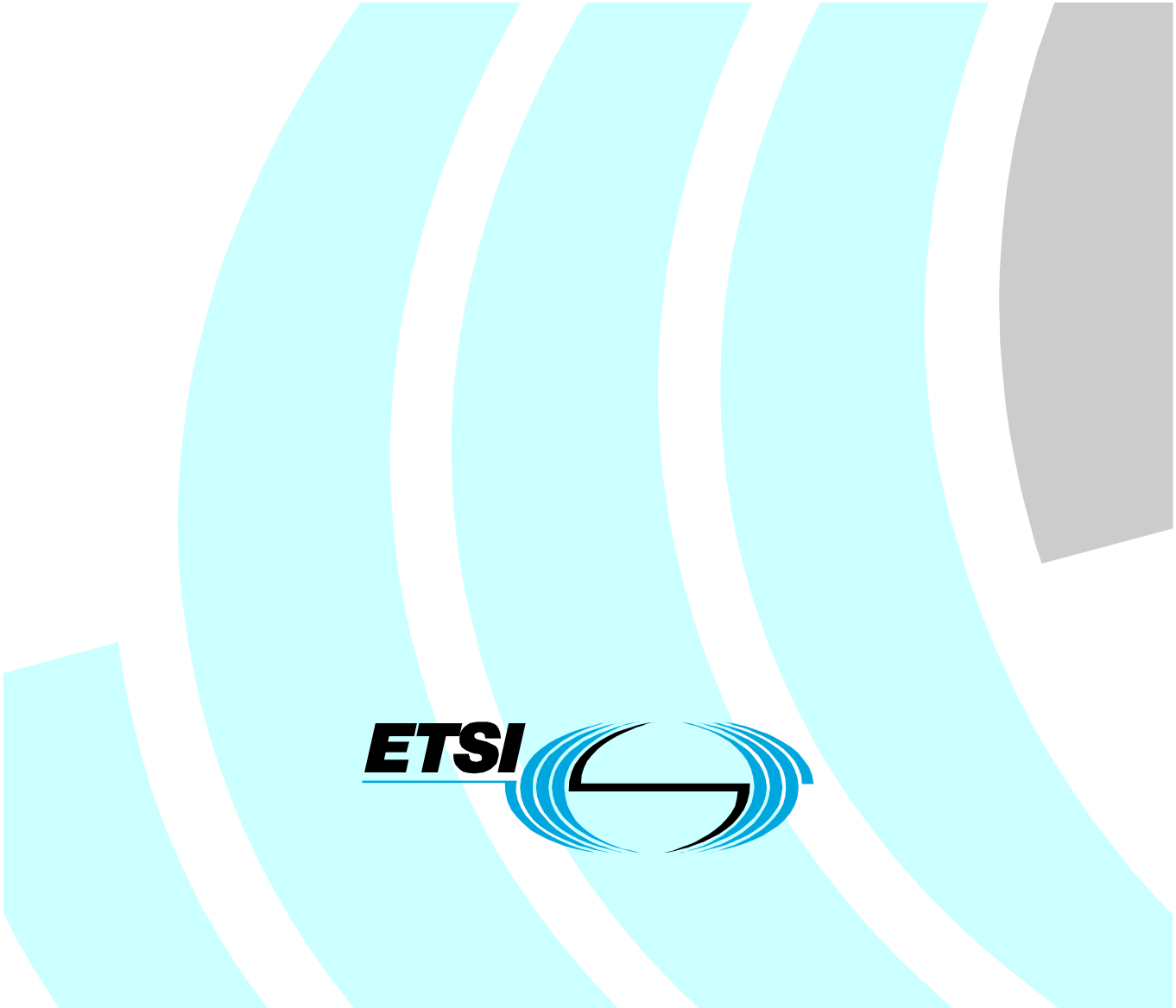
ETSI ES 201 873-2 V3.2.1 (2007-02)

ETSI Standard

Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
Part 2: TTCN-3 Tabular presentation Format (TFT)



Reference
RES/MTS-00090-2[2] ttcn3 tft

Keywords
methodology, MTS, testing, TFT, 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 2 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

1 Scope

The present document defines the tabular presentation format of TTCN Version 3 (or TTCN-3). The present document is based on the TTCN-3 core language defined in ES 201 873-1 [1].

The specification of other formats is outside the scope of the present document.

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.

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

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

[2] ETSI ES 201 873-4: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics".

3 Abbreviations

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

ASN.1	Abstract Syntax Notation One
ATS	Abstract Test Suite
BNF	Backus-Nauer Form
MTC	Master Test Component
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
TFT	Tabular presentation Format for TTCN-3
TTCN	Testing and Test Control Notation

4 Introduction

The Tabular presentation Format for TTCN-3 (TFT) is a graphical format that is similar in appearance and functionality to earlier versions of TTCN, which are conformance testing oriented. The core language of TTCN-3 is defined in ES 201 873-1 [1] and provides a full text-based syntax, static semantics as well as defining the use of the language with ASN.1. The operational semantics are defined in ES 201 873-4 [2]. The tabular format provides an alternative way of displaying the core language as well as emphasizing those aspects that are particular to the requirements of a standardized conformance test suite.

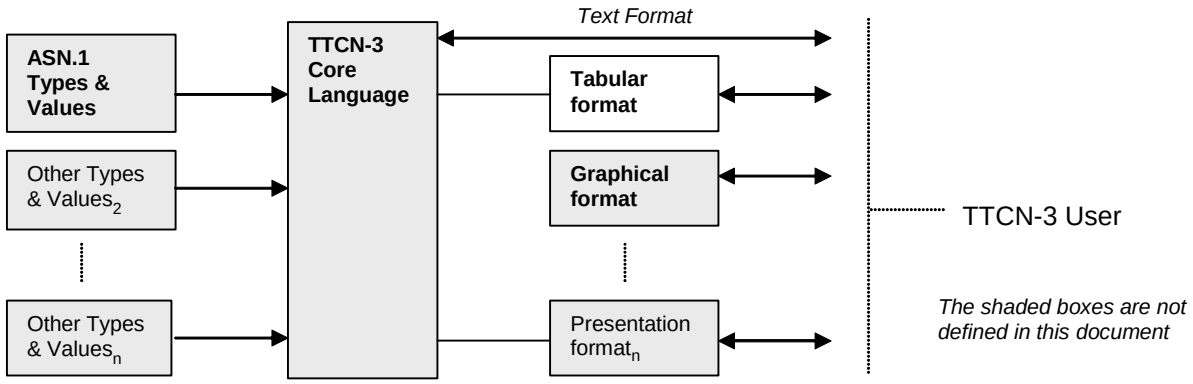


Figure 1: User's view of the core language and the various presentation formats

The core language may be used independently of the tabular presentation format. However, the tabular format cannot be used without the core language. Use and implementation of the tabular presentation format shall be done on the basis of the core language.

The present document defines the:

a) proformas;

b) syntax mappings;

c) additional static semantics;

d) operational semantic restrictions;

e) display and other attributes.

Together these characteristics form the tabular presentation format.

5 Conventions

This clause defines the conventions, which have been used when defining the TTCN proformas and the TTCN core language grammar.

5.1 Syntactic metanotation

Table 1 defines the metanotation used to specify the extended BNF grammar for TTCN (henceforth called BNF).

