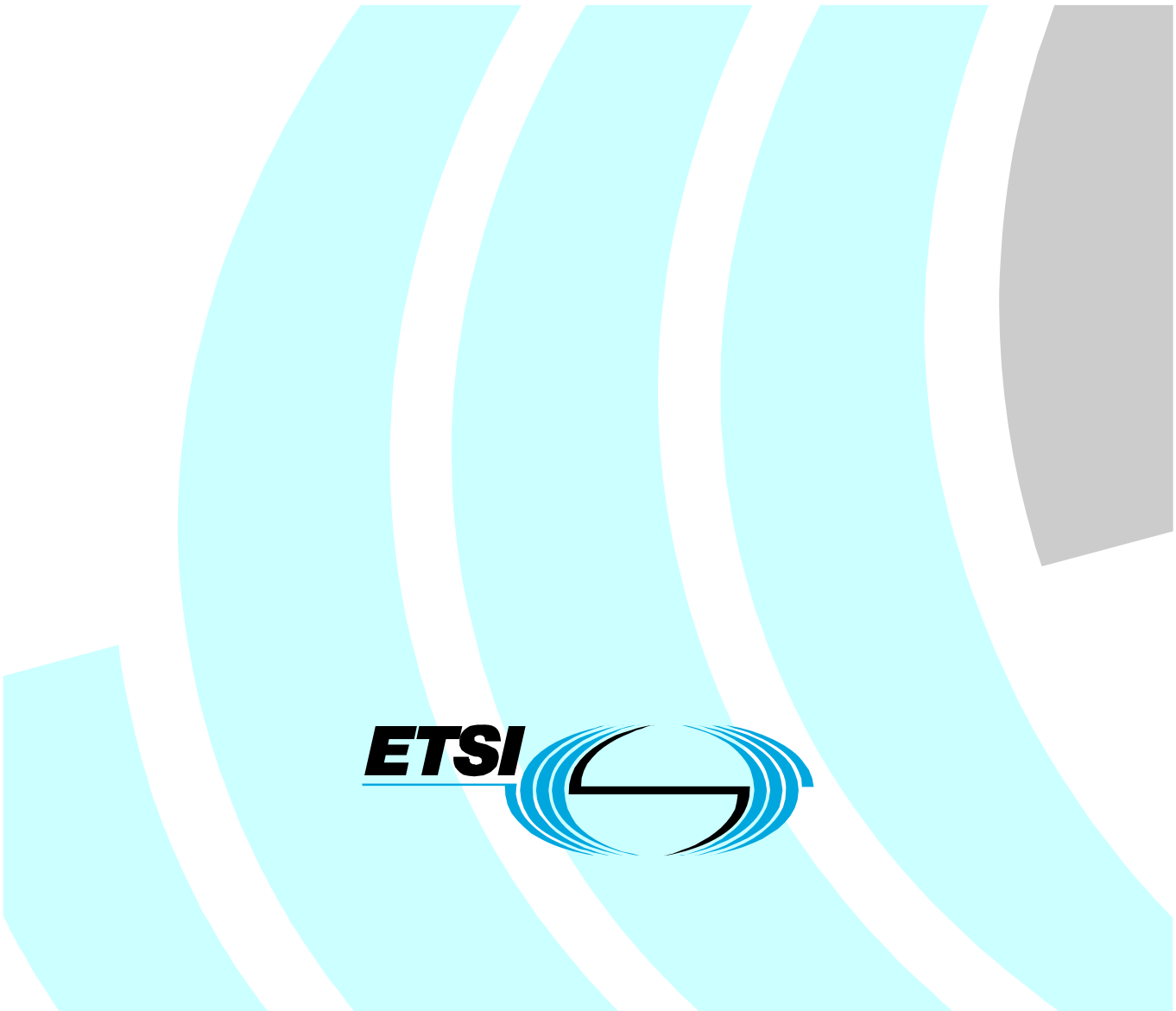
ETSI ES 201 873-10 V3.4.1 (2008-09)

ETSI Standard

Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
Part 10: TTCN-3 Documentation Comment Specification



Reference
RES/MTS-00108-10 T3 ed341 DOC

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

The present document is part 10 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

1 Scope

The present document defines a documentation of TTCN-3 source code using special documentation comments. The source code documentation can then be produced automatically from the TTCN-3 Core Language, e.g. in the form of hypertext web pages.

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.

For online referenced documents, information sufficient to identify and locate the source shall be provided. Preferably, the primary source of the referenced document should be cited, in order to ensure traceability. Furthermore, the reference should, as far as possible, remain valid for the expected life of the document. The reference shall include the method of access to the referenced document and the full network address, with the same punctuation and use of upper case and lower case letters.

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 (V3.2.1): "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[2] W3C Recommendation (24 December 1999): "HTML 4.01 Specification".

[3] IETF RFC 4248 (October 2005): "The telnet URI Scheme".

[4] IETF RFC 2616 (June 1999): "Hypertext Transfer Protocol -- HTTP/1.1".

[5] IETF RFC 2660 (August 1999): "The Secure HyperText Transfer Protocol".

[6] IETF RFC 2818 (May 2000): "HTTP Over TLS".

