INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Z.141

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(07/2001)

SERIES Z: LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

Formal description techniques (FDT)

The Tree and Tabular combined notation version 3 (TTCN-3): Tabular presentation format

ITU-T Recommendation Z.141

(Formerly CCITT Recommendation)

ITU-T Z-SERIES RECOMMENDATIONS

LANGUAGES AND GENERAL SOFTWARE ASPECTS FOR TELECOMMUNICATION SYSTEMS

FORMAL DESCRIPTION TECHNIQUES (FDT)	
Specification and Description Language (SDL)	Z.100–Z.109
Application of Formal Description Techniques	Z.110–Z.119
Message Sequence Chart	Z.120–Z.129
PROGRAMMING LANGUAGES	
CHILL: The ITU-T programming language	Z.200–Z.209
MAN-MACHINE LANGUAGE	
General principles	Z.300–Z.309
Basic syntax and dialogue procedures	Z.310–Z.319
Extended MML for visual display terminals	Z.320–Z.329
Specification of the man-machine interface	Z.330–Z.399
QUALITY OF TELECOMMUNICATION SOFTWARE	Z.400–Z.499
METHODS FOR VALIDATION AND TESTING	Z.500–Z.599

For further details, please refer to the list of ITU-T Recommendations.

ITU-T Recommendation Z.141

The Tree and Tabular combined notation version 3 (TTCN-3): Tabular presentation format

Summary

This Recommendation defines the Tabular presentation format of the Tree and Tabular combined notation version 3 (TTCN-3).

Source

ITU-T Recommendation Z.141 was prepared by ITU-T Study Group 10 (2001-2004) and approved under the WTSA Resolution 1 procedure on 22 July 2001.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2001

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from ITU.

CONTENTS

ITU-T Recommendation Z.141

The Tree and Tabular combined notation version 3 (TTCN-3): Tabular presentation format

1 Scope

This Recommendation defines the tabular presentation format of TTCN Version 3 (or TTCN-3). It is based on the TTCN-3 core language defined in ITU-T Z.140 [1].

The specification of other formats is outside the scope of this Recommendation.

2 References

The following ITU-T Recommendations and other references contain provisions, which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

[1] ITU-T Z.140 (2001), *The Tree and Tabular Combined Notation version 3 (TTCN-3): Core language*.

3 Abbreviations

This Recommendation uses the following abbreviations:

ASN.1 Abstract Syntax Notation One

ATS Abstract Test Suite

BNF Backus-Naur Form

IUT Implementation Under Test

MTC Master Test Component

PICS Protocol Implementation Conformance Statement

PIXIT Protocol Implementation eXtra Information for Testing

TTCN Tree and Tabular Combined Notation

4 Introduction

The tabular presentation format 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 ITU-T Z.140 [1] and provides a full text-based syntax, static semantics and operational semantics as well as defining the use of the language with ASN.1. 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.

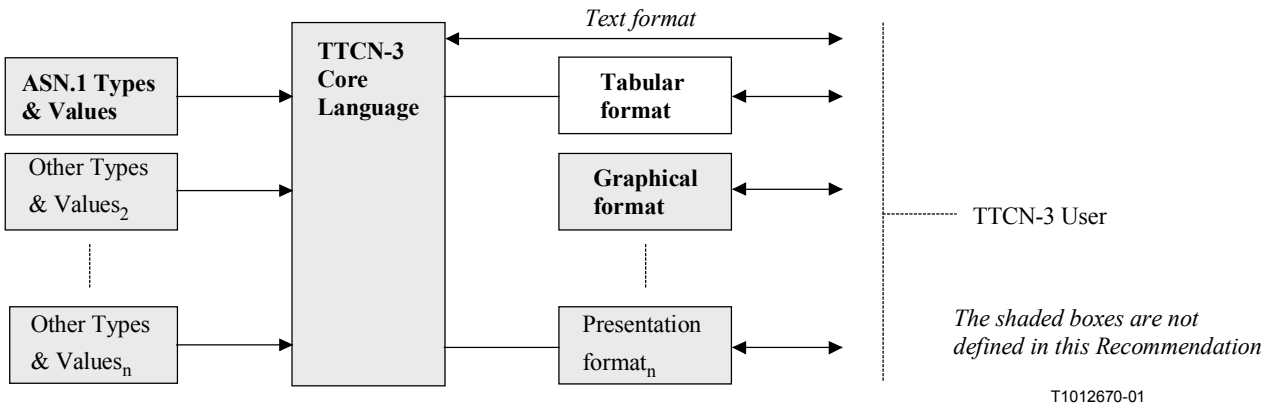

Figure 1/Z.141 – User's view of the core language and the various presentation formats

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.

This Recommendation 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/Z.141 – 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 clause 8 of this Recommendation. Productions that are not defined in clause 8 can be found in ITU-T Z.140 [1].

5.2 Proformas

a) Bold text (**like this**) shall appear verbatim in each actual table in a TTCN test suite.

b) Text in italics (*like this*) shall not appear verbatim in a TTCN test suite. This font is used to indicate that actual text shall be substituted for the italicised symbol. Syntax requirements for the actual text can be found either following the definition of the proforma, either in the TTCN Core language BNF.

c) Greyed background means that the field (or row, or column) is optional.

6 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.

These transformations fall into two classes. The first class directly converts from a tabular element to a core language construct with the same meaning. The second class converts a tabular element into 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 some 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 **`with`** statement.

