitives defined for requesting the IPv6 stack to send a specific IP packet or to check if one has been received. Consequently, implementations of this interface are vendor specific and may even vary between different IUTs.

EXAMPLE: it may based on primitives or the socket API and often requires a tight integration with the IUT.

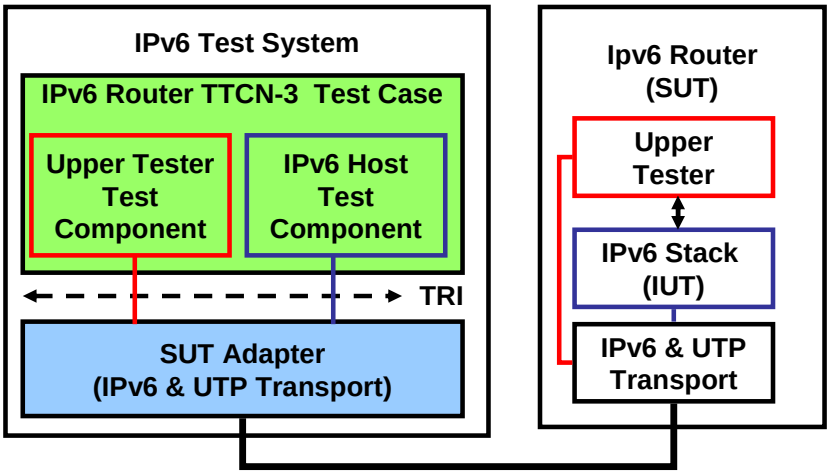


Figure 2: An example test configuration with an upper tester

In conformance testing methodology the tight integration problem can be resolved by implementing an Upper Tester (UT) in the SUT, i.e. outside of the test system. The purpose of the UT is to play the role of a (dummy) IPv6 application which interacts with the IPv6 stack. It is, however, controlled by the test system via a message channel. Therefore, another task of the UT is to convert the messages sent by the test system into concrete IPv6 interface calls and vice versa. This allows a fairly generic design and encoding of a protocol between the UT and the test system.

A UT may be implemented in the concrete implementation language used by the IUT which allows an easy integration of the UT with the IUT. As the UT implementation is clearly SUT specific it is not provided as part of the TTCN-3 IPv6 test system. It is expected to be provided by the party which intends to use the test suite.

10.1 The UT in the IPv6 test system

In the test system the UT is represented in each test case by its own PTC. During the execution of a test case this PTC issues commands to interact with the upper tester in the SUT using messages or procedure calls. Although the Ipv6 test system does not mandate how a UT implements such an API invocation, it requires the UT to support the IPv6 UT protocol.

The IPv6 UT protocol is used by IPv6 library test cases to communicate with the UT. It defines primitives which, for example, indicate the start and end of a test case, reset the UT in case of test case errors and send or indicate the reception of an IPv6 packet. In order to be as independent of the upper IPv6 interface as possible, the protocol leaves IPv6 packet related information in encoded format.

The TTCN-3 type definitions for the Ipv6 UT protocol and the encoding of its primitives are discussed in more detail in. IPv6 UT protocol messages are to be transported using UDP/Ipv4. The UDP port to be used for communication should be 5080.

Annex A (informative): A formal notation for expressing test purposes

A.1 Introduction

A simple but formal notation has been developed for the expression of TPs in a consistent and structured form. This notation provides structure through the use of defined keywords (see table 6) but also allows the TP writer considerable freedom in the use of text between the keywords.

The notation allows the grouping of TPs (to provide the TSS). It provides header information for each TP and a description of the TP. Line comments may be expressed using "`//`". Comments that cover more than one line should be enclosed by "`/*`" and "`*/`".

Table 6: TP notation keywords

TP grouping keywords	TP Body keywords
description	**accepts**
group	**after**
id	**and**
title	**before**
	containing
TP header keywords	**ensure**
rc	**ignores**
ref	**indicating**
title	**not**
tp	**receives**
id	**rejects**
	remains
	sends
	that
	then
	to
	when
	with

A.2 Grouping

The TSS (Test Suite Structure) is expressed using the **group** keyword. Groups may be nested to provide sub-grouping. The body of the group (i.e. subsequent groups or TPs) should be enclosed in curly braces, i.e. **{** ... **}**.

Each group should have:

- an identifier (**group id**) as described in clause 9.2;
- a long form of the identifier (**title**) as described in clause 9.2;
- a short free text description of the test group (**description)**.

Indentation may be used to indicate a sub group. But in cases of deep sub-grouping this should be avoided for readability reasons.

In order to aid readability it is recommended that the end of the end of the group is followed by a comment that shows the group name (identifier).

Example of one group and a sub group:

```
:
group id TP_RT
title Router
description Test Purposes for Router
{   group id TP_RT_PS
    title Router(RT)/Provide IPv6 Services(PS)
    description Test Purposes for Provide IPv6 Services
    { ... TPs or more subgroups can go here ...
    } // end TP_RT_PS
} // end TP_RT
:
```

A.3 TP Header

Each TP should begin with a header which contains a number of items of descriptive information about the TP with each item introduced by a defined keyword but written in free text. The elements that should be included in the header are:

- the TP Identifier (**tp id**):

  the unique identifier of the TP as described in clause 9.4;

- the TP title (**title**):

  free text descriptive title of the TP;

- a reference to the Requirements Catalogue (**rc ref**) :

  identification of the relevant text in the base standard(s) where the requirement to be tested is specified;

- a reference to the testing configuration (**config ref**):

  identification of the predefined testing architecture which is applicable to the test.

