EG 201 148 V1.1.2 (1998-03)

ETSI Guide

**Methods for Testing and Specification (MTS);
Guide for the use of the second edition of TTCN**

European Telecommunications Standards Institute

Reference

DEG/MTS-00036 (amo00ide.PDF)

Keywords

TTCN, testing,

Contents

Intellectual Property Rights

Foreword

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

Introduction

The present document summarizes the additional features and capabilities of the Tree and Tabular Combined Notation (TTCN) that are considered to be of most interest when specifying Abstract Test Suites (ATS) for telecommunications protocols and services. The features that support major new functionality are:

- concurrency, which allows execution of different dynamic behaviours in parallel;
- encoding, which allows to define encoding for each PDU type and to modify it for each PDU Constraint (or even each field);
- modularity, which allows to reuse parts of existing ATS and to specify modules.

Less extensive, but nonetheless important, changes are:

- the possibility to pass matching symbols to Constraints;
- PDUs need not contain any fields;
- the RETURN statement to exit Test Steps (TS);
- it is no longer mandatory that an OTHERWISE in a Default must lead to a fail verdict;
- Collective Comments in some tables;
- predefined type (R_Type) for verdicts;
- the use of Active Defaults to switch on/off default behaviour;
- Test Suite Operations (TSOs) may be specified as procedures rather than informal text (not described in the present document);
- ability to declare Test Suite Constants by reference (not described in the present document).

1 Scope

The present document provides an introduction to the new features defined in the second edition of the Tree and Tabular Combined Notation (TTCN) as defined in TR 101 101 [1]. The second edition of TTCN incorporates Amendments 1 and 2 and a number of technical corrigenda into the 1991 TTCN standard as defined in ISO/IEC 9646-3 [2].

The present document is intended to be used as introductory reference material for those wishing to gain an understanding of the new TTCN features (a more detailed version of the present document is intended to be produced by MTS in 1998).

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] TR 101 101 (1997): "Methods for Testing and Specificatin (MTS); TTCN interim version including ASN.1 1994 support [ISO/IEC 9646-3] (second edition mock-up for JTC1/SC21 review)".

[2] ISO/IEC 9646-3 (1991): "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)".

[3] CCITT Recommendation X.208 (1990) : "Specification of Abstract Syntax Notation One (ASN.1)".

3 Abbreviations

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

ASN.1	Abstract Syntax Notation One
ASP	Abstract Service Primitive
BER	Basic Encoding Rules
CM	Coordination Message
CP	Coordination Point
IUT	Implementation Under Test
MTC	Main Test Component
PCO	Point of Control and Observation
PDU	Protocol Data Unit
PTC	Parallel Test Component
TS	Test Steps
TTCN	Tree and Tabular Combined Notation
UT	Upper Tester

4 Concurrency

4.1 Introduction to concurrent TTCN

Unlike TTCN version 1, concurrent TTCN allows test suites:

- to use more than two Points of Control and Observation (PCO);
- to use more than one underlying service provider;
- to have dynamic behaviours executing in parallel;
- to specify co-ordination between concurrently executing components.

These capabilities have introduced a number of additional concepts, proformas, statements and verdict mechanisms to TTCN.

Additional concepts:

- Test Components;
- Test Component Configurations;
- Coordination Points (CPs);
- Coordination Messages (CMs).

Additional proformas:

- Test Component Declarations;
- Test Component Configuration Declaration(s);
- CP Declarations;
- CM Declarations (tabular and Abstract Syntax Notation No.1 (ASN.1));
- CM Constraints (Tabular and ASN.1).

Additional constructs and statements:

- CREATE
- DONE

Additional verdict mechanisms:

- local result variables;
- global result variable.

4.2 Test Components

The building-blocks of Concurrent TTCN are called Test Components. In the present document we will sometimes use the term component for short. A Test Component can be either a Main Test Component (MTC) or a Parallel Test Component (PTC). All Test Components are declared in a single Test Components Declaration table. In a Test Suite this table comes after the Timer Declarations table.