Table 1: The TTCN.MP Syntactic Metanotation

::=	is defined to be
abc xyz	abc followed by xyz
|	alternative
[abc]	0 or 1 instances of abc
{abc}	0 or more instances of abc
{abc}+	1 or more instances of abc
(...)	textual grouping
abc	the non-terminal symbol abc
abc	a terminal symbol abc
"abc"	a terminal symbol abc

The BNF productions are defined in annex A of ES 201 873-1 [1].

5.2 Specification text

a) **Bold text** shall be used for references to proforma fields.

b) *Italics text* shall be used for references to the TTCN-3 core language BNF productions.

c) **`Bold courier new`** text shall be used for core language keywords.

5.3 Proformas

a) **`Bold text`** shall appear verbatim in each actual table in a TTCN-3 module.

b) *`Italics text`* shall not appear verbatim in a TTCN-3 module. This font is used to indicate that actual text shall be substituted for the italicized symbol. Syntax requirements for the actual text can be found either following the definition of the proforma or in the TTCN-3 core language BNF. Square brackets before and after the *`Italics text`* indicates that inclusion of the text into the given field of the proforma is optional.

5.4 Core language

a) **`Bold text`** of characters in quotes (e.g. '{') is used for reserved keywords and terminals in the core language.

b) *`Italics text`* shall not appear verbatim in a TTCN-3 module. This font is used to indicate that actual text shall be substituted for the italicized symbol. Syntax requirements for the actual text can be found either following the definition of the proforma or in the TTCN-3 core language BNF.

c) The "**`...`**" notation is a place holder for any arbitrary contents that is not explicitly shown.

5.5 General Mapping Rules

The mapping between the tabular presentation format and the TTCN-3 core language consists of a set of transformations. For every syntactical element within each proforma there is an associated transformation. The transformations make it also possible to transform any core language module into a tabular representation.

These transformations fall into two classes. The first class directly converts between a tabular element and a core language construct with the same meaning. The second class converts between a tabular element and an associated core language construct, which has no meaning at the core language level.

A typical example for the first class of transformations would be an identifier field. This field can be directly transformed from tabular to the core language and retains its meaning i.e. identifying same language element.

The second class of transformations is typically some form of comment or directive as to how a language element should be displayed in the presentation format. These elements have no direct meaning in the core language and are expressed using the *WithStatement*.

The syntax and semantics specified in the present document are specific to the ETSI tabular presentation format. In order to unambiguously identify within the core language which presentation format is being used the following special display statement shall be specified as the first display statement associated with the TTCN-3 core language module:

```
1: module TTCN3ModuleId    "{"
2:      ...
3: "}" with "{"
4:      display """ "presentation format" ":=" "ETSI Tabular version"
5:      MajorVersion "." MinorVersion """ ";"
6:      ...
7: "}"
```

NOTE: All With Statements associated with a given proforma should be grouped together in a contiguous list.

The **Group** fields in the proformas are never translated into *WithStatement*s but are derived from the actual group structure of the module specification.

6 Proformas

6.1 Test Suite Control

Test Suite Control	
Module Name	*TTCN3ModuleId*
Version	*[TabFreeText]*
Date	*[TabFreeText]*
Base Standard Ref	*[TabFreeText]*
Test Standard Ref	*[TabFreeText]*
PICS Ref	*[TabFreeText]*
PIXIT Ref	*[TabFreeText]*
Test Method(s)	*[TabFreeText]*
Encoding	*[TabFreeText]*
Comments	*[TabFreeText]*

Local Def Name	**Type**	**Initial Value**	**Comments**
[VarConstOrTimerIdentifier]	*[ConstTypeOrTimer]*	*[Expression]*	*[TabFreeText]*
...	...	...	...

Behaviour	
ModuleControlBody	
Detailed Comments	*[TabFreeText]*

Figure 2: Test Suite Control Proforma

6.1.1 Mapping

The Test Suite Control proforma is translated into three parts. The first part consists of the header fields and the **Detailed Comments** field, which are converted to display attributes within the *WithStatement* associated with the overall TTCN-3 module. The **Module Name** field is mapped to the module identifier.

The second part consists of local constants, variables and timers defined in the control part. These definitions can occur anywhere in the control part of the core language, but for the proforma they are separated from the rest of the module control body and displayed in a separate table. The order of the definitions shall be preserved, since the definitions can depend on each other. The **Type** column shall be set to the keyword **timer** for all timers and to the constant type preceded by the keyword **const** for all constants. The **Comments** fields of the local definitions table are converted to display attributes within the *WithStatement* associated with the control part of the TTCN-3 core language module.

The third part is the control part of the TTCN-3 core language module minus the local constants, variables and timers.

```
 1: module TTCN3ModuleId "{"
 2:     control "{"
 3:  var Type VarIdentifier [":=" Expression] ";"
 4:  timer TimerIdentifier [":=" Expression] ";"
 5:  const Type ConstIdentifier ":=" ConstantExpression;
 6:         ModuleControlBody
 7:     "}" with "{"
 8:         { VarConstOrTimerCommentsAttribute }
 9:     "}"
10: "}" with "{"
11:     ModuleAttributes
12:  [EncodeAttribute;]
13: "}"
```

EXAMPLE:

Test Suite Control			
Module Name	Example1		
Version	1.01		
Date	19 July 2001		
Base Standards Ref	ITU-T Recommendation Q.123		
Test Standards Ref	ITU-T Recommendation Q.123.1		
PICS Ref	ITU-T Recommendation Q.123.2, Annex A		
PIXIT Ref	ITU-T Recommendation Q.123.2, Annex B		
Test Method(s)	local		
Encoding	BER		
Comments	ATS written by STF 133		
Local Def Name	**Type**	**Initial Value**	**Comments**
PI	const float	3.14	the ratio
x	float	PI * 2	double PI
t1	timer	15	a 15 second timer
Behaviour			

/* group1/ */
/* group1_1/ */
execute(test1);
execute(test2);
/* group1_2/ */
execute(test3);
execute(test4);
/* group2/ */
execute(test5);			

Detailed Comments	detailed comments		

Maps to:

```
 1: module Example1 {
 2:     control {
 3:         const float PI := 3.14;
 4:         var float x := PI * 2;
 5:         timer t1 := 15;
 6:
 7:         /* group1/ */
 8:             /* group1_1/ */
 9:                 execute(test1());
10:                 execute(test2());
11:             /* group1_2/ */
12:                 execute(test3());
13:                 execute(test4());
14:             /* group2/ */
15:                 execute(test5());
16:     } with {
17:         display (PI) "comments := the ratio";
18:         display (x) "comments := double PI";
19:         display (t1) "comments := a 15 second timer";
20:     }
21: } with {
22:     display "presentation format := ETSI Tabular version 1.0";
23:     display "module version := 1.01";
24:     display "module date := 19 July 2001";
25:     display "module base standards ref := ITU-T Recommendation Q.123";
26:     display "module test standards ref := ITU-T Recommendation Q.123";
27:     display "module pics ref := ITU-T Recommendation Q.123, Annex A";
28:     display "module pixit ref := ITU-T Recommendation Q.123, Annex A";
29:     display "module test method := local";
30:     display "module comments := ATS written by STF 133";
31:     display "module detailed comments := detailed comments";
32:     encode "BER";
33: }
```

6.2 Test Suite Parameters

Test Suite Parameters				
Name	**Type**	**Initial Value**	**PICS/PIXIT Ref**	**Comments**
. *ModuleParIdentifier* .	. *ModuleParType* .	. *[ConstantExpression]* .	. *[TabFreeText]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*			

Figure 3: Test Suite Parameters Proforma

6.2.1 Mapping

All entries in the Test Suite Parameters proforma are mapped to the *ModuleParList*s in *ModuleParameterDefs* of the associated TTCN-3 module. If there is more than one *ModuleParameterDef*then all *ModuleParList*s are collected and represented in one **Test Suite Parameters** proforma.