These **`with`** statements have a common format which takes the form:

```
with display "<ProfomaIdentifier> { <ElementIdentifier1> := FreeText;
                                    <ElementIdentifier2> := FreeText;
                                    <ElementIdentifierN> := FreeText }"
```

The *<ProfomaIdentifier>* is the name of the associated table in the tabular format and the *<ElementIdentifier>* is the name of the field or element in that table which is defined in the statement. The value of the field or element is specified in free text after the ':=' character. The precise BNF definition for the interpretation of the display string is defined in this Recommendation.

The syntax and semantics specified in this Recommendation 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:

```
module ModuleName()
{
}
with {   display "PresentationFormatIdentifer" }

PresentationFormatIdentifer ::= PresentationKeyword FormatKeyword
AssignmentChar "ETSI Tabular     v1.0"
PresentationKeyword ::= presentation
FormatKeyword ::= format
```

NOTE – All **`with`** statements associated with a given proforma should be grouped together in a contiguous list.

7 Proformas

7.1 Test suite control

Test Suite Control				
Name	:	*TTCN3ModuleId*		
Version	:	*VersionIdentifier*		
Date	:	*FreeText*		
Base Standard Ref	:	*FreeText*		
Test Standard Ref	:	*FreeText*		
PICS Ref	:	*FreeText*		
PIXIT Ref	:	*FreeText*		
Test Method(s)	:	*FreeText*		
Detailed Comments	:	***[FreeText]***		
Name		**Type**	**Initial Value**	**Comments**
. *[VarConstOrTimerIdentifier]* . .		. *[TypeOrTimer]* . .	. *[ConstantExpression]* . .	. *[FreeText]* . .
Behaviour				**Comments**
. . *ModuleControlBody*				. . *[FreeText]* . .
Detailed Comments:	*[FreeText]*			

Figure 2/Z.141 – Test suite control proforma

7.1.1 Mapping

The Test suite Control proforma is translated into two parts. The first part is the control part of the TTCN-3 core language module. The comments fields are converted to a **`with`** statement associated with the control definition in the core language. The header information in the Test Suite Control proforma is converted to a **`with`** statement associated with the overall TTCN-3 module.

```
module MyModule()
{
     control{
          var Type VarIdentifier [ ":=" ConstantExpression]
          timer TimerIdentifier  [ ":=" ConstantExpression]
          const Type ConstIdentifier ":=" ConstantExpression

          ModuleControlBody

     } with display "ControlProforma"
}
with display "ReferenceProforma"
```

Example:

Test Suite Control		
Name	:	*MyATS*
Version	:	*1.1*
Date	:	*23 May 1999*
Base Standards Ref	:	
Test Standards Ref	:	
PICS Ref	:	
PIXIT Ref	:	
Test Method(s)	:	*local*
Detailed Comments	:	*ATS written by STF 133*

Name	Type	Initial Value	Comments
X	**integer**	7	
T1	**timer**	15 min	

Behaviour	Comments
/* group1/ */ /* group1_1/ */ **execute**(test1); **execute**(test2); /* group1_2/ */ **execute**(test3); **execute**(test4); /* group2/ */ **execute**(test5);	basic tests check capability 1

Detailed Comments:

Maps to:

```
module MyATS()
{

     control{
          var integer x := 7;
          timer T1 := 15 min;

          /* group1/ */
               /* group1_1/ */
                    execute( test1);
                    execute( test2);
               /* group1_2/ */
                    execute( test3);
                    execute( test4);
               /* group2/ */
               execute( test5);
     } with display " Control { extracomments := "basic tests
                                                 check capability 1"}";

} with
     {
display "presentation format                    :=  "ETSI Tabular v1.0";
display "reference {    version                 :=     "1.1";
                        date                    :=     "23 May 1999";
                        testmethod              :=     "local";
                        detailedcomments        :=     "ATS written by STF 133" }"
     }
```

7.2 Test suite parameters

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

Figure 3/Z.141 – Test suite parameters proforma

7.2.1 Mapping

All entries in the Parameter table are mapped into the parameter list of the associated TTCN-3 module. The PICS/PIXITref and comments are mapped to the **`with`** statements of the TTCN-3 module.

```
module MyModule(ModuleType ModuleParIdentifier)
{
}
with display "ParametersProforma";
```

Example:

Test Suite Parameters				
Group : PICS/				
Name	**Type**	**Initial Value**	**PICS/PIXIT Ref**	**Comments**
CAP_1	**boolean**	**true**	A.1.3	Option 1 implemented by IUT
Detailed Comments:				

Maps to:

```
module MyModule(boolean CAP_1 := true)
{
}
with display "parameters {
              group := PICS/;
              pics pixet := { CAP_1 := "A.1.3"}
              comments := { CAP1 := "Option 1 implemented by IUT"}
              }"
```

7.3 Module imports

Imports		
Source Name	:	*ModuleIdentifier [DefinitiveIdentifier]] ModuleIdentifier [DefinitiveIdentifier]*
Source Language	:	*[LanguageSpec]*
Group	:	*[GroupReference]*
Source Ref	:	*[FreeText]*
Encoding	:	*[FreeText]*
Comments	:	*[FreeText]*

Type	Name	NR	Comments
.	.	.	.
.	.	.	.
ImportType	*ImportIdentifier*	*Mark*	*[FreeText]*
.	.	.	.
.	.	.	.