[7] IETF RFC 3986 (January 2005): "Uniform Resource Identifier (URI): Generic Syntax".

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 terms and definitions given in ES 201 873-1 [1] and the following apply:

directly follows, directly precedes, before, follows, precedes: these terms are used to describe the textual order between a documentation block and a TTCN-3 language element and, in this respect, empty spaces, empty lines and simple (non-documentation) TTCN-3 comments will not be considered

documentation block: documentation block comprises all information included into the TTCN-3 source code and which relates to the documentation of a single given TTCN-3 entity

ignore: ignoring means that the information ignored (text, tagged paragraph, a whole documentation block, etc.) will not contribute to any generated output

source code, TTCN-3 source code: content of one or more TTCN-3 modules in TTCN-3 core language as defined in ES 201 873-1 [1]

tagged paragraph: one or more lines of a documentation block starting with a tag specified in clause 6 of ES 201 873-10

NOTE: Tagged paragraphs end with the next tagged paragraph or by the end of the documentation block.

3.2 Abbreviations

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

HTML	Hypertext Markup Language
RFC	Request For Comments
TTCN-3	Testing and Test Control Notation version 3

4 Introduction

TTCN-3 comments (see clause A.1.4 of ES 201 873-1 [1]) can also be used to include specific information for the purpose of documenting the TTCN-3 source code itself. This part of the TTCN-3 standard defines the syntax and the use of such documentation-purpose comments.

4.1 Conformance

For an implementation claiming to conform to the present document, all features (e.g. documentation block formats, tags and error handling procedures) specified in the present document shall be implemented consistently with the requirements given in the present document.

4.2 Syntactical conventions

Descriptions of the syntactical structures in the present document use the following convention: tag names (which includes the `@` character) shall appear in the documentation block as they given in the syntactical rule. The `freetext` refers to a text complying with the rules for TTCN-3 comments of the core language specification (ES 201 873-1 [1]), the internal structure of which is not defined by the present document with one exception: if the commercial at ("@") character necessary to be included into the generated documentation text, in `freetext` a pair of commercial at characters ("@@") shall be used on the same line with no intervening space characters. Other terminals (e.g. Identifier) are start with a capital character. It is recommended, that names used as Identifiers are checked by TTCN-3 documentation tools.

5 Documentation blocks

Documentation blocks can appear before:

a) TTCN-3 modules;

b) Module control parts;

c) Groups;

d) Definitions of global visibility (see ES 201 873-1 [1]);

e) Fields of structured type-, module parameter-, constant- and template definitions;

f) Enumerations of enumerated type definitions;

g) Signature- and formal parameters;

h) Return clauses of function and signature definitions; and

i) Exception clauses of signature declarations.

Documentation comments shall be valid TTCN-3 comments, conforming to the TTCN-3 core language standard (ES 201 873-1 [1]).

NOTE: Tools may also extract other information from the TTCN-3 code based, for example, on TTCN- 3 keywords or headers of TTCN-3 definitions. The description of that extraction is beyond the scope of the present document.

5.1 Identifying documentation blocks

Both block comments and line comments may be used for documentation purposes (see clause A.1.4 of ES 201 873-1 [1]). Documentation comments shall be valid TTCN-3 comments identified by an asterisk (`*`) character directly following the opening symbol pairs of the TTCN-3 comment (i.e. without a space or any other character between the comment symbol pair and the asterisk).

When using block comments for source code documentation, one TTCN-3 block comment constitutes one documentation block.

When using line comments for source code documentation, the documentation block may consist of several subsequent line comments, each identified by the asterisk (`*`) character following the comment symbol (`//`). Empty lines between subsequent documentation line comments do not influence the documentation block (i.e. shall be ignored).

Examples

EXAMPLE 1:

```
* text for documentation
  using block comment */
```

EXAMPLE 2:

```
* text for documentation
*    using line comments
```

5.2 The Body

The body of a documentation comment consists of the characters:

- Between the `/**` that begins the comment and the `*/` that ends it when TTCN-3 block comments are used for documentation.
- Following the documentation comment symbol(s) in each line when TTCN-3 line comments are used (i.e. characters between the `//*` symbol and the end of the line, with the exceptions given below).

The body of a documentation block may be divided into one or more lines. On each of these lines, leading and closing `*` characters not belonging to the documentation comment symbols shall be ignored. For lines other than the first, whitespaces (see [1] clause B.1.5) preceding the initial and closing asterisk (`*`) and following the closing asterisk (*) characters shall also be discarded.

Examples

So, for example:

```
/************************************
  ** @desc XYZ                      **
  ** Initialize to pre-trial defaults.**
  **                                **
  123                               **
  ************************************/
```

This TTCN-3 comment has six lines. The /** in the first line makes it a documentation block. The rest of the line is ignored as it contains leading asterisk (*) only. The second line consists of the documentation text `"@desc XYZ"`; the third line consists of the text `" Initialize to pre-trial defaults."`, the text in the fourth line is empty and the fifth line consists of the text `"  123"`. The * characters in the last line are ignored again as leading asterisks (the */ closes the TTCN-3 comment).