The **PICS/PIXITref** and **Comments** fields are mapped to display attributes qualified by the parameter identifier within the *WithStatement*s associated with the enclosing *ParamDef*. The **Detailed Comments** field is mapped to a display attribute within the *WithStatement* associated with the enclosing *ParamDef*.

```
1: module TTCN3ModuleId "{"
2: parameters "{" ModuleParList "}"
3:  with "{"
4:  [ModuleParPicsPixitRefAttribute ";"]
5:  [ModuleParComments ";"]
6:  [DetailedComments ";"]
7: "}"
8: "}"
```

EXAMPLE:

Test Suite Parameters				
Name	**Type**	**Initial Value**	**PICS/PIXIT Ref**	**Comments**
CAP_1	boolean	true	A.1.3	option 1 implemented
Tall	float	600.0	A.1.4	overall module timer
Detailed Comments	detailed comments			

Maps to:

```
  1: module MyModule{
  2:  parameters { boolean CAP_1 := true, float Tall := 600.0 }
  3:  with {
  4:  display (CAP_1) "pics/pixit ref := A.1.3";
  5:  display (CAP_1) "comments := option 1 implemented";
  6:  display (Tall) "pics/pixit ref := A.1.4";
  7:  display (Tall) "comments := overall module timer";
  8:  display "detailed comments := detailed comments"
  9:  }
 10: }
```

6.3 Module Imports

Imports	
Source Name	*GlobalModuleId [**recursive**]*
Source Language	*[LanguageSpec]*
Group	*[GroupReference]*
Source Ref	*[TabFreeText]*
Encoding	*[TabFreeText]*
Comments	*[TabFreeText]*
Type	**Name**
. *[ImportType]* .	. *ImportSpecification* .
Detailed Comments	*[TabFreeText]*

Figure 4: Imports Proforma

6.3.1 Mapping

The Imports proforma is mapped to an *ImportDef* statement in the TTCN-3 core language. The **Source Name**, **Source Language**, **Type** and **Name** fields are directly used in the corresponding core language *ImportDef* statement. The **Source Ref**, **Comments** and **Detailed Comments** fields are translated into display attributes within the *WithStatement* associated with the *ImportDef* statement. The **Encoding** field is translated into an encode attribute within the *WithStatement* associated with the *ImportDef* statement.

If all definitions of a module are imported then the *ImportType* shall be empty and the *ImportSpecification* shall use the keyword **`all`**.

```
  1: module TTCN3ModuleId "{"
  2:  ImportDef
  3:     with "{"
  4:  [ImportsSourceRefAttribute ";"]
  5:  [CommentsAttribute ";"]
  6:  [ImportsSourceDefinitionCommentsAttribute ";"]
  7:  [DetailedCommentsAttribute ";"]
  8:  [EncodeAttribute ";"]
  9:  "}"
 10: "}"
```

EXAMPLE:

Imports			
Source Name	ModuleA recursive		
Source Language	ASN.1:1997		
Group			
Source Ref	EN 800 900 version 2		
Encoding	BER		
Comments	importing declarations from ATS		
Type		**Name**	**Comments**
constant		all except foobar	
Type		MyType	foobar
Group		AtoU_CTR	
Detailed Comments	detailed comments		

Maps to:

```
  1: module MyModule {
  2:  import from ModuleA recursive language "ASN.1997" {
  3:  const all except foobar;
  4:  type MyType;
  5:  Group AtoU_CTR;
  6:  } with {
  7:  display "imports source ref := EN 800 900 version 2";
  8:  display "comments := importing declarations from ATS";
  9:  display "detailed comments := detailed comments";
 10:  encode "BER";
 11:  }
 12: }
```

6.4 Simple Types

Simple Types			
Group	*[GroupReference]*		
Name	**Definition**	**Encoding**	**Comments**
. *SubTypeIdentifier* .	. *Type [ArrayDef] [SubTypeSpec]* .	. *[TabFreeText]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 5: Simple Types Proforma

6.4.1 Mapping

The Simple Types proforma is mapped to a series of simple type definition statements on the same group level. Simple type definitions are all *SubTypeDef* type definitions.

The **Detailed Comments** field is mapped to a display attribute within the *WithStatement* associated with the enclosing group or the module. The **Encoding** and **Comments** fields are mapped to encoding and display attributes respectively within the *WithStatement* associated with the respective simple type definition.

```
  1: module TTCN3ModuleId "{"
  2:  type Type SubTypeIdentifier [ArrayDef] [SubTypeSpec] with "{"
  3:  [EncodeAttribute ";"]
  4:  [CommentsAttribute ";"]
  5: "}" with "{"
  6:  [SimpleTypesDetailedCommentsAttribute ";"]
  7: "}"
```

EXAMPLE:

Simple Types			
Group	SimpleTypes/		
Name	**Definition**	**Encoding**	**Comments**
EQ_NUMBER	integer (1 .. 20)	PER	God knows
Detailed Comments	detailed comments		

Maps to:

```
  1: module MyModule {
  2:  group SimpleTypes {
  3:  type integer EQ_NUMBER (1..20) with {
  4:  encode "PER";
  5:  display "comments := God knows";
  6:  }
  7:  } with {
  8:  display "simple types detailed comments := detailed comments";
  9:  }
 10: }
```

6.5 Structured Types

Structured Type			
Name	*StructTypeIdentifier[StructDefFormalParList]*		
Group	*[GroupReference]*		
Structure	*StructureType*		
Encoding	*[TabFreeText]*		
Comments	*[TabFreeText]*		
Field Name	**Field Type**	**Field Encoding**	**Comments**
. *FieldIdentifier* . . .	. *Type [ArrayDef] [SubTypeSpec] [OptionalKeyword]* .	. *[TabFreeText]* . . .	. *[TabFreeText]* . . .
Detailed Comments	*[TabFreeText]*		

Figure 6: Structured Type Proforma

6.5.1 Mapping

The Structured Type proforma is mapped to a structured type definition statement in TTCN-3. The following types will use this proforma: *RecordDef*, *UnionDef* and *SetDef*.