Detailed Comments: *[FreeText]*

Figure 4/Z.141 – Imports proforma

7.3.1 Mapping

The imports proforma is mapped to group of import statements in the TTCN-3 core language. The source name, import type, import name and recursion tabular elements are directly used in the corresponding core language import statement. The group is named **`Imports`** with a unique number appended at the end of the identifier when necessary to make the group name unique. All other fields are translated into a **`with`** statement associated with the enclosing group.

```
module MyModule()
{
     group Imports1 {
          import ImportType  ImportIdentifier from
ModuleIdentifier[DefinitiveIdentifier] [language LanguageIdentifer];
}
with {    display "ImportsProforma";
          encode "FreeText" }
```

Example:

Imports		
Source Name	:	ModuleA
Source Ref	:	EN 800 900 version 2
Encoding	:	
Comments	:	importing declarations from an existing ATS

Type	Name	NR	Comments
all constant type group	 MyType AtoU_CTR	*	(1)

Detailed Comments: (1) Tick indicates: import recursively what is needed for MyType definition

Maps to:

```
module MyModule()
{
     group Imports1 {
          import all constant  from ModuleA;
          import type  MyType from ModuleA;
          import group  AtoU_CTR from ModuleA;
     }
     with display "imports { source    := "EN 800 900 version 2";
                            comments := "importing declarations from an existing
                            ATS";
                            extracomments := "(1)";
                            detailedcomments := "(1) asterisk indicates: import
                                                recursively what is needed for
                                                MyType definition"}"
}
```

7.4 Encoding

Encoding Definitions			
Group : *[GroupReference]*			
Name	**Reference**	**Default**	**Comments**
.	.	.	.
.	.	.	.
EncodingRuleIdentifier	*FreeText*	*[BooleanExpression]*	*[FreeText]*
.	.	.	.
.	.	.	.
Detailed Comments: *[FreeText]*			

Figure 5/Z.141 – Encoding definitions proforma

7.4.1 Mapping

The encoding proforma is mapped to a series of statements in the **with** statement associated with the TTCN-3 core module. All table elements are mapped to display statements. In addition one encoding statement is added to the **with** statement for the encoding rule whose default value evaluates to **true**.

```
module MyModule()
{
}
with {
     display " EncodingProforma ";
     encode "EncodingRuleIdentifier"
}
```

Example:

Encoding Definitions			
Encoding Rule Name	**Reference**	**Default**	**Comments**
BER	ISO/IEC 8825-1: 1993	TRUE	Basic Encoding Rules
PER	ISO/IEC 8825-1: 1993		Packed Encoding Rules
DER	ISO/IEC 8825-1: 1993		Distinguished Encoding Rules
Detailed Comments:			

Maps to:

```
module MyModule()
{
}
with {
display "encoding {     reference := {    BER := "ISO/IEC 8825-1: 1993",
                                          PER := "ISO/IEC 8825-1: 1993",
                                          DER := "ISO/IEC 8825-1: 1993"};
                        default   := {    BER := TRUE};
                        comments  := {    BER := "Basic Encoding Rules",
                                          PER := "Packed Encoding Rules",
                                          DER := "Distinguished Encoding Rules"}";
encode "BER" }
```

7.5 Simple types

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

Figure 6/Z.141 – Simple types proforma

7.5.1 Mapping

The simple types proforma is mapped to a TTCN-3 group containing a series of type definition statements. The group reference and detailed comments are mapped to display statements inside the **with** statement associated with the group. The encoding and comment fields are mapped to statements with the **with** statement associated with the separate type definitions.

The group will be named $SimpleTypes_n$ where 'n' is an integer number used to distinguish more than one simple type group.

```
module MyModule()
{
     group SimpleTypes1 {
          type Type  SubTypeIdentifier SubTypeSpec
     }
     with {
          encode (SubTypeIdentifier) "FreeText";
          display "SimpleTypesProforma ";
 }
}
```

Example:

Simple Types			
Name	**Definition**	**Encoding**	**Comments**
EQ_NUMBER	**integer** (1 .. 20)		
Detailed Comments:			

Maps to:

```
module MyModule()
{
     group SimpleTypes1 {
          type integer EQ_NUMBER (1..20)
     }
     with display "simpletypes {}";
}
```

7.6 Structured types

Structured Type			
Name	:	*StructTypeIdentifier*	
Group	:	*[GroupReference]*	
Structure	:	*StructureType*	
Encoding	:	*[FreeText]*	
Comments	:	*[FreeText]*	
Element Name	**Type Definition**	**Field Encoding**	**Comments**
. . *StructFieldIdentifier* . .	. . *Type [SubTypeSpec]* *[OptionalKeyword]* . .	. . *[FreeText]* . .	. . *[FreeText]* . .
Detailed Comments:	*[FreeText]*		

Figure 7/Z.141 – Structured type proforma

7.6.1 Mapping

The structured type proforma is mapped to a type definition statement in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **with** statement.

```
module MyModule()
{
          type StructureType  StructTypeIdentifier
          {
               Type FieldIdentifier [ SubtypeSpec][ OptionalKeyword]
          }
          with {
               display "StructuredTypeProforma";
               encode "FreeText";
               encode (StructFieldIdentifier)"FreeText";
     }

}
```

Example:

Structured Type				
Name	:	*MaleMind*		
Group	:			
Structure	:	*record*		
Encoding	:			
Comments	:			
Element Name		**Type Definition**	**Field Encoding**	**Comments**
Car Money Football		**integer** **integer** octetstring		
Detailed Comments:				

Maps to:

```
module MyModule()
{
          type record MaleMind
          {
               integer        Car,
               integer        Money,
               octetstring    Football
          }
          with display "structuredtype {   comments := "";
                                           comments := {}
                                           detailedcomments := ""}
}
```