Test Component Declarations				
Component Name	**Component Role**	**Nr of PCOs**	**Nr of CPs**	**Comments**
MTC1	MTC	0	2	
PTC1	PTC	1	1	
PTC2	PTC	1	1	

Figure 1: Declaring Test Components

Each component must have a name that is unique within the test suite and be assigned the role of either **MTC** or **PTC**. This table must contain at least one MTC and zero or more PTCs.

NOTE 1: MTC and PTC are now reserved words in TTCN.

At this stage we need only define placeholders for the actual PCOs and Coordination Points (CPs) that will later be associated with the components in a particular configuration. This is done by simply stating the number of PCOs and/or CPs that may be associated with each component.

NOTE 2: A Test Component may have neither PCOs nor CPs. In fact, it is quite feasible that this could apply even to an MTC. The MTC could CREATE the PTCs and achieve co-ordination, albeit limited, using the DONE statement and the implicit verdict mechanism.

4.3 Co-ordination of Test Components

Explicit co-ordination between components is achieved using CPs and CMs.

4.3.1 Coordination points

CPs are very similar to PCOs. They allow asynchronous communication between exactly two PTCs or between one PTC and the MTC. In other words, a CP may not be shared between more than two Test Components. A CP may not be connected to the Implementation Under Test (IUT), either directly or indirectly via a service provider.

In a test suite the CP declarations come after the PCOs.

Coordination Point Declarations	
CP Name	**Comments**
CP1	
CP2	

Figure 2: Declaration of Coordination Points

There is a predefined type in TTCN called **CP**. This is useful when parameterizing PTCs.

NOTE: CP is now a reserved word in TTCN.

4.3.2 Coordination Messages

CMs are very similar to Abstract Service Primitives (ASP) except that they occur at CPs. You can do all the things with a CM that you can do with an ASP. Either the tabular form or ASN.1 as defined in CCITT Recommendation X.208 [3] can be used.

NOTE: CMs are treated like ASPs and not as Protocol Data Units (PDUs) because TTCN is not concerned with the encoding of these messages. That is a matter for the implementors of the test suite.

A CM parameter may be of any TTCN type including structured types and the metatype **PDU**. Generally, though, it is recommended that CMs are kept as simple as possible. In many cases a CM will not even have parameters, the name itself will be adequate (e.g. STOP, WAIT, etc.).

There are no predefined TTCN CMs.

CMs may be declared using either TTCN tables or ASN.1. In a test suite, CM Declarations come after the PDU Declarations.

CM Type Definition		
CM Name : CM_ERROR Comments :		
Parameter Name	**Parameter Type**	**Comments**
Error	INTEGER	

Figure 3: Definition of a Coordination Message

CM Constraints are similar to ASP Constraints. In a test suite, CM Constraints come after the PDU Constraints.

CM Constraint Declaration		
Constraint Name : ERR (err_num:INTEGER) CM Type : CM_ERROR Derivation Path : Comments :		
Parameter Name	**Parameter Value**	**Comments**
Cause	err_num	

Figure 4: A Coordination Message Constraint

Note that when a CM has no parameters it is not necessary to define a constraint for it. This also means that an entry in the constraints' reference column of a dynamic behaviour is not required.

NOTE: In TTCN Version 2 this point also applies to ASPs and PDUs generally.

4.4 Test Configurations

An actual abstract test architecture is defined by connecting together a number of Test Components in what is called a Test Component Configuration. In the present document we will sometimes use the term configuration for short.

A configuration will consist of exactly one MTC and zero or more PTCs.

In most practical applications a single ATS will make use of more than one configuration. Each configuration is defined in a separate Test Component Configuration Declaration Table. In a test suite, Test Component Configuration Declarations come after the Test Component Declarations.

Test Component Configuration Declaration			
Configuration Name : CONFIG1 Comments :			
Components Used	**PCOs Used**	**CPs Used**	**Comments**
MTC1		CP1, CP2	
PTC1	PCO1	CP1	
PTC2	PCO2	CP2	

Figure 5: A typical configuration

4.4.1 Connection to the IUT

The PCOs **Used** column lists the actual PCOs (if any) associated with the Test Components. The relation of the PCOs to the Implementation Under Test (IUT), either directly (as in the case of a PTC that is part of an Upper Tester(UT)) or indirectly via an underlying service provider is indicated, as usual, in the PCO Declarations table.