The **Comments** and **Detailed Comments** fields are mapped to display attributes in the corresponding *WithStatement*, and the **Encoding** field is mapped to an encode attribute in the corresponding *WithStatement*. The **Comments** and **Field Encoding** fields of each field element are mapped to a display and an encode attribute respectively, qualified by the *FieldIdentifier* in the corresponding *WithStatement*.

```
  1: module TTCN3ModuleId "{"
  2:  type StructureType StructTypeIdentifier [StructDefFormalParList] "{"
  3:  {Type FieldIdentifier [ArrayDef] [SubtypeSpec] [OptionalKeyword]}
  4:  "}" with "{"
  5:  [EncodeAttribute ";"]
  6:  [CommentsAttribute ";"]
  7:  {FieldCommentsAttribute ";"}
  8:  {FieldEncodeAttribute ";"}
  9:  [DetailedCommentsAttribute ";"]
 10:  "}"
 11: "}"
```

EXAMPLE:

Structured Type			
Name	routing_label(SLSel_Type)		
Group			
Structure	record		
Encoding	BER		
Comments	header for routing info		
Element Name	**Type Definition**	**Field Encoding**	**Comments**
DestPC	BIT_14		destination point code
OrigPC	BIT_14		origination point code
SLSel	SLSel_Type	PER	signalling link selection
Detailed Comments	overrides previous definitions		

Maps to:

```
  1: module MyModule {
  2:  type record routing_label(SLSel_Type) {
  3:  BIT_14 DestPC,
  4:  BIT_14 OrigPC,
  5:  SLSel_Type SLSel
  6:  } with {
  7:  encode "BER";
  8:  display "comments := header for routing info";
  9:  display (DestPC) "comments := destination point code";
 10:  display (OrigPC) "comments := origination point code";
 11:  display (SLSel) "comments := signalling link selection";
 12:  encode (SLSel) "PER";
 13:  display "detailed comments := overrides previous definition";
 14:  }
 15: }
```

6.6 SequenceOf Types

SequenceOf Types					
Group	*[GroupReference]*				
Name	**Type**	**Kind**	**Length**	**Encoding**	**Comments**
. *StructTypeIdentifier* .	. *Type [SubTypeSpec]* .	. *RecordOrSet* .	. *[StringLength]* .	. *[TabFreeText]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*				

Figure 7: SequenceOf Types Proforma

6.6.1 Mapping

The SequenceOf Types proforma is mapped to a series of sequenceof type definition statements on the same group level. This proforma shall be used for *RecordOfDef* and *SetOfDef* type definitions.

The **Detailed Comments** field is mapped to a display attribute within the *WithStatement* associated with the enclosing group or the module. The **Encoding** and **Comments** fields are mapped to encoding and display attributes respectively within the *WithStatement* associated with the respective SequenceOf type definition.

```
 1: module TTCN3ModuleId "{"
 2:  type record of [StringLength] Type StructTypeIdentifier [SubTypeSpec]
 3:  with {
 4:  [EncodeAttribute ";"]
 5:  [CommentsAttribute ";"]
 6:  }
 7:  type set of [StringLength] Type StructTypeIdentifier [SubTypeSpec]
 8:  with {
 9:  [EncodeAttribute ";"]
10:  [CommentsAttribute ";"]
11:  }
12: } with {
13:  [SequenceOfTypesDetailedCommentsAttribute ";"]
14: }
```

EXAMPLE:

SequenceOf Types					
Group	SequenceOfTypes/				
Name	**Type**	**Kind**	**Length**	**Encoding**	**Comments**
RecordOfIntegers	integer(1..10)	record	10	BER	ten integers
SetOfBooleans	boolean	set	3	PER	three booleans
Detailed Comments	example sequenceof types				

Maps to:

```
 1: module MyModule {
 2:  group SequenceOfTypes {
 3:  type record of length(10) integer RecordOfIntegers(1..10) with {
 4:  encode "BER";
 5:  display "comments := ten integers";
 6:  }
 7:  type set of length(3) boolean SetOfBooleans with {
 8:  encode "PER";
 9:  display "comments := three booleans";
10:  }
11:  } with {
12:  display "sequenceof types detailed comments := example sequenceof types";
13:  }
14: }
```

6.7 Enumerated Type

Enumerated Type			
Name	*EnumTypeIdentifier*		
Group	*[GroupReference]*		
Encoding	*[TabFreeText]*		
Comments	*[TabFreeText]*		
Enumeration Name		**Enumeration Value**	**Comments**
. *EnumerationIdentifier* .		. *[Number]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 8: Enumerated Type Proforma

6.7.1 Mapping

The Enumerated Type proforma is mapped to an enumerated type definition statement in the TTCN-3 core language. The **Comments** and **Detailed Comments** fields are mapped to display attributes in the corresponding *WithStatement*, and the **Encoding** field mapped to an encode attribute within the corresponding *WithStatement*. The **Comments** fields of each enumeration are mapped to display attributes qualified by the *EnumerationIdentifier* in the corresponding *WithStatement*.

```
 1: module TTCN3ModuleId "{"
 2:  type enumerated EnumTypeIdentifier "{"
 3:  EnumerationIdentifier ["(" Number ")"]
 4:  {"," EnumerationIdentifier ["(" Number ")"]}
 5:  } with {
 6:  [EncodeAttribute ";"]
 7:  [CommentsAttribute ";"]
 8:  {NamedValueCommentsAttribute ";"}
 9:  [DetailedCommentsAttribute ";"]
10:  }
11: }
```

EXAMPLE:

Enumerated Type		
Name	Weekdays	
Group		
Encoding	BER	
Comments	days of the week	
Enumeration Name	**Enumeration Value**	**Comments**
Monday	1	
Tuesday	2	
Wednesday	3	half way there
Thursday	4	
Friday	5	TGIF
Saturday	6	
Sunday	7	
Detailed Comments	wish it were Friday	

Maps to:

```
 1: module MyModule {
 2:  type enumerated Weekdays {
 3:  Monday(1), Tuesday(2), Wednesday(3), Thursday(4), Friday(5),
 4:  Saturday(6), Sunday(7)
 5:  } with {
 6:  encode "BER";
 7:  display "comments := days of the week";
 8:  display (Wednesday) "comments := half way there";
 9:  display (Friday) "comments := TGIF";
10:  display "detailed comments := wish it were Friday";
11:  }
12: }
```

6.8 Port Types

Port Type			
Name	*PortTypeIdentifier*		
Group	*[GroupReference]*		
Communication Model	*PortModelType*		
Comments	*[TabFreeText]*		
Type/Signature		**Direction**	**Comments**
. *TypeOrSignature* .		. *InOutOrInout* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 9: Port Type Proforma

6.8.1 Mapping

The Port Type proforma is mapped to a port type definition in the TTCN-3 core language. The **Comments** and **Detailed Comments** fields are mapped to display attributes in the corresponding *WithStatement*. The **Comments** fields of the types and signature table are mapped to display attributes in the corresponding *WithStatement* qualified by the type or signature identifier. There will always be one row for every type or signature.

The **Type/Signature** field is set to the keyword **all** if all types or all procedure signatures defined in the module can be passed over that communication port.

```
 1: module TTCN3ModuleId "{"
 2:  type port PortTypeIdentifier PortModelType "{"
 3:  PortTypeDef
 4:  "}" with "{"
 5:  [CommentsAttribute ";"]
 6:  {TypeOrSignatureCommentsAttribute ";"}
 7:  [DetailedCommentsAttribute ";"]
 8:  "}"
 9: "}"
```

EXAMPLE:

Port Type			
Name	MyPortType		
Group			
Communication Model	message		
Comments	example port type		
Type/Signature		**Direction**	**Comments**
MsgType1		in	first comment
MsgType2		in	second comment
MsgType3		out	
Detailed Comments	detailed comment		

Maps to:

```
 1: module MyModule {
 2:  type port MyPortType message {
 3:  in MsgType1;
 4:  in MsgType2;
 5:  out MsgType3;
 6:  } with {
 7:  display "comments := example port type";
 8:  display (MsgType1) "comments := first comment";
 9:  display (MsgType2) "comments := second comment";
10:  display "detailed comments := detailed comment";
11:  }
12: }
```

6.9 Component Types

Component Type			
Name	*ComponentTypeIdentifier*		
Group	*[GroupReference]*		
Comments	*[TabFreeText]*		
Local Def Name	**Type**	**Initial Value**	**Comments**
. *VarConstOrTimerIdentifier* .	. *TypeOrTimer [ArrayDef]* .	. *[ConstantExpression | Expression]* .	. *[TabFreeText]* .
Port Name	**Port Type**		**Comments**
. *PortIdentifier* .	. *PortType[ArrayDef]* .		. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 10: Component Type Proforma

6.9.1 Mapping

The Component Type proforma is mapped to a component type definition in the TTCN-3 core language. The proforma is translated into three parts.