7.7 Port types

Port Type			
Name	:	*PortTypeIdentifier*	
Group	:	*[GroupReference]*	
Communication Model	:	*PortModelType*	
Comments	:	*[FreeText]*	
Type Definition			**Comments**
PortTypeDef . ..			. . *[FreeText]* . .
Detailed Comments:		*[FreeText]*	

Figure 8/Z.141 – Port type proforma

7.7.1 Mapping

The port type proforma is mapped to a port type definition in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **with** statement.

```
module MyModule()
{
          type port   PortTypeIdentifier PortModelType
          {
               PortTypeDef
          }
          with  display "PortTypeProforma";
   }
```

Example:

Port Type		
Name	**:**	*MyPortType*
Group	**:**	
Communication Model	**:**	*message*
Comments	**:**	
Type Definition		**Comments**
in MsgType1, MsgType2; **out** MsgType3;		
Detailed Comments:		

Maps to:

```
module MyModule()
{
          type port MyPortType message
          {
               in MsgType1, MsgType2;
               out MsgType3;
          }
          with display "porttype {}";
}
```

7.8 Component types

Component Type			
Name	**:**	*ComponentTypeIdentifier*	
Group	**:**	*[GroupReference]*	
Comments	**:**	*[FreeText]*	
Name	**Type**	**Initial Value**	**Comments**
. *[VarConstOrTimerIdentifier]* . .	. *[TypeOrTimer]* . .	. *[ConstantExpression]* . .	. *[FreeText]* . .
Port Definitions			**Comments**
PortList . .			*[FreeText].* . . . .
Detailed Comments:	*[FreeText]*		

Figure 9/Z.141 – Component type proforma

7.8.1 Mapping

The component type proforma is mapped to a component type definition in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **with** statement.

```
module MyModule()
{
          type component ComponentTypeIdentifier
          {
               var Type VarIdentifier [ ":=" ConstantExpression]
               timer TimerIdentifier  [ ":=" ConstantExpression]
               const Type ConstIdentifier ":=" ConstantExpression
               PortList
          }
          with  display "ComponentTypeProforma";
}
```

Example:

Component Type			
Name : MyComponentType			
Group :			
Comments :			
Name	**Type**	**Initial Value**	**Comments**
x	**integer**	7	
T1	**timer**	15 min	
Port Definitions			**Comments**
MyMessagePortType PCO1, PCO2; MyOtherPortType PCO3, PCO4;			
Detailed Comments:			

Maps to:

```
module MyModule()
{
          type component MyComponentType
          {
               var integer x := 7;
               timer T1 := 15 min;

               MyMessagePortType PCO1, PCO2;
               MyOtherPortType PCO3, PCO4;
          }
          with display "componenttype {}";
}
```

7.9 Constants

Constants			
Group : *[GroupReference]*			
Name	**Type**	**Value**	**Comments**
. *ConstIdentifier* .	. *Type* .	. *ConstantExpression* .	. *[FreeText]* .
Detailed Comments: *[FreeText]*			

Figure 10/Z.141 – Constants proforma

7.9.1 Mapping

The constants proforma is mapped to a group containing constant declarations in the TTCN-3 language. The group reference and comments are mapped to display statements within the associated **with** statement. The group is named $Constants_n$ where 'n' is a unique integer number appended to the end of the identifier.

```
module MyModule()
{
     group Constants1 {
          const Type  ConstIdentifier := ConstantExpression
     }
     with display "ConstantsProforma";
```

Example:

Constants			
Group : Misc			
Name	**Type**	**Value**	**Comments**
sel2	**boolean**	(5 + TOTO) < 10	TOTO is a constant
T1	**integer**	15	
Detailed Comments:			

Maps to:

```
module MyModule()
{
          group Constants1 {
               const boolean sel2 := (5 + TOTO) < 10;
               const integer T1 := 15;
          }
          with display "constants {    group := "Misc";
                                       comments := {sel2 ::= "TOTO is a
                                       constant"}}";
}
```

7.10 Signature

Signature Definition		
Name	:	*SignatureIdentifier&ParList*
Group	:	*[GroupReference]*
Return Type	:	*[Type]*
Comments	:	*[FreeText]*
Exception List		**Comments**
ExceptionTypeList . ..		. . *[FreeText]* . .
Detailed Comments:	*[FreeText]*	

Figure 11/Z.141 – Signature definition proforma

7.10.1 Mapping

The signature proforma is mapped to a signature definition in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **`with`** statement.

```
signature SignatureIdentifier&ParList [return Type]
{
     ExceptionTypeList
}
with display "SignatureProforma";
```

Example:

Signature Definition		
Name	:	MySignature(in integer Par1, out float Par2)
Group	:	
Return Type	:	boolean
Comments	:	
Exception List		**Comments**
integer, boolean, MyType		
Detailed Comments:		

Maps to:

```
signature MySignature( in interger Par1, out float Par2) return boolean
{
          integer, boolean, MyType
}
with display "signature {}";
```

7.11 Simple templates

Simple Templates				
Group :	*[GroupReference]*			
Name	**Type**	***Value***	**Encoding**	**Comments**
. *TemplateIdentifier* .	. *SimpleType* .	. *SingleValueOrAttrib*	. *[FreeText]* .	. *[FreeText]* .
Detailed Comments:	*[FreeText]*			

Figure 12/Z.141 – Simple templates proforma

7.11.1 Mapping

The simple templates proforma is mapped to a TTCN-3 group containing a series of template definition statements. The group reference and detailed comments are mapped to display statements inside the **with** statement associated with the group. The encoding and comment fields are mapped to statements with the **with** statement associated with the separate template definitions.