The example below will result in the same documentation text:

```
//* @desc XYZ
//*  ** Initialize to pre-trial defaults.**
//*  **                                **
//*  123                               **
```

5.3 Formatting

The text of documentation blocks not part of any tagged paragraph and the `freetext` part of tagged paragraphs (see clause 6) may use special markers to indicate the documentation tool that a text fragment shall be formatted in the way defined by the markers.

The following HTML elements shall be supported:

- Paragraph `<P>` and `</P>`;
- Forced line break `<BR>` and `</BR>`;
- Italic text style `<I>` and `</I>`;
- Emphasis `<EM>` and `</EM>`;
- Computer code fragment `<CODE>` and `</CODE>`;
- Pre-formatted text `<PRE>` and `</PRE>`;
- Unordered list `<UL>` and `</UL>`;

- Ordered list <OL> and </OL>;
- List item <LI> and </LI>.

The complete description of HTML is defined in [2].

NOTE: In addition, tools may support other HTML elements.

6 Tagged paragraphs

Tagged paragraphs can be used in documentation blocks preceding TTCN-3 modules, control parts, groups and definitions of global visibility. In documentation blocks used before these language elements only tagged paragraphs shall contribute to the generated documentation.

Tagged paragraphs identify certain types of information that has a specified structure, such as the description of a function or the meaning of a formal parameter. Thus, the documentation comment processor can e.g. easily marshal the information into standard typographical formats for purposes of presentation and cross-reference.

Tagged information cannot be embedded into other tagged paragraphs unless specified differently in the subsequent clauses.

Each tagged paragraph defines its own multiplicity. The order of tagged paragraphs should not be changed by documentation tools when producing TTCN-3 source code documentation.

NOTE: In addition, tools may support other, non standardized tags.

Whitespaces and newline characters (see [1] clause B.1.5) between a tag and the field following it shall be ignored.

EXAMPLE 1: Ignoring whitespace and newline directly following a tag

```
//* @author Alan Alexander Milne, Mary Wollstonecraft Shelley
module Mymodule {
// module content
}
// Shall produce exactly the same documentation output as
//* @author                      *
//* Alan Alexander Milne,        *
//* Mary Wollstonecraft Shelley  *
module Mymodule {
// module content
}
```

6.1 The @author Tag

This tag shall be used to specify the names of the authors or an authoring organization related to the TTCN-3 object being documented.

Syntactical Structure

```
@author [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag documents the language element it precedes.

A documentation block may contain more than one @author tag. It is recommended to specify one author per @author paragraph, which allows the documentation processing tool to provide the correct punctuation in all circumstances.

Examples

EXAMPLE 1: Using the @ author tag for listing names of authors.

```
//* @author Mary Wollstonecraft
//* @author Hildegard von Bingen
type record MyMessage {
// definition content
}
```

EXAMPLE 2: Using the @ author tag for listing names of organization.

```
//* @author Test Competence Center
module SCTP_Types {
// module content
}
```

EXAMPLE 3: A single @author paragraph may mention several authors.

```
//* @author Jack Kent, Peggy Parish, Crockett Johnson,
//*         A.A. Milne, Marjorie Weinman Sharmat,
//*         Mary Shelley, and Madeleine L'Engle
module Mymodule {
// module content
}
```

6.2 The `@config` Tag

This tag shall be used to specify the test configuration used by the documented TTCN-3 test case.

Syntactical Structure

```
@config [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag shall be used in documentation blocks preceding TTCN-3 testcase definitions only and it shall refer to the configuration of the test case.

The information in this tag shall contain the name or a reference of the test configuration used by the given test case.

A documentation block may contain at most one `@config` tag.

NOTE: Though this tag is used in documentation blocks preceding testcase definitions, the scope of this tag is not limited to the MTC only but should relate to all PTCs and the TSI as well.

Example

```
//* @config Configuration 1 from clause 4.1 of the ATS document.
testcase tc_MytestCase () runs on MTCCompType system SystemCompType {
// test case body
}
```

6.3 The `@desc` Tag

This tag shall be used to describe the intention and functionality of a particular TTCN-3 object.

NOTE: Note that the purpose of TTCN-3 test cases are documented using the `@purpose` tag.

Syntactical Structure

```
@desc [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag documents the language element it precedes.

A documentation block may contain more than one `@desc` tag.

Examples

EXAMPLE 1: Using the @desc tag for structured types.

```
//* @desc This type defines what a test packet consists of.
type record TestPackage {
//content of the type definition
}
```

EXAMPLE 2: Using the @desc tag for modulepars.

```
//* @desc SUT port number used for interface 1.
modulepar integer SUTport1 := 8888,
//* @desc SUT port number used for interface 2;
                  SUTport2 := 9999;
```

EXAMPLE 3: Using the @desc tag for constants.