The first part consists of the header **Comments** and **Detailed Comments** fields, which are converted to display attributes within the *WithStatement* associated with the component type definition.

The second part consists of local constants, variables and timers defined in the component type. These definitions can occur anywhere in the component type definition of the core language, but for the proforma they are separated from the port instances and displayed in a separate table. The order of their definition shall be preserved, since the definitions can depend on each other. The **Type** column shall be set to the keyword **`timer`** for all timers and to the constant type preceded by the keyword **`const`** for all constants. There will always be one row for every constant, variable or timer. The **Comments** column of this table is converted to display attributes qualified by the local definition's identifier within the *WithStatement* associated with the component type definition.

The third part consists of port instances defined in the component type. Any array definitions are appended to the port type. There will always be one row for every port instance. The **Comments** column of this table is converted to display attributes qualified by the *PortIdentifier* within the *WithStatement* associated with the component type definition.

```
 1: module TTCN3ModuleId "{"
 2:  type component ComponentTypeIdentifier "{"
 3:  var Type VarIdentifier [":=" Expression] ";"
 4:  timer TimerIdentifier [":=" Expression] ";"
 5:  const Type ConstIdentifier ":=" ConstantExpression ";"
 6:  PortList
 7:  "}" with "{"
 8:  [CommentsAttribute ";"]
 9:  {PortCommentsAttribute ";"}
10:  [DetailedCommentsAttribute ";"]
11:  "}"
12: "}"
```

EXAMPLE:

Component Type			
Name	MyComponentType		
Group			
Comments	an example component type		
Local Def Name	**Type**	**Initial Value**	**Comments**
PI	const float	3.14	the ratio
x	float	PI * 2	double PI
t1	timer	15 min	a 15 second timer
Port Name	**Port Type**		**Comments**
PCO1	MyMessagePortType		first comment
PCO2	MyProcedurePortType		second comment
Detailed Comments	detailed comments		

Maps to:

```
 1: module MyModule {
 2:  type component MyComponentType {
 3:  const float PI := 3.14;
 4:  var float x := PI * 2;
 5:  timer t1 := 15;
 6:  port MyMessagePortType PCO1;
 7:  port MyProcedurePortType PCO2;
 8:  } with {
 9:  display "comments := an example component type";
10:  display (PI) "comments := the ratio";
11:  display (x) "comments := double PI";
12:  display (t1) "comments := a 15 second timer";
13:  display (PCO1) "comments := first comment";
14:  display (PCO2) "comments := second comment";
15:  display "detailed comments := detailed comments";
16:  }
17: }
```

6.10 Constants

Constants			
Group	*[GroupReference]*		
Name	**Type**	**Value**	**Comments**
. *ConstIdentifier* | *ExtConstIdentifier* .	. *Type [ArrayDef]* .	. *ConstantExpression* | **external**	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 11: Constants Proforma

6.10.1 Mapping

The Constants proforma is mapped to a series of constant and external constant definition statements on the same group level. The **Detailed Comments** field is mapped to a display attribute within the *WithStatement* associated with the enclosing group or the module. The **Comments** fields are mapped to display attributes within the *WithStatement* associated with the respective constant definition. For an external constant the **Value** field is set to the keyword **`external`.**

```
 1: module TTCN3ModuleId "{"
 2:  const Type ConstIdentifier[ArrayDef] ":=" ConstantExpression with "{"
 3:  [CommentsAttribute ";"]
 4:  "}"
 5:  external const Type ConstIdentifier with "{"
 6:  [CommentsAttribute ";"]
 7:  "}"
 8: "}" with "{"
 9:  [ConstantsDetailedCommentsAttribute ";"]
10: "}"
```

EXAMPLE:

Constants			
Group	Constants1		
Name	**Type**	**Value**	**Comments**
TOTO	integer	external	defined somewhere else
SEL2	boolean	(5 + TOTO) < 10	TOTO limit reached
T1	integer[1..3]	{1,3,2}	
Detailed Comments	detailed comments		

Maps to:

```
 1: module MyModule {
 2:  group Constants1 {
 3:  external const integer TOTO with {
 4:  display "comments := defined somewhere else";
 5:  }
 6:  const boolean SEL2 := (5 + TOTO) < 10 with {
 7:  display "comments := TOTO limit reached";
 8:  }
 9:  const integer T1[1..3] := {1,3,2};
10:  } with {
11:  display "detailed comments := detailed comments";
12:  }
13: }
```

6.11 Signature

Signature Definition		
Name	*SignatureIdentifier*([*SignatureFormalParList*])	
Group	*[GroupReference]*	
Return Type	*[Type]* | **noblock**	
Comments	*[TabFreeText]*	
Exception Type		**Comments**
. *[ExceptionType]* .		. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*	

Figure 12: Signature Definition Proforma

6.11.1 Mapping

The Signature Definition proforma is mapped to a signature definition in the TTCN-3 core language. The **Comments** and **Detailed Comments** fields are mapped to display attributes within the corresponding *WithStatement*. The **Comments** fields of the exceptions table are mapped to display attributes qualified by the exception type in the corresponding *WithStatement*. Non-blocking procedures shall specify the keyword **noblock** as the return type.

```
 1: module TTCN3ModuleId "{"
 2:  signature SignatureIdentifier "(" [SignatureFormalParList] ")"
 3:  [return Type | noblock]
 4:  [exception "(" ExceptionTypeList ")"]
 5:  with "{"
 6:  [CommentsAttribute ";"]
 7:  [ExceptionCommentsAttribute ";"]
 8:  [DetailedCommentsAttribute ";"]
 9:  "}"
10: "}"
```

EXAMPLE:

Signature Definition		
Name	read(integer fields, inout charstring buf, integer nbyte)	
Group		
Return Type	integer	
Comments	reads from a file	
Exception Type		**Comments**
integer		error code
MyException		user defined
Detailed Comments	required: unistd.h	

Maps to:

```
 1: module MyModule {
 2:  signature read_syscall(in integer fields,
 3:  inout charstring buf,
 4:  in integer nbyte)
 5:  return integer
 6:  exception (integer)
 7:  with {
 8:  display "comments := reads from a file";
 9:  display (integer) "comments := error code of system call";
10:  display "detailed comments := required: unistd.h";
11:  }
12: }
```

6.12 Simple Templates

Simple Templates					
Group	*[GroupReference]*				
Name	**Type**	**Derived**	**Value**	**Encoding**	**Comments**
. *TemplateIdentifier* .	. *BaseTemplate* .	. *[DerivedDef]* .	. *TemplateBody* .	. *[TabFreeText]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*				

Figure 13: Simple Template Proforma

6.12.1 Mapping

The Simple Templates proforma is mapped to a series of simple template definition statements on the same group level. Simple template definitions are all template definitions that have a *SimpleSpec* or *ArrayValueOrAttrib* as the *TemplateBody*. The corresponding types are defined in a Simple Types, SequenceOf Type and Enumerated Type proforma.