The group will be named $SimpleTemplates_n$ where 'n' is an integer number used to distinguish more than one simple templates groups.

```
module MyModule()
{
     group SimpleTemplates1 {
          template SimpleType  TemplateIdentifier := SingleVlaueOrAttrib
     }
     with {
          encode (TemplateIdentifier) "FreeText";
          display "SimpleTemplatesProforma ";
 }
}
```

Example:

Simple Templates				
Name	**Type**	***Value***	**Encoding**	**Comments**
AgeField	**integer**	?		
Detailed Comments:				

Maps to:

```
module MyModule()
{
     group SimpleTemplates1 {
          template integer AgeField := ?;
     }
     with display "simpletemplates {}";
}
```

7.12 Structured template

Structured Template				
Name	**:**	*TemplateIdentifier&ParList*		
Group	**:**	*[GroupReference]*		
Type	**:**	*TemplateStructIdentifier*		
Encoding	**:**	*[FreeText]*		
Comments	**:**	*[FreeText]*		
Element Name		**Element Value**	**Element Encoding**	**Comments**
. . *FieldReference* . .		. . *FieldValueOrAttrib* .	. . . *[FreeText]*	. . *[FreeText]* . .
Detailed Comments	**:**	*[FreeText]*		

Figure 13/Z.141 – Structured template proforma

7.12.1 Mapping

The structured template proforma is mapped to a template definition statement in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **`with`** statement.

```
template [Type | Signature] TemplateIdentifier&ParList [modifies TemplateRef] :=
{
          FieldReference ":=" FieldValueOrAttrib
}
with {    display "StructutedTemplateProforma";
          encode "FreeText";
          encode (FieldReference)"FreeText";
     }
```

Example:

Structured Template				
Name	**:**	Setup01		
Group	**:**			
Type	**:**	SetupMsgType		
Derivation Path	**:**			
Encoding	**:**			
Comments	**:**			
Element Name		**Element Value**	**Element Encoding**	**Comments**
MsgId		34		
CrLength		1		
CrValue		42		
IE1		?		
IE2		?		
Detailed Comments:				

Maps to:

```
template SetupMsgType Setup01 :=
{
         MsgId :=      34,
         CrLength :=   1,
         CrValue   :=  42,
         IE1 :=        ?,
         IE2 :=        ?
}
with display "structuredtemplate {}";
```

7.13 Function

Function			
Name	*:*	*FunctionIdentifier&ParList*	
Group	*:*	*[GroupReference]*	
Runs on	*:*	*[ComponentType]*	
Return Type	*:*	*[Type]*	
Comments	*:*	*[FreeText]*	
Name	**Type**	**Initial Value**	**Comments**
. *[VarConstOrTimerIdentifier]* . .	. *[TypeOrTimer]* . .	. *[ConstantExpression]* . .	. *[FreeText]* . .
Function Definition			Comments
TabularBehaviour . .			*[FreeText].* . . .
Detailed Comments: *[FreeText]*			

Figure 14/Z.141 – Function Proforma

7.13.1 Mapping

The function proforma is mapped to a function or external function definition in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **`with`** statement. The keyword **`external`** before the function name denotes that the function should be mapped to an external function.

```
[external] function FunctionIdentifier&ParList [return Type]
[Runs On ComponentType]
{
     var Type VarIdentifier [:= ConstantExpression] ;
     timer TimerIdentifier [:= ConstantExpression] ;
     const  Type ConstIdentifier  := ConstantExpression

     TabularBehaviour
}
with display "FunctionProforma";
```

Example:

Function			
Name : MyFunction(**in integer** Par1)			
Group :			
Runs on :			
Return Type : **boolean**			
Comments :			
Name	**Type**	**Initial Value**	**Comments**
MyLocalVar	**boolean**	**false**	
T1	**timer**	15 min	
Function Definition			**Comments**

if(Par1 = 21) {
MyLocalVar := **true**;
}
if(MyLocalVar) {
T1.**start**;
T1.**timeout**;
}
return(MyLocalVar);			

Detailed Comments:			

Maps to:

```
function MyFunction( in integer Par1) return boolean
{
        var boolean MyLocalVar := false;
        timer T1 := 15 min;

        if(  Par1 = 21 ) {
             MyLocalVar := true;
        }
        if(  MyLocalVar) {
             T1.start;
             T1.timeout;
        }
        return( MyLocalVar);
}
with display "function";
```

7.14 Defaults

Default Definition		
Name	:	*NamedAltIdentifier&ParList*
Group	:	*[GroupReference]*
Purpose	:	*[FreeText]*
Comments	:	*[FreeText]*
Behaviour		**Comments**
AltGuardList . ..		*[FreeText].* . . . .
Detailed Comments:	*[FreeText]*	

Figure 15/Z.141 – Default Definition Proforma

7.14.1 Mapping

The default proforma is mapped to a named alt definition in the TTCN-3 module. The group and comment fields are mapped to display statements inside the **`with`** statement associated with the definition.

```
named alt NamedAltIdentifier&ParList{
          AltGuardList
}
with display "default {purpose := "";}";
```

Example:

Default Definition		
Name	:	Default1
Group	:	
Purpose	:	
Comments	:	
Behaviour		**Comments**
[] PCO2.**receive**(DL_EST_IN); PCO2.**send**(DL_EST_CO); [] PCO2.**receive**(DL_EST_CO); // do nothing		
Detailed Comments:		