Note the following rules:

- each entry in this column is a list of zero or more PCOs;
- for each MTC and each PTC the number of entries in this list must be the same as the corresponding number of PCOs stated in the declaration of the component;
- no PCO may be used more than once in a single configuration (i.e. Test Components cannot share PCOs).

4.4.2 Connecting MTCs and PTCs

The CPs Used column lists the actual CPs (if any) associated with the Test Components. These entries are used to interconnect Test Components.

Note the following rules:

- each entry in this column is a list of zero or more CPs;
- for each PTC the number of entries in this list must be the same as the corresponding number of CPs stated in the declaration of the component;
- for each MTC the number of entries may be the same or less but not more. This allows for flexibility when using the same MTC in various configurations;
- no CP name is allowed to appear more than once in a single list;
- each CP name that is in one list must appear in exactly one other list. In this manner you can specify connected pairs.

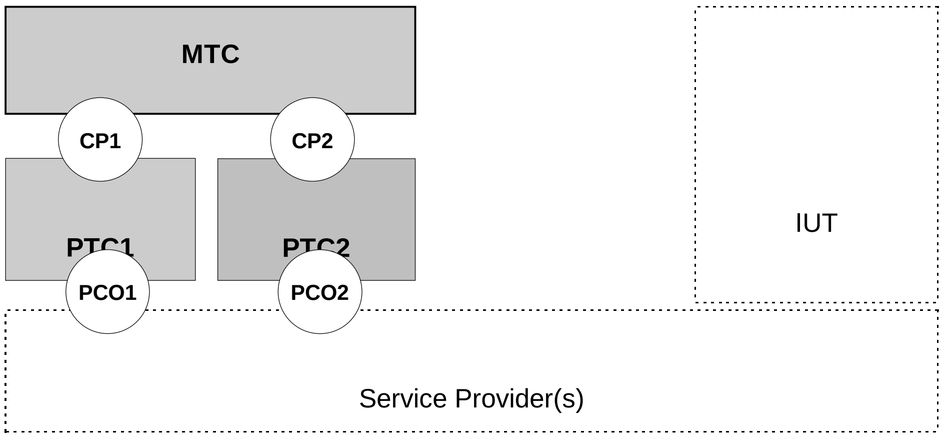

Figure 6: The configuration specified in figure 5

4.4.3 The Master and Parallel Test Components

Each configuration shall have one and only one MTC. Furthermore, the MTC shall be located on the Lower Tester side of the architecture. The MTC is responsible for creating the PTCs, for overall co-ordination of the Test Case and for assigning the final verdict.

A particular configuration is associated to a Test Case by the (new) entry in the Test Case Dynamic Behaviour header.

Test Case Dynamic Behaviour					
Test Case Name :					
Group :					
Purpose :					
Configuration : CONFIG1					
Defaults :					
Comments :					
Nr	**L**	**Behaviour Description**	**Constraint Ref**	**V**	**Comments**

Figure 7: Associating a configuration with a Test Case

This means that the Test Case *is* the MTC. The variables, constants, timers, behaviour, etc. of the Test Case are the variables, constants, timers, behaviour, etc. of the MTC.

4.4.4 The CREATE construct

The MTC binds dynamic behaviour to the PTCs using the CREATE construct. This construct also has the effect of starting execution of the named PTC or PTCs.

7		:			
8		CREATE (PTC1:Step1 (CP1), PTC2:Step2(CP2))			
9		:			

Figure 8: The CREATE construct

In the previous example, the Test Step Step1 is bound to PTC1 and the Test Step Step2 is bound to PTC2.

NOTE 1: Parameters may be passed to the PTCs, in this case the Coordination Points CP1 and CP2.

NOTE 2: The Test Steps may either reside in the Test Step library or be local trees.