The **Detailed Comments** field is mapped to a display attribute within the *WithStatement* associated with the enclosing group or the module. The **Comments** and **Encoding** fields are mapped to display and encode attributes qualified by the *TemplateIdentifier* within the *WithStatement* associated with the respective simple template definition statement.

```
1: module TTCN3ModuleId "{"
2:  template BaseTemplate[DerivedDef] := TemplateBody with "{"
3:  [EncodeAttribute ";"]
4:  [CommentsAttribute ";"]
5:  "}"
6: "}" with "{"
7:  [SimpleTemplatesDetailedCommentsAttribute ";"]
8: "}"
```

EXAMPLE:

Simple Templates					
Group	SimpleTemplates1				
Name	**Type**	**Derived**	**Value**	**Encoding**	**Comments**
MyTemplate1	MyType1		3	BER	foobar
MyTemplate11(integer index)	MyType1	MyTemplate1	3*index	PER	the current index
Detailed Comments	an example				

Maps to:

```
 1: module MyModule {
 2:  group SimpleTemplates {
 3:  template MyType1 MyTemplate1 with {
 4:  encode "BER";
 5:  display "comments := foobar";
 6:  }
 7:  template MyType1 MyTemplate11(integer index)
 8:  modifies MyTemplate1 := 3 * index
 9:  with {
10:  encode "PER";
11:  display "comments := the current index";
12:  }
13:  } with {
14:  display "simple templates detailed comments := an example";
15:  }
16: }
```

6.13 Structured Template

Structured Template			
Name	*TemplateIdentifier[(TemplateFormalParList)]*		
Group	*[GroupReference]*		
Type/Signature	*TypeIdentifier | SignatureIdentifier*		
Derived From	*[TemplateRef]*		
Encoding	*[TabFreeText]*		
Comments	*[TabFreeText]*		
Element Name	**Element Value**	**Element Encoding**	**Comments**
. *FieldReference* .	. *FieldValueOrAttrib* .	. *[TabFreeText]* .	. *[TabFreeText]* .
Detailed Comments	*[TabFreeText]*		

Figure 14: Structured Template Proforma

6.13.1 Mapping

The Structured Template proforma is mapped to a TTCN-3 structured template definition statement. Structured template definitions are all template definitions that have a *FieldSpecList* as the template body. The corresponding types are defined in a Structured Type proforma.

The **Comments** and **Detailed Comments** fields are mapped to display attributes within the *WithStatement* associated with the structured template definition. The **Encoding** field is mapped to an encoding attribute within the *WithStatement* associated with the structured template definition.

The **Comments** fields of the elements table are mapped to display attributes qualified by the field reference within the *WithStatement* associated with the structured template definition. The **Element Encoding** fields are mapped to encoding attributes qualified by the field reference within the *WithStatement* associated with the structured template definition.

```
 1: module TTCN3ModuleId "{"
 2:  template BaseTemplate [DerivedDef] ":=" TemplateBody with "{"
 3:  [EncodeAttribute ";"]
 4:  [CommentsAttribute ";"]
 5:  [FieldEncodeAttribute ";"]
 6:  [FieldCommentsAttribute ";"]
 7:  [DetailedCommentsAttribute ";"]
 8:  "}"
 9: "}"
```

EXAMPLE:

Structured Template			
Name	MyStructuredTemplate11(integer para1, boolean para2)		
Group			
Type/Signature	MyStructuredType		
Derived From	MyStructuredTemplate1		
Encoding	BER		
Comments	example structured template		
Element Name	**Element Value**	**Element Encoding**	**Comments**
field1	13		first field
field2	para2	PER	second field
field3	para1		third field
Detailed Comments	detailed comments		

Maps to:

```
 1: module MyModule {
 2:  template MyStructuredType MyStructuredTemplate11(integer para1,
 3:  boolean para2)
 4:  modifies MyStructuredTemplate1 := {
 5:  field1 := 13,
 6:  field2 := para2,
 7:  field3 := para1
 8:  } with {
 9:  encode "BER";
10:  display "comments := example structured template";
11:  display (field1) "comments := first field";
12:  encode (field2) "PER";
13:  display (field2) "comments := second field";
14:  display (field3) "comments := third field";
15:  display "detailed comments := detailed comments";
16:  }
17: }
```

6.14 Function

Function			
Name	*FunctionIdentifier*(*[FunctionFormalParList]*)		
Group	*[GroupReference]*		
Runs On	*[ComponentType]*		
Return Type	*[Type]*		
Comments	*[TabFreeText]*		
Local Def Name	**Type**	**Initial Value**	**Comments**
. *VarConstOrTimerIdentifier* .	. *TypeOrTimer* .	. *[Expression | ConstantExpression]*	. *[TabFreeText]* .
Behaviour			
. *FunctionStatement* | **external** .			
Detailed Comments	*[TabFreeText]*		

Figure 15: Function Proforma

6.14.1 Mapping

The Function proforma is mapped to a TTCN-3 function definition statement or external function definition. It is translated into three parts.

The first part consists of the header fields. The **Comments** and **Detailed Comments** fields are mapped to display attributes within a *WithStatement* associated with the function definition.

The second part consists of local constants, variables and timers defined in the function definition. These definitions can occur anywhere in the function body of the core language, but for the proforma they are separated from the rest of the function body and displayed in a separate table. The order of definitions shall be preserved, since the definitions can depend on each other. The **Type** column shall be set to the keyword **`timer`** for all timers and to the constant type preceded by the keyword **`const`** for all constants. The **Comments** fields are converted to display attributes qualified by the local identifier within the *WithStatement* associated with the function definition.

The third part consists of the function body of the TTCN-3 core language minus the local constants, variables and timers.

For an external function the behaviour only contains the keyword **`external`**.

```
 1: module TTCN3ModuleId "{"
 2:  function FunctionIdentifier "(" [FunctionFormalParList] ")"
 3:  [runs on ComponentType]
 4:  [return Type] "{"
 5:  var Type VarIdentifier [":=" Expression] ";"
 6:  timer TimerIdentifier [":=" Expression]";"
 7:  const Type ConstIdentifier ":=" ConstantExpression ";"
 8:  {FunctionStatement}
 9:  "}" with "{"
10:  [CommentsAttribute ";"]
11:  [VarConstOrTimerCommentsAttribute ";"]
12:  [DetailedCommentsAttribute ";"]
13:  "}"
14: "}"
```

EXAMPLE:

Function			
Name	MyFunction(integer para1)		
Group			
Runs On	MyComponentType		
Return Type	boolean		
Comments	example function definition		
Local Def Name	**Type**	**Initial Value**	**Comments**
MyLocalVar	boolean	false	local variable
MyLocalConst	const float	60	local constant
MyLocalTimer	timer	15 * MyLocalConst	local timer
Behaviour			

if (para1 == 21) {
 MyLocalVar := true;
}
if (MyLocalVar) {
 MyLocalTimer.start;
 MyLocalTimer.timeout;
}
return (MyLocalVar);			