```
//* @desc Maximum duration a message will remain in the network.
const float MaxDuration := 5.0;
```

6.4 The `@exception` Tag

This tag shall be used to provide information on the exception types of signature definitions.

Syntactical Structure

```
@exception Identifier [freetext]
```

The `@exception` tag shall be followed by the name of an exception type specified by the signature the documentation block belongs to. The exception type name may optionally be followed by a text, the internal structure of which is not specified by the present document.

Semantic Description

This tag shall be used in documentation blocks of signatures only and it shall document the language element identified by the `Identifier`.

The `Identifier` shall be a type name listed as exception type in signature definition being documented by the documentation block.

A documentation block may contain more than one `@exception` tag and each tag shall document exactly one signature exception type.

NOTE 1: It is recommended that the descriptive text part following the exception name contains a short description of the circumstances that cause the exception(s) of the given type to be returned.

NOTE 2: Using this tag as implicitly tagged documentation information is specified in clause 7.

Example

```
// Using the @exception tag
//* @exception IndexOutOfBoundsExceptionType The index passed is too large
//* @exception FileNotFoundExceptionType
//*            value1 - the file with the specified name has not been found
//*            value2 - the specified path does not exists
signature FindXthElementInFile
  ( in charstring pathAndFileName, in integer findElement)
  return ListElementType
  exception (IndexOutOfBoundsExceptionType, FileNotFoundExceptionType);
```

6.5 The @member Tag

This tag shall be used to document the members of TTCN-3 objects.

Syntactical Structure

```
@member Identifier [freetext]
```

The @member tag shall be followed by a name of a member specified by the TTCN-3 object the documentation block belongs to. The member name may optionally be followed by a text, the internal structure of which is not specified by the present document. Field names of nested records, set and union type definitions shall use the dot notation.

Semantic Description

This tag shall only be used in documentation blocks preceding record, set, union, enumerated, port and component type definitions and record, set and union constant-, module parameter-and template definitions and it shall document the language element identified by the Identifier.

In the case of record, set and union types, module parameters, constants and templates the Identifier shall be a valid field name. In the case of enumerated types the Identifier shall be a valid enumeration value name. In the case of component types the Identifier shall be a valid constant, variable, timer or port name defined by the type. In the case of port types it shall be a valid data type or signature name referenced by the port type.

A documentation block may contain more than one @member tag and each tag shall document exactly one member.

NOTE 1: It is recommended that the order of the @member paragraphs within a documentation comment follows the order in which the members are declared or referenced.

NOTE 2: Using this tag as implicitly tagged documentation information inside definitions is specified in clause 7.

Examples

EXAMPLE 1: Using the @member tag for structured types.

```
//* @member extensionHeaderFields List of extension headers defined by RFC 2460.
//* @member extensionHeaderFields.header Headers specified in RFC 2460.
//* @member extensionHeaderFields.unknown Extra header field for headers not specified in RFC 2460.
type record IP6extensionHeader {
  record of record {
    ExtensionHeaders header,
    octetstring      unknown
  }  extensionHeaderFields
}
```

EXAMPLE 2: Using the @member tag for enumerated types.

```
//* @member ackProhibited If this non default value is taken, the SS has prohibited
//*         acknowledging RLC SDUs.
//* @member ackContinued If this non default value is taken, the SS has continued
//*         acknowledging RLC SDUs.
//* @member noNeed This default value indicates that the SS does not perform the
//*         operation mentioned in CRLC_ProhibitRLC_Ack_REQ, but performs the
//*         suspension / resume of UL RLC PDU data.
type enumerated SupportFlag { ackProhibited (0), ackContinued (1), noNeed (2) }
```

EXAMPLE 3: Using the @member tag for port types.

```
//* @member IPv6Message The complete IPv6 packet (not encoded)
//* @member octetstring When an IPv6 packet decoding fails in the TA, it will pass the
//*         received packet to the TE as octetstring.
type port IPv6PortType
  message {
    inout_IPv6Message;
    in octetstring
  }
```

EXAMPLE 4: Using the @member tag for component types.

```
//* @member v_IPv6Message Variable to store incoming IPv6 packets
//* @member T_Guard Generic send (no-response) guard timer
//* @member IPv6Port definition required for LibIpv6COmp type compatibility.
type component MyPTCType {
  var IPv6Message v_IPv6Message;
  timer T_Guard := 10.0;
  port IPv6PortType IPv6Port
}
```

EXAMPLE 5: Using the @member tag for module parameters.

```
//* @member PX_ETS_DISP_NAME Display name used by the ETS to exchange SIP messages on MTC side.
//* @member PX_ETS_IPADDR IP address used by the ETS to exchange SIP messages on MTC side.
modulepar charstring PX_ETS_DISP_NAME := "TESTER1", PX_ETS_IPADDR := "127.0.0.1";
//* @member PX_ETS_PORT Port number used by the ETS to exchange SIP messages on MTC side.
modulepar integer PX_ETS_PORT := 5061;

// Please note, modulepar syntax used in this example is deprecated and will be removed in
// the next edition of the core language (see Annex F of [1]).
//* @member PX_ETS_DISP_NAME Display name used by the ETS to exchange SIP messages on MTC side.
//* @member PX_ETS_IPADDR IP address used by the ETS to exchange SIP messages on MTC side.
//* @member PX_ETS_PORT Port number used by the ETS to exchange SIP messages on MTC side.
modulepar {
  charstring PX_ETS_DISP_NAME := "TESTER1", PX_ETS_IPADDR := "127.0.0.1";
  integer PX_ETS_PORT := 5061
}
```

6.6 The `@param` Tag

This tag shall be used to document the formal parameters of parameterized TTCN-3 definitions and parameters of signatures.

Syntactical Structure