As each created PTC is a separate executing entity the same Test Step may be used, if wished, to define the behaviour of different PTCs.

Test Steps that are invoked from a Test Case in the normal manner (i.e. + TestStepName) are part of the MTC and should not be considered as PTCs, only Test Steps that are invoked from a Test Case using the CREATE construct become PTCs.

4.4.5 Scope of variables

At the start of execution of the Test Case each Test Component is allocated its own fresh set of variables, timers, constraints, etc. These are limited in scope to the Test Component during the life of the Test Case: data is not shared between Test Components.

Only an MTC may use Test Suite Variables.

4.5 Verdicts

Concurrent TTCN includes a number of predefined variables for tracking the intermediate results and the final verdict. These are:

- each PTC maintains its own local result variable called R;
- the MTC also has a local result variable but this is called MTC_R;
- the MTC also maintains a global result variable called R.

A PTC may assign a preliminary result (for example (P)) in its verdict column. This has the effect of updating both the local R and the global R according to the priority table defined in ISO/IEC 9646-3 [2].

NOTE: These priority rules are unchanged from TTCN version 1. Also, MTC_R is now a TTCN reserved word.

A PTC may also assign a result without parentheses (for example P), in its verdict column. This updates the local and global result variables as usual and terminates execution of the PTC. However, this result is not to be considered as a final verdict.

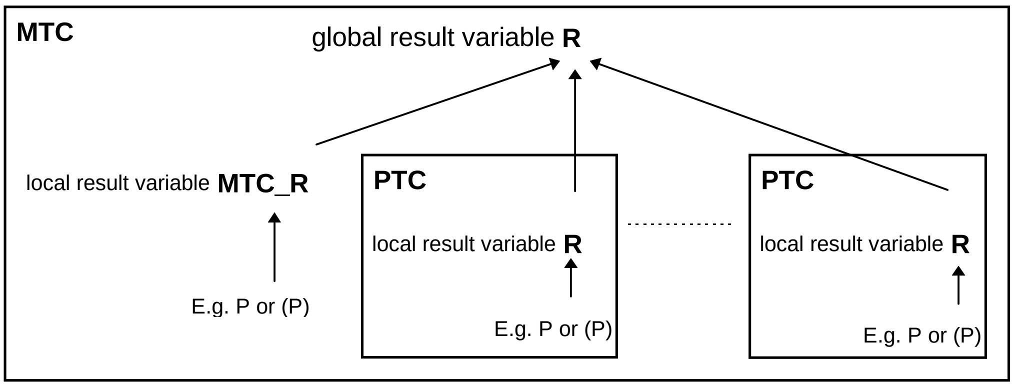

Figure 9: Relation between the different result variables

Similarly, the MTC will update its local result variable MTC_R and the global result variable R in accordance with the priority rules. A result without parentheses in the MTC is considered to be the final verdict and has the effect not only of terminating execution of the MTC but also of any PTCs that are still executing.

The TTCN standard does not say anything about the actual mechanisms that perform the updating of the local and global R variables the and MTC_R variable. This is a matter for the implementors of the test suite. Use these variables with care, for example always after DONE.

4.5.1 The DONE Statement

The DONE statement is used to check whether or not a Test Component has terminated execution. It may be used by both MTCs and PTCs, but should generally be used by the MTC to ensure that all PTCs have ceased execution before assigning the final verdict.

5		:			
6		?DONE (PTC1, PTC2)		PASS	
7		?TIMEOUT		FAIL	
8		:			

Figure 10: The DONE Statement

The DONE statement has no effect on the Test Component being interrogated (it does not terminate execution of a Test Component). In a PTC termination will either occur:

- when the PTC behaviour tree ends in a leaf; or
- when an entry without parentheses is encountered in the verdicts column of a PTC; or
- when the PTC is instructed to do so by an appropriate CM (which must be defined in the test suite).

A DONE statement without an argument list has the same effect as interrogating all the PTCs that have been created prior to the DONE. Only an MTC may do this.