Detailed Comments	detailed comments		

Maps to:

```
 1: module MyModule {
 2:  function MyFunction(in integer para1)
 3:  runs on MyComponentType
 4:  return boolean {
 5:  var boolean MyLocalVar := false;
 6:  const float MyLocalConst := 60;
 7:  timer MyLocalTimer := 15 * MyLocalConst;
 8:
 9:  if (para1 == 21) {
10:  MyLocalVar := true;
11:  }
12:  if (MyLocalVar) {
13:  MyLocalTimer.start;
14:  MyLocalTimer.timeout;
15:  }
16:  return (MyLocalVar);
17:  } with {
18:  display "comments := example function definition";
19:  display (MyLocalVar) "comments := local variable";
20:  display (MyLocalConst) "comments := local constant";
21:  display (MyLocalTimer) "comments := local timer";
22:  display "detailed comments := detailed comments";
23:  }
24: }
```

6.15 Altstep

Altstep			
Name	*AltstepIdentifier([AltstepFormalParList])*		
Group	*[GroupReference]*		
Purpose	*[TabFreeText]*		
Runs On	*[ComponentType]*		
Comments	*[TabFreeText]*		
Local Def Name	**Type**	**Initial Value**	**Comments**
. *VarConstOrTimerIdentifier* .	. *TypeOrTimer [ArrayDef]* .	. *[Expression | ConstantExpression]* .	. *[TabFreeText]* .
Behaviour			
. *AltGuardList* .			
Detailed Comments	*[TabFreeText]*		

Figure 16: Altstep Proforma

6.15.1 Mapping

The Altstep proforma is mapped to a TTCN-3 altstep definition statement. It is translated into three parts.

The first part consists of the header fields. The **Purpose**, **Comments** and **Detailed Comments** fields are mapped to display attributes within a *WithStatement* associated with the altstep definition.

The second part consists of local constants, variables and timers defined in the altstep definition. These definitions can occur anywhere in the altstep body of the core language, but for the proforma they are separated from the rest of the altstep body and displayed in a separate table. The order of definitions shall be preserved, since the definitions can depend on each other. The **Type** column shall be set to the keyword **`timer`** for all timers and to the constant type preceded by the keyword **`const`** for all constants. The **Comments** fields are converted to display attributes qualified by the local identifier within the *WithStatement* associated with the altstep definition.

The third part consists of the *AltGuardList* of the altstep of the TTCN-3 core language.

```
 1: module TTCN3ModuleId "{"
 2:  teststep AltstepIdentifier "(" [AltstepFormalParList] ")"
 3:  [runs on ComponentType] "{"
 4:  AltGuardList
 5:  "}" with "{"
 6:  [PurposeAttribute ";"]
 7:  [CommentsAttribute ";"]
 8:  [VarConstOrTimerCommentsAttribute ";"]
 9:  [DetailedCommentsAttribute ";"]
10:  "}"
11: "}"
```

EXAMPLE:

Altstep			
Name	MyAltstep(integer para1)		
Group			
Runs On	MyComponentType		
Purpose	to do something		
Comments	example altstep definition		
Local Def Name	**Type**	**Initial Value**	**Comments**
MyLocalVar	boolean	false	local variable
MyLocalConst	const float	60	local constant
MyLocalTimer	timer	15 * MyLocalConst	local timer
Behaviour			

[] PCO1.receive(MyTemplate(para1, CompVar) {
 verdict.set(inconc);
}
[] PCO2.receive {
 repeat;
}
[] CompTimer.timeout {
 verdict.set(fail);
 stop;
}			

Detailed Comments	detailed comments		

Maps to:

```
 1: module MyModule {
 2:  altstep MyTeststep(integer para1) runs on MyComponentType {
 3:  var boolean MyLocalVar := false;
 4:  const float MyLocalConst := 60;
 5:  timer MyLocalTimer := 15 * MyLocalConst;
 6:
 7:  [] PCO1.receive(MyTemplate(para1, CompVar)) {
 8:  verdict.set(inconc);
 9:  }
10:  [] PCO2.receive {
11:  repeat;
12:  }
13:  [] CompTimer.timeout {
14:  verdict.set(fail);
15:  stop;
16:  }
17:  } with {
18:  display "purpose := to do something";
19:  display "comments := example altstep definition";
20:  display "detailed comments := detailed comments";
21:  }
22: }
```

6.16 Testcase

Testcase			
Name	*TestcaseIdentifier*(*[TestcaseFormalParList]*)		
Group	*[GroupReference]*		
Purpose	*[TabFreeText]*		
System Interface	*[ComponentType]*		
MTC Type	*ComponentType*		
Comments	*[TabFreeText]*		
Local Def Name	**Type**	**Initial Value**	**Comments**
. *VarConstOrTimerIdentifier* .	. *TypeOrTimer* .	. *[Expression | ConstantExpression]*	. *[TabFreeText]* .
Behaviour			
. *FunctionStatement* .			
Detailed Comments	*[TabFreeText]*		

Figure 17: Testcase Proforma

6.16.1 Mapping

The Testcase proforma is mapped to a TTCN-3 testcase definition statement. It is translated into three parts.

The first part consists of the header fields. The **Purpose**, **Comments** and **Detailed Comments** fields are mapped to display attributes within a *WithStatement* associated with the test case definition.

The second part consists of local constants, variables and timers defined in the testcase definition. These definitions can occur anywhere in the testcase body of the core language, but for the proforma they are separated from the rest of the testcase body and displayed in a separate table. The order of the definitions shall be preserved, since the definitions can depend on each other. The **Type** column shall be set to the keyword **`timer`** for all timers and to the constant type preceded by the keyword **`const`** for all constants. The **Comments** fields are converted to display attributes qualified by the local identifier within the *WithStatement* associated with the testcase definition.

The third part consists of the testcase body of the TTCN-3 core language minus the local constants, variables and timers.

```
 1: module TTCN3ModuleId "{"
 2:  testcase TestcaseIdentifier [TestcaseFormalParList]
 3:  [runs on ComponentType]
 4:  [system ComponentType] "{"
 5:  var Type VarIdentifier [":=" Expression] ";"
 6:  timer TimerIdentifier [":=" Expression] ";"
 7:  const Type ConstIdentifier ":=" ConstantExpression;
 8:  {FunctionStatement}
 9:  "}" with "{"
10:  [CommentsAttribute ";"]
11:  [PurposeAttribute ";"]
12:  [VarConstOrTimerCommentsAttribute ";"]
13:  [DetailedCommentsAttribute ";"]
14:  "}"
15: "}"
```

EXAMPLE:

Testcase			
Name	MyTestcase(integer para1)		
Group			
Purpose	do something useful		
System Interface	MyComponentType		
MTC Type	MyComponentType		
Comments	example testcase definition		
Local Def Name	**Type**	**Initial Value**	**Comments**
MyLocalVar	boolean	false	local variable
MyLocalConst	const float	60	local constant
MyLocalTimer	timer	15 * MyLocalConst	local timer