```
@param Identifier [freetext]
```

The `@param` tag shall be followed by a name of a formal parameter defined by the TTCN-3 object the documentation block belongs to. The parameter name may optionally be followed by a text, the internal structure of which is not specified by the present document.

Semantic Description

This tag shall only be used in documentation blocks preceding parameterized TTCN-3 template, testcase, function and altstep definitions and signatures and it shall document the language element identified by the `Identifier`.

The `Identifier` shall be a valid formal parameter or signature parameter name.

A documentation block may contain more than one `@param` tag and each tag shall document exactly one parameter.

NOTE 1: It is recommended that the order of the `@param` paragraphs within a documentation comment follows the order in which the formal parameters or signature parameters are declared.

NOTE 2: Using this tag as implicitly tagged documentation information inside definitions is specified in clause 7.

Examples

EXAMPLE 1: Using the @param tag for function formal parameters.

```
//* @param pl_user The name of user in the specified realm.
//* @param pl_password A known shared secret, the password of user of the specified username.
function f_anyFunc (in charsting pl_user, in template charstring pl_password) runs on MyCompType {
// body of the function
}
```

EXAMPLE 2: Using the @param tag for formal parameters of templates.

```
//* @param loc_int Value the "forwards" field (implicit in value parameter).
template MaxForwards MaxForwards_s(integer loc_int) := {
  fieldName := MAX_FORWARDS_E,
  forwards := loc_int
};
```

EXAMPLE 3: Using the @param tag for signature parameters.

```
//* @param pathAndFileName The absolute path and name of the file containing
//*        the list of elements; shall be a charstring.
//* @param findElement Sequence number of the element to be returned
//*       (not index!); shall be an integer.
signature FindXthElementInFile
  ( in charstring pathAndFileName, in integer findElement)
  return ListElementType
  exception (IndexOutOfBoundsExceptionType, FileNotFoundExceptionType);
```

6.7 The @purpose Tag

This tag shall be used to document the test purposes of test cases.

Syntactical Structure

```
@purpose [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag shall be used in documentation blocks preceding testcase definitions only and it shall describe the purpose of the test case, e.g. the test outcome that results a pass verdict.

A documentation block may contain at most one @purpose tag.

NOTE 1: Though this tag is attached to TTCN-3 testcase definitions the test purpose is the property of the whole test case behaviour, often involving one or several PTCs.

NOTE 2: This tag is used to document the aim of the test case and not the circumstances at which the pass verdict is set within the **testcase** definition. For the latter purpose use the @verdict tag.

Example

```
//* @purpose To check that the IUT releases the connection correctly on expiry of T302.
testcase tc_MyTestCase () runs on MyCompType system MySystemType {
// testcase code
}
```

6.8 The @remark Tag

This tag shall be used to add extra information not covered by other parts of the documentation block, such as highlighting a particular feature or aspect.

Syntactical Structure

```
@remark [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag relates to the language element it precedes.

A documentation block may contain more than one @remark tag.

Example

```
//* @remark This function should _not_ be called if the MTC acts as a client.
function MyTCActsAsServer () runs on MyCompType {
// function body
}
```

6.9 The @return Tag

This tag shall be used to provide information on the value(s) returned by the TTCN-3 object being documented.

Syntactical Structure

```
@return [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag shall only be used with signatures and functions with a return clause in their headers and it shall document the return type of the function or the signature.

A documentation block may contain at most one `@return` tag.

NOTE 1: It is recommended that the `@return` paragraph describes all possible values returned by the object and, if appropriate, also the return type.

NOTE 2: Using this tag as implicitly tagged documentation information inside definitions is specified in clause 7.

Examples

EXAMPLE 1: Using the @return tag for functions.

```
//* @return The number of bytes of the payload as integer value.
function CalcPayloadLength (in octetstring pl_oct) return integer {
// function body
}
```

EXAMPLE 2: Using the @return tag for signatures.

```
//* @return The element selected from the list based on its sequence number.
signature FindXthElementInFile
  ( in charstring pathAndFileName, in integer findElement)
  return ListElementType
  exception (IndexOutOfBoundsExceptionType, FileNotFoundExceptionType);
```

6.10 The @see Tag

This tag shall be used to refer to other globally visible TTCN-3 definitions in the same or another module. This tag can also be embedded into other tagged paragraphs.

Syntactical Structure

```
@see Identifier
```

The Identifier following the `@see` tag shall be a valid name of a TTCN-3 definition. The `Identifier` may be a non-qualified name or a name qualified with the name of the module where it is defined, using the dot notation, as specified in clause 8.2.3.1 of ES 201 873-1 [1].

Semantic Description

This tag documents the language element it precedes.

This tagged information can be embedded into the `freetext` part of other tagged paragraphs, where applicable. When used as a tagged paragraph on is own or embedded into other tagged paragraphs, it shall use the same syntactical structure, as specified above.

A documentation block may contain more than one `@see` tag.