4.6 Summary of MTC and PTC characteristics

Table 1 summarizes the main characteristics of MTCs and PTCs.

Table 1: Characteristics of MTCs and PTCs

Capability	MTC	PTC
Is associated with a ...	Test Case	Test Step
Can use the CREATE construct	Yes	No
Can use the DONE statement	Yes	Yes
Results written to the global result variable R	Yes	Yes
Results written to the local result variable MTC_R	Yes	No
Results written to a local result variable R	No	Yes
Can assign a final verdict	Yes	No
Can use all TTCN types	Yes	Yes
Can use Test Suite Parameters and other Constants	Yes	Yes
Can use Test Suite Variables	Yes	No
Scope of Test Case Variables, Timers, Operations, etc.	Local to MTC	Local to PTC
Scope of attached Test Steps and Defaults	Local to MTC	Local to PTC

5 Encoding

5.1 Introduction to specifying encoding information in TTCN

This addition to TTCN allows test suites to specify the encoding of PDUs. This is useful where a base standard offers different choices of encoding rules and/or to test how an IUT handles invalid encoding. Mechanisms are provided for the definition of:

- general encoding;
- variations on the general encoding;
- invalid field encoding.

These mechanisms may be used both with tabular TTCN PDUs and ASN.1 PDUs. They shall not be used for ASPs or CMs, the encoding of which is an implementation matter.

These capabilities have introduced a number of additional proformas and added header entries and additional columns to some existing TTCN proformas.

Additional proformas:

- encoding definitions;
- encoding variations;
- invalid field encoding operation definition(s).

Changes to existing proformas:

- structured type definition(s) - one additional header entry, one additional column;
- ASN.1 Type definition(s) - one additional header entry, some additional syntax;
- ASN.1 Type definitions by Reference - one additional column;
- PDU Type definition(s) - two additional header entries, one additional column;
- ASN.1 PDU type definition(s) - two additional header entries, some additional syntax;
- ASN.1 PDU type definitions by reference - two additional columns;
- corresponding changes to the relevant constraints tables have also been made.

NOTE: All these changes are optional and need not appear if encoding is not used.

5.2 Basic principles

The mechanisms for TTCN encoding information shall only be used if the protocol or service specification defines or makes reference to a standardized set of encoding rules. If no such rules exist then a TTCN test suite cannot 'invent' them.

Encoding Information can be of the following kinds:

- reference to the general encoding rule(s) applicable to the entire test suite;
- definitions of the encoding variations (if any) on the general encoding rule(s);
- definitions of invalid PDU field encoding (if any).

The encoding may be applied at five levels, given below in *increasing* order of priority:

- the top-level applies to the entire test suite, that is, the encoding rules apply to all PDUs sent or received in the test suite, unless overridden by one of the following cases;
- the second level applies to all PDUs of a particular type. All PDUs of this type will be encoded according to the variant given in the PDU type definition header;
- the third level allows specific fields of a particular PDU type to be given either a variant encoding or an invalid encoding;
- the fourth level allows an (entire) individual constraint to be given a variant encoding;
- finally, the fifth level allows specific fields of an individual PDU Constraint to be given either a variant encoding or an invalid encoding;

NOTE: For simplicity, in this section we have only talked about PDUs and PDU constraints. The above equally applies to structured types and structured constraints (that are used in PDUs). Similarly, the same applies to Simple Types and Structure elements as to PDU fields.

5.3 Encoding Definitions

The Encoding Definitions table states the encoding rules that are used in the ATS by referencing the appropriate standard where the rules are defined. In a Test Suite this table comes after the test suite type definitions and before the Test Suite Operation Definition(s).

Encoding Definitions			
Encoding Rule Name	**Reference**	**Default**	**Comments**
BER	ISO/IEC 8825-1: 1993	TSP1	
PER	ISO/IEC 8825-1: 1993	[TSP2 OR TSP3]	
DER	ISO/IEC 8825-1: 1993		

Figure 11: Encoding Definitions

The Boolean expression in the Default column is used to determine the default set of encoding rules. In our example, if the Test Suite Parameter TSP1 has the value TRUE then BER is used. If either TSP2 or TSP3 evaluates to TRUE then PER applies. No entry in the Default column has the same effect as writing FALSE. If this column is empty then the first entry in the Encoding Rule Name column is taken to be the default.