Behaviour

```
default.activate { [expand] OtherwiseFail(); }; /* Default activation */
ISAP1.send(ICONreq {}); /* Inline template definition */
alt {
    [] MSAP2.receive(Medium_Connection_Request()) { /* use of a template */
        MSAP2.send(MDATreq Medium_Connection_Confirmation());
        alt {
            [] ISAP1.receive(ICONconf {} ); {
                ISAP1.send(Data_Request(TestSuitePar) );
                alt {
                    [] MSAP2.receive(Medium_Data_Transfer()) {
                        MSAP2.send(MDATreq cmi_synch1());
                        ISAP1.send(IDISreq {});
                    }
                    [] ISAP1.receive(IDISind {}) {
                        verdict.set(inconclusive);
                        stop();
                    }
                }
            }
            [] MSAP2.receive(MDATind_Connection_Request()) {
                verdict.set(inconclusive);
                stop();
            }
            [] ISAP1.receive(IDISind {}) {
                verdict.set(inconclusive);
                stop();
            }
        }
    }
    [] ISAP1.receive(IDISind {}) {
        verdict.set(inconclusive);
        stop();
    }
}
```

Detailed Comments	detailed comments

Maps to:

```
 1: module MyModule {
 2:     testcase MyTestcase(in integer para1)
 3:         runs on MyComponentType
 4:         system MyComponentType {
 5:         var boolean MyLocalVar := false;
 6:         const float MyLocalConst := 60;
 7:         timer MyLocalTimer := 15 * MyLocalConst;
 8:         var default MyDefault := activate(OtherwiseFail());
 9:
10:         ISAP1.send(ICONreq:{}); /* Inline template definition */
11:         alt {
12:             /* use of a template */
13:             [] MSAP2.receive(Medium_Connection_Request()) {
14:     MSAP2.send(MDATreq:Medium_Connection_Confirmation());
15:                 alt {
16:                     [] ISAP1.receive(ICONconf:{}) {
17:                         ISAP1.send(Data_Request(TestSuitePar));
18:                         alt {
19:                             [] MSAP2.receive(Medium_Data_Transfer()) {
20:                                 MSAP2.send(MDATreq:cmi_synch1());
21:                                 ISAP1.send(IDISreq:{});
22:                             }
23:                             [] ISAP1.receive(IDISind:{}) {
24:                                 verdict.set(inconc);
25:                                 stop;
26:                             }
27:                         }
28:                     }
29:                     [] MSAP2.receive(MDATind_Connection_Request()) {
30:                         verdict.set(inconc);
31:                         stop;
32:                     }
33:                     [] ISAP1.receive(IDISind:{}) {
34:                         verdict.set(inconc);
35:                         stop;
36:                     }
37:                 }
38:             }
39:             [] ISAP1.receive(IDISind:{}) {
40:                 verdict.set(inconc);
41:                 stop;
42:             }
43:         }
44:     } with {
45:         display "purpose := do something useful";
46:         display "comments := example testcase definition";
47:         display (MyLocalVar) "comments := local variable";
48:         display (MyLocalConst) "comments := local constant";
49:         display (MyLocalTimer) "comments := local timer";
50:         display "detailed comments := detailed comments";
51:     }
    }
```

7 BNF productions

1. TabFreeText ::= [ExtendedAlphaNum]

2. GroupReference ::= {GroupIdentifier "/"}+

3. EncRuleIdentifier ::= Identifier

4. CommentsAttribute :: = **display** """ "comments" ":=" TabFreeText """

5. DetailedCommentsAttribute :: = **display** """ "detailed comments" ":=" TabFreeText """

6. TTCN3ModuleId ::= ModuleIdentifier [DefinitiveIdentifier]

```
7.  ModuleAttributes ::= TabularPresentationFormatAttribute ";"
    ModuleVersionAttribute ";"
    ModuleDateAttribute ";"
    ModuleBaseStandardRefAttribute ";"
    ModuleTestStandardRefAttribute ";"
    ModulePICSRefAttribute ";"
    ModulePIXITRefAttribute ";"
    ModuleTestMethodAttribute ";"
    ModuleCommentsAttribute ";"
    ModuleDetailedCommentsAttribute ";"

8.  TabularPresentationFormatAttribute ::=
    display """ "presentation format := ETSI Tabular version" MajorVersion "." MinorVersion """

9.  MajorVersion ::= Number

10. MinorVersion ::= Number

11. ModuleVersionAttribute ::=
    display """ "module version" ":=" TabFreeText """

12. ModuleDateAttribute ::=
    display """ "module date" ":=" TabFreeText """

13. ModuleBaseStandardRefAttribute ::=
    display """ "module base standards ref" ":=" TabFreeText """

14. ModuleTestStandardRefAttribute ::=
    display """ "module test standards ref" ":=" TabFreeText """

15. ModulePICSRefAttribute ::=
    display """ "module pics ref" ":=" TabFreeText """

16. ModulePIXITRefAttribute ::=
    display """ "module pixit ref" ":=" TabFreeText """

17. ModuleTestMethodAttribute ::=
    display """ "module test method" ":=" TabFreeText """

18. ModuleCommentsAttribute ::=
    display """ "module comments" ":=" TabFreeText """

19. ModuleDetailedCommentsAttribute ::=
    display """ "module detailed comments" ":=" TabFreeText """

20. ModuleParPicsPixitRefAttribute ::=
    display "(" ModuleParIdentifier ")"
    """ "pics/pixit ref" ":=" TabFreeText """

21. ModuleParComments ::=
    display "(" ModuleParIdentifier ")"
    """ "comments" ":=" TabFreeText """

22. ImportsSourceRefAttribute ::=
    display """ "imports source ref" ":=" TabFreeText """

23. ImportsSourceDefinitionCommentsAttribute ::=
    display "(" ImportIdentifier ")"
    """ "comments" ":=" TabFreeText """

24. ImportSpecification ::= ( (Identifier | FullGroupIdentifier) | AllKeyword ) [ ExceptionsDef ]
    /* STATIC SEMANTIC: FullGroupIdentifier shall only be used for group imports. */

25. EncodeAttribute ::= encode """ TabFreeText """

26. SimpleTypesDetailedCommentsAttribute ::=
    display """ "simple types detailed comments" ":=" TabFreeText """

27. StructureType ::= record | union | set

28. FieldCommentsAttribute ::=
    display "(" FieldIdentifier ")" """ "comments" ":=" TabFreeText """

29. FieldEncodeAttribute ::=
    encode "(" FieldIdentifier ")" """ TabFreeText """
```

```
30. SequenceOfTypesDetailedCommentsAttribute ::=
    display """ "sequenceof types detailed comments" ":=" TabFreeText """

31. NamedValueCommentsAttribute ::=
    display "(" NamedValueIdentifier ")"
    """ "comments" ":=" TabFreeText """

32. TypeOrSignatureCommentsAttribute ::=
    display "(" TypeOrSignatureIdentifier ")"
    """ "comments" ":=" TabFreeText """

33. PortCommentsAttribute ::=
    display "(" PortIdentifier ")"
    """ "comments" ":=" TabFreeText """

34. ConstantsDetailedCommentsAttribute ::=
    display """ "simple types detailed comments" ":=" TabFreeText """

35. ExceptionCommentsAttribute ::=
    display "(" Type ")"
    """ "comments" ":=" TabFreeText """

36. VarConstOrTimerCommentsAttribute ::=
    display "(" VarConstOrTimerIdentifier ")"
    """ "comments" ":=" TabFreeText """

37. PurposeAttribute ::= display """ "purpose" ":=" TabFreeText """

38. SimpleTemplatesDetailedCommentsAttribute ::= display """ "simple templates detailed comments"
    ":=" TabFreeText """
```

History

Document history		
V1.1.1	March 2001	Publication
V1.1.2	June 2001	Publication
V2.2.1	February 2003	Publication
V3.1.1	June 2005	Publication
V3.2.1	February 2007	Publication