Maps to:

```
named alt Default1{
     []   PCO2.receive( DL_EST_IN){
          PCO2.send( DL_EST_CO)}
     []   PCO2.receive( DL_EST_CO);
}
with display "default{ purpose := "";}";
```

7.15 Named Alternative

Named Alternative Definition		
Name	:	*NamedAltIdentifier&ParList*
Group	:	*[GroupReference]*
Purpose	:	*[FreeText]*
Comments	:	*[FreeText]*
Behaviour		**Comments**
AltGuardList . ..		. . . . .
Detailed Comments:	*[FreeText]*	

Figure 16/Z.141 – Named Alternative Definition Proforma

7.15.1 Mapping

The named alternative proforma is mapped to a named alt definition in the TTCN-3 module. The group and comment fields are mapped to display statements inside the **with** statement associated with the definition.

```
named alt NamedAltIdentifier&ParList{
          AltGuardList
}
with display "NamedAltProforma ";
```

Example:

Named Alternative Definition		
Name	:	TS01
Group	:	
Purpose	:	
Comments	:	
Behaviour		**Comments**
[] PCO2.**receive**(DL_EST_IN); PCO2.**send**(DL_EST_CO); [] PCO2.**receive**(DL_EST_CO); // do nothing		
Detailed Comments:		

Maps to:

```
named alt Default1{
     []   PCO2.receive( DL_EST_IN){
          PCO2.send( DL_EST_CO)}
     []   PCO2.receive( DL_EST_CO);
}
with display "namedalt{ purpose := ""; }";
}
```

7.16 Testcase

Test Case Definition				
Name	**:**	*TestcaseIdentifier&ParList*		
Group	**:**	*[GroupReference]*		
Purpose	**:**	*[FreeText]*		
System Interface	**:**	*[ComponentType]*		
MTC Type	**:**	*ComponentType*		
Comments	**:**	*[FreeText]*		
Name		**Type**	**Initial Value**	**Comments**
. *[VarConstOrTimerIdentifier]* . .		. *[TypeOrTimer]* . .	. *[ConstantExpression]* . .	. *[FreeText]* . .
Behaviour				**Comments**
TabularBehaviour . ..				. . . . .
Detailed Comments:	*[FreeText]*			

Figure 17/Z.141 – Test Case Definition Proforma

7.16.1 Mapping

The test case proforma is mapped to a Testcase definition in TTCN-3, with the group and comment fields mapped to display statements in the corresponding **`with`** statement. The parameter list may only contain test suite variables.

```
testcase TestcaseIdentifier&ParList
runs on ComponentType[system ComponentType]
{
         var Type VarIdentifier [":=" ConstantExpression] ;
         timer TimerIdentifier [":=" ConstantExpression] ;
         const Type ConstIdentifier ":=" ConstantExpression ;

         TabularBehaviour

}
with display "TestcaseProforma";
```

Example:

Test Case Definition			
Name : MyTestcase			
Group :			
Purpose : First Example Testcase			
System Interface :			
MTC Type : MyComponentType			
Comments :			
Name	**Type**	**Initial Value**	**Comments**
MyLocalVar	**integer**	0	
TimerT1	**timer**	15 min	
Behaviour			**Comments**

```
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:

Maps to:

```
testcase MyTestcase
runs on MyComponentType
{
          var integer MyLocalVar:= 0;
          timer T1 := 15 min;

          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( MDATindConnection_Request()); {
                              verdict.set(inconclusive);
                              stop();
                         }
                    []   ISAP1.receive( IDISind {} ); {
                              verdict.set(inconclusive);
                              stop();
                         }
               };
          }
          []   ISAP1.receive( IDISind {} ); {
               verdict.set(inconclusive);
               stop();
          }
}
}
with display "testcase { purpose "";}";
}
```

8 Tabular PresentationFormat BNF

```
TabularPresentationFormat ::=    ReferenceProforma |
                                 ParametersProforma |
                                 ControlProforma |
                                 ImportsProforma |
                                 EncodingProforma |
                                 SimpleTypesProforma |
                                 StructuredTypesProforma |
                                 PortTypeProforma |
                                 ComponentTypeProforma |
                                 ConstantsProforma |
                                 SignatureProforma |
                                 SimpleTemplatesProforma |
```

```
                                StructuredTemplatesProforma |
                                FunctionProforma |
                                DefaultsProforma |
                                TeststepProforma |
                                TestcaseProforma
```

8.1 ReferenceProforma

```
ReferenceProforma ::=   ReferenceKeyword
                        BeginChar
                        ReferenceFieldList
                        EndChar

ReferenceFieldList ::= VersionKeyword AssignmentChar VersionIdentifier  SemiColon
                       DateKeyword    AssignmentChar FreeText           SemiColon
                       BaseKeyword    AssignmentChar FreeText           SemiColon
                       TestKeyword    AssignmentChar FreeText           SemiColon
                       PicsKeyword    AssignmentChar FreeText           SemiColon
                       PixitKeyword   AssignmentChar FreeText           SemiColon
                       MethodKeyword  AssignmentChar FreeText           SemiColon
                       [DetailedComments]

VersionIdentifier ::= Number { "." Number }

VersionKeyword ::=      version
DateKeyword ::=         date
BaseKeyword ::=         basestandard
TestKeyword ::=         teststandard
PicsKeyword ::=         pics
PixitKeyword ::=        pixit
MethodKeyword ::=       testmethod
DCommentsKeyword ::=    detailedcomments