5.4 Encoding Variations

The Encoding Variations table states variations on the general encoding rules that are used in the ATS, if any. These variations must be allowed according to the original encoding rules. In a Test Suite this table comes after the Encoding Definitions.

Encoding Variations			
Encoding Rule Name : BER			
Type List : INTEGER			
Comments :			
Encoding Variation	**Reference**	**Default**	**Comments**
SD	6.3.3.1	TRUE	
LD (len:INTEGER)	6.3.3.1		

Figure 12: Encoding Variations

This example shows two variations of Basic Encoding Rules (BER), (Short Definite and Long Definite encodings for INTEGER values). The default variant is Short Definite.

The Type List entry indicates to which types this encoding applies. If this entry is empty then the encoding applies to all types.

NOTE: The encoding variation may be followed by a formal parameter list, if required. In our example, *len* states the length of the LD encoding.

5.5 Invalid Field Encoding Definitions

The Invalid Field Encoding Definitions table is used to define invalid encodings of PDU fields, if any. In a Test Suite these tables come after the Encoding Variations.

The invalid encodings are defined using the (new) TTCN syntax for procedures (not the free text format).

5.6 Using variant and invalid encodings in PDUs and Constraints

In the following example, let us assume that the default encoding rules BER apply (as stated in the Encoding Rules table earlier).

PDU Type Definition

PDU Name	:	A_PDU
PCO Type	:	L
Encoding Rule Name	:	
Encoding Variation	:	LD
Comments	:	All PDUs (Constraints) of this type will be encoded to BER. However, all fields of type INTEGER in this PDU will be encoded to the Long Definite (LD) variation.

Field Name	Field Type	Field Encoding	Comments
F1	INTEGER		This field will be encoded Long Definite
F2	INTEGER		This field will be encoded Long Definite
F3	INTEGER		This field will be encoded Long Definite

Figure 13: PDU Type definition with encoding information

PDU Constraint Declaration

Constraint Name	:	C1
PDU Type	:	A_PDU
Derivation Path	:	
Encoding Rule Name	:	
Encoding Variation	:	
Comments	:	This constraint will be encoded according to the encoding information given in the definition of A_PDU (i.e. BER and Long Definite). However, the encoding of fields F1 and F2 is explicitly overridden.

Field Name	Field Value	Field Encoding	Comments
F1	123	SD	Switch back to Short Definite
F2	456	INVALID_LENGTH(2)	This field will given an invalid encoding
F3	789		This field will be encoded Long Definite

Figure 14: A constraint with encoding derived from the PDU type

5.7 Using the ENC keyword

In ASN.1 PDU Type Definitions and ASN.1 PDU Constraint Declarations the field encodings are specified using the ENC keyword instead of the additional column. In ASN.1 the constraint of the example above would be:

ASN.1 PDU Constraint Declaration

Constraint Name	:	C1
PDU Type	:	A_PDU
Derivation Path	:	
Encoding Rule Name	:	
Encoding Variation	:	
Comments	:	This constraint will be encoded according to the encoding information given in the definition of A_PDU. However, the encoding of fields F1 and F2 is explicitly overridden.

Constraint Value

```
SEQUENCE {
    f1 123 ENC SD;
    f2 456 ENC INVALID_ENCODING(2);
    f3 789 }
```

Figure 15: Use of the ENC keyword in ASN.1

6 Modular TTCN

6.1 Introduction

The introduction of modularity in TTCN allows the separation of parts of an abstract test suite into modules. This is particular useful in view to maintencance and reusability issues. TTCN Edition 2 permits the modularization of test suites by allowing the specification of two separate entities:

- Test Suites (as in normal TTCN);
- Modules.

A Test Suite may now consist of five (rather than four) parts, that is:

- Test Suite Overview Part;
- Imports Part (new);
- Declarations Part;
- Constraints Part;
- Dynamic Part.