Examples

EXAMPLE 1: Using the @see tag as tagged paragraph.

```
//* @see SIP_TypesAndConf.PX_T2
testcase SIP_CC_OE_CR_TI_006(inout CSeq loc_CSeq_s) runs on SipComponent system SipInterfaces {
//test case content
}
```

EXAMPLE 2: Using the @see tag embedded into another tagged paragraph.

```
//* @remark This function does not handles multiple events, you should use @see
//*       func_multiEvent for such purposes. The event to be processed
//*       shall be stored in the component variable v_event.
function f_handleEvent () runs on MyCompType {
// function body
}
```

EXAMPLE 3: Using the @see tag embedded into an implicitly tagged paragraph (see clause 7).

```
testcase SIP_CC_OE_CR_TI_006
  //* Request method decimal sequence number, @see SIP_TypesAndConf.CSeq
  (inout CSeq loc_CSeq_s)
   runs on SipComponent system SipInterfaces {
//test case content
}
```

6.11 The `@since` Tag

This tag indicates the version of the module at which the documented TTCN-3 object was added to the module.

Syntactical Structure

```
@since [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag documents the language element it precedes.

A documentation block may contain at most one `@since` tag.

Example

```
//* @since 493.0.1beta
function CalcLength (in octetstring pl_oct) return integer {
// function body
}
```

6.12 The `@status` Tag

This tag shall be used to describe the status of a particular TTCN-3 object like draft, reviewed, approved, deprecated, etc.

Syntactical Structure

```
@status Status [freetext]
```

`Status` shall define the actual status of the object and it shall conform to the naming rules for TTCN-3 identifiers (see annex A of ES 201 873-1 [1]). The content of the `freetext` part is not specified by the present document.

Semantic Description

This tag documents the language element it precedes.

A documentation block may contain at most one `@status` tag.

NOTE: It is recommended that the `@status` tag states the actual status of the object, when the status of the documented object has been changed, why it has been changed and, if the last change is that it was deprecated, what to use as a replacement.

Example

```
//* @status deprecated as of version 1.2, replaced by ExtensionHeaderList.
//*     ExtensionHeaderList has a more efficient implementation.
type record of Header ListOfExtensionHeaders;
```

6.13 The `@url` Tag

This tag shall be used to reference an external resource (like a file or a web page) to the TTCN-3 object.

Syntactical Structure

```
@url uri
```

The `@url` tag shall be followed by a valid URI. The general syntax of `uri` is defined in RFC 3986 [7]. The `uri` part of this tag shall use one of the URI schemes given below and shall comply with one of the following:

a) RFC 4248 [3] when the "file" or the "ftp" URI scheme is used;

b) RFC 2616 [4] when the "http" URI scheme is used;

c) RFC 2660 [5] when the "shttp" URI scheme is used; or

d) RFC 2818 [6] when the "https" URI scheme is used.

NOTE: As addition, TTCN-3 documentation tools may also support other URI schemes.

Semantic Description

This tag relates to the language element it precedes. This tagged information can be embedded into the `freetext` part of other tagged paragraphs, where applicable. When used as a tagged paragraph on is own or embedded into other tagged paragraphs, it shall use the same syntactical structure, as specified above.

A documentation block may contain more than one `@url` tag.

Examples

EXAMPLE 1: Using the @url tag with the "http" URI scheme

```
//* @url http://www.ietf.org/rfc/rfc3261.txt?number=3261
type record Request {
// definition content
}
```

EXAMPLE 2: Using the @url tag with the "file" URI scheme

```
//* @url file://D:/docs/DTS-TIPHON-06021-2.pdf
//* @url file://C:/images/Myimg.jpg
function f_SetUpTestConfiguration1 () runs on MyMTCType {
// function body
}
```

6.14 The `@verdict` Tag

This tag shall be used to provide information on the circumstances at which a given verdict is set.

Syntactical Structure

```
@verdict Verdict [freetext]
```

The `Verdict` following the `@verdict` tag shall have one of the `pass`, `fail`, or `inconc` values, optionally be followed by a text, the internal structure of which is not specified by the present document.

Semantic Description

This tag shall only be used with test cases, functions, and altsteps and it relates to the language element it precedes.

A documentation block may contain more than one `@verdict` tag and each tag shall document exactly one verdict value.

NOTE: It is recommended that the `freetext` part describes the reasons for assigning the given verdict.

Example

```
//* @verdict fail MAC Address for test cleanup could not be configured.
//* @verdict pass Registration successful.
function iUTRegistration() runs on SipComponent {
// function body
}
```

6.15 The `@version` Tag

This tag shall be used to describe the version of the TTCN-3 object being documented.

Syntactical Structure

```
@version [freetext]
```

For this tag no internal structure is defined by the present document.

Semantic Description

This tag documents the language element it precedes.

A documentation block may contain at most one `@version` tag.

Example

```
//* @version 493.0.1beta
module Mymodule {
// module content
}
```

7 Implicitly Tagged Documentation Information

In documentation blocks where tagged paragraphs are not allowed (see clause 6) the tags and `Identifier`-s shall be suppressed.

Syntactical Structure