Example TP header:

```
:
tp id TP_01_147_04_IPv6Host
title Pad1 option
rc ref Item_0 and Item_1
config ref Config_RUT_2
:
```

A.4 TP body

The main body of the TP follows the header and it is here that the test itself is described. The TP is written from the viewpoint of the IUT. The general form of a TP is as follows:

```
ensure that {       // start of TP body
    with { ... }    // initial conditions
    when { ... }    // tester activities
    then { ... }    // iut responses and verdict criteria
    }               // end of TP body
```

The **when** and **then** statements may be repeated, for example:

```
ensure that {
    with { the iut in some initial state or condition }
    when { tester does action 1 }
    then { iut does response 1 }
    when { tester does action 2 }
    then { iut does response 2 and verdict criteria }
    }
```

A.4.1 The **with** statement

The **with** statement expresses the initial state or condition of the IUT at which point the TP description begins. Note, this does not define the steps, or actions, needed to reach this starting condition, only the condition itself.

Apart from free text, typical keywords that may appear in the condition are **and**, **or**, **not**. For example:

```
with { the IUT in idle state and port80 open }
```

A.4.2 The **when** statement

The **when** statement expresses some action, in most cases performed by the tester and observed by the IUT. Typically this will be a **receives** statement (i.e. the IUT has received some stimulus) with a description of the IPv6 header and relevant fields (**containing**, **indicating**). Other typical keywords that may appear in the **when** statement are **and**, **or**, **not**.

For example:

```
when {   IUT receives    Echo Request
         containing      Hop-by-Hop Options Header
         indicating      Header Ext Length field ZERO
                 and         a PadN option
                 containing  the Opt Data Len field set to 4
     }
```

In cases where there is more than one test interface in the test configuration the keyword **from** can be added to the **receives**. For example:

```
IUT receives Echo Request from TestNode1
```

A.4.3 The **then** statement

The **then** statement expresses some response (usually by the IUT) to the **when** statement. For example,

```
then { IUT sends the Echo Request }
```

In cases where there is more than one test interface in the test configuration the keyword **to** can be added to the **sends**. For example:

```
then { RUT sends the Echo Request to TestNode2}
```

The keywords **accepts**, **ignores**, **rejects** are other possible response to a received message. For example:

```
then { IUT rejects Echo Request }
```

The **remains** keyword can be used to express that the IUT does not change state or condition. For example:

```
then { IUT remains in the idle state }
```

Apart from free text, the keywords **and**, **or**, **not** may be used in a **then** statement. For example, the following is equivalent to the two **then** statements above:

```
then {  IUT rejects Echo Request

        and remains in the idle state }
```

Finally, the keywords **before** and **after** can be used to express ordering, especially in the context of timers. For example:

```
before T1 expires

after 15 seconds
```

Annex B (informative): Example TTCN-3 library modules

B.1 Electronic annex, zip file with TTCN-3 code -

TTCN-3 library mdoules are contained in archive ts_102351v010101p0.zip which accompanies the present document. They can be used in a TTCN-3 editor as examples.

Annex C (informative): TTCN-3 type definitions and encoding for a UT protocol

The IPv6 UT protocol (UTP) primitives can be grouped into generic primitives as well as Ipv6 specific primitives. The following UTP messages are defined in the IPv6 test suite which can be found in the electronic attachment to annex B:

- UTP specific primitives:
  - IPv6 Request (UtpIpv6Request);
  - IPv6 Response (UtpIpv6Response);
- Generic primitives:
  - Start Test Case request (UtpStartTCRequest);
  - UtpEndTCRequest
  - UtpResetRequest
  - UtpGenericResponse.

In IPv6 specific UTP messages, the application id field allows the test system to interact with multiple UTs or applications with one UT. The use of these identifiers should start from 0. The command used in this message should reflect actions to be taken from the perspective of a test case. In a successful case the return code in a response should provide information about the state of an application in the UT.

In the UTP the test system always acts as a client whereas the UT acts as a server. Therefore, UT protocol requests can only be sent by the test system. UTP also requires that the test system polls the UT for received messages. It does so using the "getPkt" command. The UT must respond to all generic request messages with "UtpGenericResponse" messages. The UtpIp6Request must be answered by a "UtpIpv6Response" message.

The encoding of UT protocol messages has been designed to be as simple and humanly readable as possible in order to simplify the analysis of communication between the test system and the UT. An encoded UT protocol message can be thought of as a concatenation of various text as well as octet strings. Most of the strings in a UT message are TTCN-3 values. TTCN-3 enumeration values are to be encoded using their name (without the "e_" prefix), i.e. not their integer value. Finally, user defined types and their fields are encoded using the prefix string specified for a type or type field in the UT protocol type definition.

Each string (no matter if it is a prefix for a TTCN-3 field, its string value, or an enumeration value) is encoded with one octet specifying the length of the following string and then the string value in its native format, i.e. either an octet or text string. Such an encoding of strings is already used for strings by other IETF protocols, e.g. the DNS protocol [6] clause 3.3.

EXAMPLE: TTCN-3 message (type see type definition above):

```
template UtpStartTCRequest m_utpStartTC_001 := {
    testCaseId := 'TC_001'
}
```

Corresponding encoding (length octets shown in \xhh format):

```
\x12Utp/1.0/StartTcReq\x06TC_001
```

History

Document history		
V1.1.1	September 2004	Publication