Whereby the Test Suite Overview Part contains a new section for the Exports Part:

- Test Suite Index;
- Test Suite Structure;
- Test Case Index;
- Test Step Index;
- Default Index;
- Test Suite Exports (new).

A Module consists of a similar five parts:

- Module Overview Part;
- Module Imports Part;
- Declarations Part;
- Constraints Part;
- Dynamic Part.

Whereby the Module Overview Part contains the section for the Module Exports:

- TTCN Module Exports;
- TTCN Module Structure;
- Test Case Index;
- Test Step Index;
- Default Index.

These capabilities have introduced a number of additional concepts and proformas to TTCN.

Additional concepts:

- Modules;
- Import and Export of objects;
- EXTERNAL objects.

Additional proformas:

- Imports proforma used in Test Suite and Modules.

Imports			
Source Name : **Source Ref :** **Standards Ref :** **Comments :**			
Object Name	**Object Type**	**Source Name**	**Comments**
Detailed Comments:			

- Exports proforma used in Test Suites.

Test Suite Exports				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
Detailed Comments:				

- Exports proforma used in modules.

Module Exports				
TTCN Module Name : **Objective :** **TTCN ModuleRef :** **Standards Ref :** **PICS Ref :** **PIXIT Ref :** **Test Method(s) :** **Comments :**				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
Detailed Comments:				

- External objects proforma used in modules located in the module imports part.

External Objects		
Object Name	**Object Type**	**Comments**

6.2 Additions to Test Suites

The types of objects that may be imported/exported are:

Table 2: List of object types which can be imported

SimpleType_Object	Timer_Object	StructTypeConstraint_Object
StructType_Object	Tcomp_Object	ASN1_TypeConstraint_Object
ASN1_Type_Object	TcompConfig_Object	TTCN_ASP_Constraint_Object
TS_Op_Object	TTCN_ASP_Type_Object	ASN1_ASP_ Constraint_Object
TS_Proc_Object	ASN1_ASP_Type_Object	TTCN_PDU_ Constraint_Object
TS_Par_Object	TTCN_PDU_Type_Object	ASN1_PDU_ Constraint_Object
SelectExpr_Object	ASN1_PDU_Type_Object	TTCN_CM_ Constraint_Object
TS_Const_Object	TTCN_CM_Type_Object	ASN1_CM_ Constraint_Object
TS_Var_Object	ASN1_CM_Type_Object	TestCase_Object
TC_Var_Object	EncodingRule_Object	TestStep_Object
PCO_Type_Object	EncodingVariation_Object	Default_Object
PCO_Object	InvalidFieldEncoding_Object	NamedNumber_Object
CP_Object	Alias_Object	Enumeration_Object

The imported objects are declared in the Imports proforma. In a Test Suite this table comes after the test suite overview.

Imports			
Source Name : Module_1 **Source Ref** : **Standards Ref** : **Comments** :			
Object Name	**Object Type**	**Source Name**	**Comments**
SimpleType_A Timer_A PDU_A	SimpleType_Object Timer_Object TTCN_PDU_Type_Object	Omit Module_2	
Detailed Comments:			

Figure 16: Use of the Import proforma

The exported objects are declared in the exports proforma which is located in the test suite overview part.

Test Suite Exports				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
String5	SimpleTypeDef_Object		3	
Wait	TimerDcl_Object	Module_B		
INTC	TTCN_PDU_Type_Object		13	
DEF1	Default_Object	TestSuite_1		
TC_2	TestCase_Object	TestSuite_2		
TC_3	TestCase_Object		33	
Preamble	TestStep_Object	EXTERNAL		
Detailed Comments:				

Figure 17: Use of the Export proforma

6.3 Modules

A module is very similar to a Test Suite in that it may contain declarations, constraints and dynamic behaviours but it is not complete in itself. In practice a module should concentrate on a particular aspect, for example, only constraints or only Test Steps. A Module also has an overview part, similar in function to the Test Suite Overview.