```
[freetext]
```

The internal structure of implicitly tagged documentation information is not specified by the present document.

Semantic Description

Implicitly tagged documentation information shall be used instead of the tagged paragraphs @exception, @member, @param and @return only. Namely, no explicitly and implicitly tagged documentation information shall apply for the same documented element and implicitly tagged information shall only be used preceding the language elements allowed to be documented by the above tags. For example, a return type of a function can be documented by a @return tagged paragraph in the documentation block preceding the function definition or, alternatively, by an implicitly tagged information (de-facto a documentation block without any tags), placed directly before the return clause of the function definition's header.

The kind of the TTCN-3 language element following the documentation block determines the type of the implicit tag unambiguously. Documentation tools shall produce the code documentation entry in the same manner disregarding if it is originating from an implicitly or an explicitly tagged documentation information.

Implicitly tagged information can be placed before the related clause (e.g. parameter list, return clause, exception clause etc.) or directly before the documented element (e.g. field, parameter, return type, exception type etc.) and in both cases a single documentation block relates to the first applicable language element (field, parameter, return type, exception type etc.) following it. In this respect simple (non-documentation) TTCN-3 comments between the documentation block and the language element being documented shall be ignored.

Examples

EXAMPLE 1: Using the implicitly tagged information for signature parameters, return and exception types (using documentation blocks before the clauses).

```
signature FindXthElementInFile
// This information shall be associated to the parameter pathAndFileName
    //* The absolute path and name of the file containing the list of elements;
    //* shall be a charstring.
  ( in charstring pathAndFileName,

// This information shall be associated to the parameter findElement
   //* Sequence number of the element to be returned (not index!); shall be an integer.
    in integer findElement)

// This information shall be associated to the return type ListElementType
  //* The element selected from the list based on its sequence number.
  return ListElementType

// This information shall be associated to the exception type IndexOutOfBoundsExceptionType
  //* The index passed is too large
  exception (IndexOutOfBoundsExceptionType,

// This information shall be associated to the exception type FileNotFoundExceptionType
  //* value1 - the file with the specified name has not been found
  //* value2 - the specified path does not exists
             FileNotFoundExceptionType);
```

EXAMPLE 2: Using the implicitly tagged information for signature parameters, return and exception types (using documentation blocks before the elements being documented).

```
signature FindXthElementInFile (
// This information shall be associated to the parameter pathAndFileName
    //* The absolute path and name of the file containing the list of elements.
    in charstring pathAndFileName,

// This information shall be associated to the parameter findElement
   //* Sequence number of the element to be returned (not index!).
    in integer findElement)

// This information shall be associated to the return type ListElementType
  return /** The element selected from the list based on its sequence number.*/ ListElementType

// This information shall be associated to the exception type IndexOutOfBoundsExceptionType
  exception (
    /** The index passed is too large */              IndexOutOfBoundsExceptionType,

// This information shall be associated to the exception type FileNotFoundExceptionType
  /** value1 - the file with the specified name has not been found
      value2 - the specified path does not exists */  FileNotFoundExceptionType);
```

EXAMPLE 3: Using the implicitly tagged information for function formal parameters and return types.

```
function CalcPayloadLength
  //* The complate encoded message; the length of the payload which shall be calculated
  (in octetstring pl_oct)

  //* The number of bytes of the payload as integer value
  return integer {
// function body
}
```

EXAMPLE 4: Using the implicitly tagged information for members of structured type fields.

```
type record IP6extensionHeader {
// Pls. note, this information shall be associated to the whole embedded record of record type
  //* List of extension headers defined by RFC 2460.
  record of record {
```

```
    //* Headers specified in RFC 2460.
    ExtensionHeaders header,

    //* Extra header field for headers not specified in RFC 2460.
    octetstring      unknown
  }  extensionHeaderFields
}
```

EXAMPLE 5: Using the implicitly tagged information for members of enumerated types.

```
type enumerated SupportFlag {
  //* If this non default value is taken, the SS has prohibited acknowledging RLC SDUs.
  ackProhibited (0),

  //* If this non default value is taken, the SS has continued acknowledging RLC SDUs.
  ackContinued (1),

  //* This default value indicates that the SS does not perform the operation mentioned
  //* in CRLC_ProhibitRLC_Ack_REQ, but performs the suspension / resume of UL RLC PDU data.
  noNeed (2)
}
```

EXAMPLE 6: Using the implicitly tagged information for members of port types.

```
type port IPv6PortType
  message {
    //* The complete IPv6 packet (not encoded)
    inout_IPv6Message;

    //* When an IPv6 packet decoding fails in the TA, it will pass the received packet to the
    //* TE as octetstring.
    in octetstring
  }
```

EXAMPLE 7: Using the implicitly tagged information for members of component types.

```
type component MyPTCType {
  //* Variable to store incoming IPv6 packets
  var IPv6Message v_IPv6Message;

  //* Generic send (no-response) guard timer
  timer T_Guard := 10.0;

  //* Definition required for LibIpv6COmp type compatibility.
  port IPv6PortType IPv6Port
}
```

EXAMPLE 8: Using the implicitly tagged information for module parameters.