DetailedComments ::= DCommentsKeyword     AssignmentChar  FreeText
```

8.2 ParametersProforma

```
ParametersProforma ::=  ParametersKeyword BeginChar
                        ParametersFieldList
                        EndChar

ParameterKeyword ::= parameters

ParameterFieldList ::=  [GroupDef]
                        PICSRefList
                        [ParameterCommentsList]
                        [DetailedComments]

GroupDef ::= GroupKeyword             AssignmentChar  GroupReference    SemiColon
GroupKeyword ::= group
CommentListKeyword ::= commentlist

PICSRefList ::= PicsKeyword PixitKeyword  AssignmentChar   BeginChar
                [PicsRef {"," PicsRef}]
                EndChar [SemiColon]

PicsRef ::= ModuleParIdentifier AssignmentChar FreeText

ParameterCommentsList ::=   CommentListKeyword       AssignmentChar    BeginChar
                            [ParComment {"," ParComment}]
                            EndChar [SemiColon]

ParComment ::= ModuleParIdentifier AssignmentChar FreeText
```

8.3 ControlProforma

```
ControlProforma ::= ControlKeyword BeginChar ControlFieldList EndChar

ControlFieldList ::= [VarConstOrTimerCommentList]
                     [ExtraComments]
                     [DetailedComments]
```

8.4 ImportsProforma

```
ImportsProforma ::= ImportsKeyword BeginChar ImportsFieldList EndChar

ImportsKeyword ::= imports

ImportType ::= AllKeyword [DefKeyword] | DefKeyword
ImportIdentifier ::= [TypeDefIdentifier | TemplateIdentifier | ConstIdentifier |
                     TestcaseIdentifier | FunctionIdentifier |
                     NamedAltIdentifier]

Mark ::= ["*"]

ImportsFieldList ::= [GroupDef]
                     SourceKeyword AssignmentChar FreeText SemiColon
                     [SingleComment]
                     [ExtraCommentList]
                     [DetailedComments]

SourceKeyword ::= source

SingleComment ::= CommentsKeyword AssignmentChar FreeText SemiColon
CommentsKeyword ::= comments
```

8.5 EncodingProforma

```
EncodingProforma ::= EncodingKeyword BeginChar EncodingFieldList EndChar
EncodingKeyword ::= encoding

EncodingRuleIdentifier ::= identifier

EncodingFieldList ::=   [GroupDef]
                        EncodingRefList
                        EncodingDefaultList
                        [EncodingCommentList]
                        [DetailedComments]

RefKeyword ::= reference
DefaultKeyword ::= default

EncodingRefList ::= RefKeyword AssignmentChar  BeginChar
                    [EncodingRef {"," EncodingRef}]
                    EndChar [SemiColon]

EncodingRef ::= EncodingRuleIdentifier AssignmentChar FreeText

EncodingDefaultList ::= DefaultKeyword AssignmentChar   BeginChar
                        [EncodingDefault {"," EncodingDefault}]
                        EndChar [SemiColon]]

EncodingDefault ::= EncodingRuleIdentifier AssignmentChar BooleanExpression
```

```
EncodingCommentList ::= CommentListKeyword AssignmentChar BeginChar
                        [EncodingComment {"," EncodingComment}]
                        EndChar [SemiColon]

EncodingComment ::= EncodingRuleIdentifier AssignmentChar FreeText
```

8.6 SimpleTypesProforma

```
SimpleTypesProforma ::= SimpleTypeKeyword BeginChar SimpleTypeFieldList EndChar
SimpleTypesKeyword ::= simpletypes

SimpleTypeFieldList ::=     [GroupDef]
                            [SimpleTypeCommentList]
                            [DetailedComments]

SimpleTypeCommentList ::=   CommentListKeyword AssignmentChar BeginChar
                            [SimpleTypeComment {"," SimpleTypeComment}]
                            EndChar [SemiColon]

SimpleTypeComment ::= SubTypeIdentifier AssignmentChar FreeText
```

8.7 StructuredTypesProforma

```
StructuredTypeProforma ::=  StructTypeKeyword BeginChar StructTypeFieldList
                            EndChar
StructTypeKeyword ::= structuredtype

StructTypeFieldList ::=     [GroupDef]
                            [SingleComment]
                            [StructTypeCommentList]
                            [DetailedComments]

StructTypeCommentList ::=   CommentListKeyword AssignmentChar BeginChar
                            [StructTypeComment {"," StructTypeComment}]
                            EndChar [SemiColon]

StructTypeComment ::= StructFieldIdentifier AssignmentChar FreeText

StructureType ::=  record | set | union
```

8.8 PortTypeProforma

```
PortTypeProforma ::= PortTypeKeyword BeginChar PortTypeFieldList EndChar
PortTypeKeyword ::= porttype

PortTypeFieldList ::=   [GroupDef]
                        [SingleComment]
                        [PortTypeCommentList]
                        [DetailedComments]

PortTypeCommentList ::= CommentListKeyword AssignmentChar BeginChar
                        [PortTypeComment {"," PortTypeComment}]
                        EndChar [SemiColon]

PortTypeComment ::= PortTypeIdentifier AssignmentChar FreeText

PortModelType ::= MessageKeyword | ProcedureKeyword | MixedKeyword
PortTypeDef ::= BeginChar MixedList {SemiColon MixedList} [SemiColon] EndChar
```

8.9 ComponentTypeProforma

```
ComponentTypeProforma ::= ComponentTypeKeyword BeginChar ComponentTypeFieldList
                          EndChar
ComponentTypeKeyword ::= componenttype