```
modulepar charstring
  //* Display name used by the ETS to exchange SIP messages on MTC side.
  PX_ETS_DISP_NAME := "TESTER1",
  //* IP address used by the ETS to exchange SIP messages on MTC side.
  PX_ETS_IPADDR := "127.0.0.1";
modulepar integer
//* Port number used by the ETS to exchange SIP messages on MTC side.
  PX_ETS_PORT := 5061;
```

EXAMPLE 9: The documentation block is associated to the first appropriate language element.

```
function f_anyFunc
// Pls. note that the documentation tool shall associate the documentation block below to the para-
// meter pl_user, though the test suite writer meant the second line for the parameter pl_password
  //* The name of user in the specified realm.
  //* A known shared secret, the password of user of the specified username.
  (in charstring pl_user, in template charstring pl_password) runs on MyCompType {
// body of the function
}
```

8 Error processing rules

TTCN-3 documentation tools shall follow the error processing rules in this clause. As the directing principle, documentation tools shall try to avoid using erroneous information in the generated documentation. Hence, error situations not regulated in this clause shall be resolved by ignoring the concerned documentation information.

NOTE 1: Tools should notify the user on all kind of problems found, including ignored ones. This is, however, not regulated by the present document.

Subsequent documentation blocks related to the same TTCN-3 definition shall be combined to one logical documentation block that is related to the respective declaration. Non-documentation comments enclosed between documentation blocks or between the last documentation block and the definition following the block do not contribute to the logical documentation block and shall be ignored.

Unrecognized text formatting HTML elements shall be ignored (see clause 5.3).

Inappropriately placed documentation blocks shall be ignored.

NOTE 2: At least the following error cases fall into this category:

- documentation blocks that are directly followed by a TTCN-3 language element which shall not have a documentation comment (e.g. followed by a runs on or a system clause),
- documentation blocks placed at positions where documentation comments are not allowed (e.g. within a function-, altstep- or testcase body or within the control part),
- documentation blocks that are not followed by any TTCN-3 language element at all (e.g. before the closing bracket of a module).

NOTE 3: Please note that this error case also includes inappropriately placed implicit tags.

Text before the first tagged paragraph in bodies of documentation blocks preceding TTCN-3 modules, control parts, groups and definitions of global visibility and text that cannot be associated with the preceding tag owing to its syntax rule (e.g. because the tagged paragraph do not have a `freetext` part), shall be ignored.

Tagged paragraphs with unrecognized tags shall be ignored.

NOTE 4: This rule entails that fragments of `freetext` parts of tagged paragraphs starting with a single commercial at ("@") character (that is, followed by any other character as @) and ended by the end of the `freetext` part is suppressed, including the commercial at character itself.

Inappropriate tagged paragraphs shall be ignored. When the error occurs due to breaking the multiplicity rule defined for a given kind of tag (see also the notes below), the last tagged paragraph of the kind shall be used for documentation purposes and the other(s) shall be handled as inappropriate.

NOTE 5: At least the following type of errors fall into this category:

- The tagged paragraph is not allowed for the TTCN-3 definition following it (e.g. a @config tag used in the documentation block preceding a function definition).
- The TTCN-3 language element the tagged paragraph is referring to does not exist (e.g. a `@return` tag used in the documentation block preceding a function definition without a return clause or a `@member` tag is referring to a non-existing member names).
- Several tagged paragraph of the same kind is defined where only a single occurrence is allowed (e.g. several `@purpose` tags.
- Several tagged paragraphs of the same kind using the same `Identifier` in the same documentation block (e.g. subsequent `@exception` tags for the same exception type).

Url tags (see clause 6.13) with non-complying `uri` parts shall be ignored.

When a TTCN-3 language element is documented by both an explicitly tagged paragraph and an implicitly tagged documentation information, only the implicitly tagged documentation information shall contribute to the generated documentation.

Fragments of implicitly tagged documentation information starting with a single commercial at ("@") character (that is, followed by any other character as @) and ended by the end of the documentation block, shall be ignored, including the commercial at character itself.

Annex A (informative): Where Tags can be used

Table A.1 describes which tags can be used in documenting which TTCN-3 language element.

Table A.1: Relation of documentation tags and TTCN-3

	Simple Data Types	Structured Data Types	Component Types	Port Types	Modulepars	Constants (TTCN-3 and external)	Templates	Signatures	Functions (TTCN-3 and external)	Altsteps	Test Cases	Modules	Groups	Control Parts	Used in implicit form (see clause 7)	Embedded in other tags
@author	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@config											X					
@desc	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@exception								X							X	
@member		X[1]	X	X	X[1]	X[1]	X[1]								X	
@param							X	X	X	X	X				X	
@purpose											X					
@remark	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@return								X	X						X	
@see	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@since	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@status	X	X	X	X	X	X	X	X	X	X	X	X	X	X		
@url	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X
@verdict									X	X	X					
@version	X	X	X	X	X	X	X	X	X	X	X	X	X	X		

NOTE: [1] Preceding language elements of record, set, union or enumerated types only.

Annex B (informative): Bibliography

- IETF RFC 1738 (December 1994): "Uniform Resource Locators (URL)".

History

Document history		
V3.2.1	December 2007	Publication
V3.4.1	July 2008	Membership Approval Procedure MV 20080829: 2008-07-01 to 2008-07-29
V3.4.1	September 2008	Publication