ComponentTypeFieldList ::=  [GroupDef]
                            [SingleComment]
                            [VarConstOrTimerCommentList]
                            [ExtraComments]
                            [DetailedComments]

ExtraComments ::= ECommentsKeyword AssignmentChar FreeText SemiColon
ECommentsKeyword ::= extracomments

PortList ::= {PortInstance}

TypeOrTimer ::= Type | TimerKeyword
```

8.10 ConstantsProforma

```
ConstantsProforma ::= ConstantsKeyword BeginChar ConstantsFieldList EndChar
ConstantsKeyword ::= constants

ConstantsFieldList ::=  [GroupDef]
                        [ConstantsCommentList]
                        [DetailedComments]

ConstantsCommentList ::=    CommentListKeyword AssignmentChar BeginChar
                            [ConstantsComment {"," ConstantsComment}]
                            EndChar [SemiColon]

ConstantsComment ::= ConstIdentifier AssignmentChar FreeText
```

8.11 SignatureProforma

```
SignatureProforma ::= SignatureKeyword BeginChar SignatureFieldList EndChar

SignatureFieldList ::=  [GroupDef]
                        [SingleComment]
                        [ExtraComments]
                        [DetailedComments]

SignatureIdentifer&ParList ::=   SignatureIdentifier "("
                                 [SignatureFormalParList] ")"
```

8.12 SimpleTemplatesProforma

```
SimpleTemplatesProforma ::= SimpleTemplatesKeyword
BeginCharSimpleTemplatesFieldList EndChar
SimpleTemplatesKeyword ::= simpleTemplates

SimpleTypeFieldList ::=     [GroupDef]
                            [SimpleTemplatesCommentList]
                            [DetailedComments]

SimpleTemplatesCommentList ::=   CommentListKeyword AssignmentChar BeginChar
                                 [SimpleTemplateComment {","
                                 SimpleTemplateComment}]
                                 EndChar [SemiColon]
```

```
SimpleTemplateComment ::= TemplateIdentifier AssignmentChar FreeText

SimpleType ::= Type | DerivedDef
/* STATIC SEMANTICS – The referenced type or base template shall not be of a
constructed type */
```

8.13 StructuredTemplatesProforma

```
StructuredTemplateProforma ::= StructuredTemplateKeyword BeginChar
TemplateFieldList EndChar
StructuredTemplateKeyword ::= structuredtemplate

TemplateFieldList ::=   [GroupDef]
                        [SingleComment]
                        [TemplateCommentList]
                        [DetailedComments]

TemplateCommentList ::=     CommentListKeyword AssignmentChar BeginChar
                            [TemplateComment {"," TemplateComment}]
                            EndChar [SemiColon]

TemplateComment ::= FieldReference AssignmentChar FreeText

TemplateIdentifier&ParList ::= TemplateIdentifier ["(" TemplateFormalParList")" ]
TemplateStructIdentifier ::= Type | Signature | DerivedDef
```

8.14 FunctionProforma

```
FunctionProforma ::= FunctionKeyword BeginChar FunctionFieldList EndChar

FunctionFieldList ::=   [GroupDef ]
                        [SingleComment]
                        [VarConstOrTimerCommentList]
                        [ExtraComments]
                        [DetailedComments]

FunctionIdentifier&ParList ::= FunctionIdentifier "(" [FunctionFormalParList]")"

FunctionCommentList ::= FunctionComment {"," FunctionComment}
FunctionComment ::= VarConstOrTimerRef AssignmentChar FreeText

VarConstOrTimerCommentList ::= CommentListKeyword AssignmentChar  BeginChar
                                                                  [FunctionCommentList]
                                                                  EndChar [SemiColon]

VarConstOrTimerIdentifier ::=    ConstKeyword ConstIdentifier |
                                 VarIdentifier [ArraySpec] |
                                 TimerIdentifier [ArraySpec]

VarConstOrTimerRef ::=           ConstIdentifier | VarIdentifier |
                                 TimerIdentifier

TabularBehaviour ::= FunctionBody
/* STATIC SEMANTICS – The FunctionBody production shall not contain any variable,
timer or constant definitions */
```

8.15 DefaultsProforma

```
DefaultProforma ::= DefaultKeyword BeginChar DefaultFieldList EndChar
DefaultKeyword ::= default

DefaultFieldList ::=    [GroupDef]
                        PurposeDef
                        [SingleComment]
                        [DetailedComments]

PurposeDef ::= PurposeKeyword AssignmentChar FreeText SemiColon
PurposeKeyword ::= purpose
NamedAltIdentifier&ParList ::= NamedAltIdentifier ["(" FunctionFormalParList")"]
```

8.16 NamedAltProforma

```
NamedAltProforma ::= NamedAltKeyword BeginChar DefaultFieldList EndChar
NamedAltKeyword ::= namedalt
```

8.17 TestcaseProforma

```
TestcaseProforma ::= TestcaseKeyword BeginChar TestcaseFieldList EndChar

TestcaseFieldList ::=       [GroupDef]
                            PurposeDef
                            [SingleComment]
                            [VarOrTimerCommentList]
                            [ExtraComments]
                            [DetailedComments]


TestcaseIdentifier&ParList ::= TestcaseIdentifier "("
[TestcaseRestrictedFormalParList]")"
TestcaseRestrictedFormalParList ::= FormalVarValuePar {"," FormalVarValuePar}
```

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	**Languages and general software aspects for telecommunication systems**

Geneva, 2001