Objects defined in modules are intended to be imported by Test Suites or other Modules. The objects that are visible to (i.e. may be imported by) Test Suites and other Modules must be declared in the Module Exports proforma.

Module Exports				
TTCN Module Name :	Module_A			
Objective :				
TTCN ModuleRef :				
Standards Ref :				
PICS Ref :				
PIXIT Ref :				
Test Method(s) :				
Comments :				
Object Name	**Object Type**	**Source Name**	**Page Nr**	**Comments**
String5	SimpleTypeDef_Object	Suite_1	2	
Wait	TimerDcl_Object	Module_1	3	
INTC	TTCN_PDU_Type_Object	EXTERNAL	13	
DEF1	Default_Object		45	
TC_2	TestCase_Object		56	
TC_3	TestCase_Object		67	
Preamble	TestStep_Object			
Detailed Comments:				

Figure 18: Use of Module exports

NOTE: The Source Name column may either:

- be empty, i.e. the object is defined in this module; or
- contain a Module Identifier, i.e. the object is defined in another Module; or
- contain a Test Suite Identifier, i.e. the object is defined in another Test Suite; or
- contain the keyword EXTERNAL, i.e. the object is defined externally.

Objects that are defined in the Module, but not declared in the Module Exports table may not be imported directly although they may be used by the exported objects. For example, suppose the Module defines Test_Case_A which attaches Test_Step_B but only Test_Case_A is declared in the Module Exports table. This means that Test_Step_B is still used by Test_Case_A but cannot itself be imported from the Module.

6.4 External Objects

This proforma list the objects being referenced by their identifier in a TTCN module. The object defined in the External Objects table need to be defined when importing the TTCN module.

External Objects		
Object Name	**Object Type**	**Comments**
CRC(P:A_Pdu)	TS_Op_Object	
CONSTRAINT_A(acstr:T_CONNECT)	TTCN_PDU_Constraint_Object	
TESTSTEP_A(I:INTEGER)	TestStop_Object	
DEF3	Default_Object	

Figure 19: Use of External objects

NOTE: For objects that have formal parameter lists then the list shall be provided too.

6.5 Renaming

It is possible that an object is imported from a module or test suite exists already in the importing instance. In this case a name clash occurs. The importing instance then needs to resolve the name clash by renaming the imported object to:

Source_Module_or_Test_Suite::Object_Identifier

The renaming of an object means:

- the object definition; and/or
- the references to the object;

are renamed.

7 Other additional features

This clause describes various other minor, but important, features.

7.1 Passing Matching Symbols to constraints

Matching symbols may now be passed as actual parameters to constraints. An example is given in figure 20.

7		:			
8		L?A_PDU	C1(?, (1,2,3), *,(1..6), "ab?xy*z")		
9		:			

Figure 20: Use of matching symbols in a constraints reference

7.2 Empty PDUs

PDU declarations need not have any field entries (i.e. the body of the table may be empty) if the corresponding PDU in the standard does not have any fields. Previously this applied only to ASPs. This also means that constraints for the 'empty' PDU need not be defined.

7.3 The RETURN statement

This statement may only appear in Default dynamic behaviour descriptions. It is intended to be used when incoming PDUs can occur at any time but which are not considered to be part of the test purpose and which should be ignored. A RETURN from a Default will cause processing to continue at the first alternative in the set of alternatives that caused the Default behaviour to be invoked. For example:

7		:			
8		L? PDU_1	C1		Ignore this PDU
9		RETURN			
10		?OTHERWISE		inconc	
11		:			

Figure 21: Use of the RETURN statement

NOTE: Do not confuse RETURN with the RETURNVALUE keyword which is part of the procedural test suite operations syntax.

7.4 OTHERWISE and the fail verdict

It is no longer mandatory that an OTHERWISE in a Default must lead to a fail verdict.

7.5 Collective comments

Tables for multiple TTCN objects (e.g. Test Suite Parameters, Test Case Variables) may now contain additional lines called Collective Comments that can be used to group entries in the table. For example:

Test Suite Parameter Declarations			
Parameter Name	**Type**	**PICS/PIXIT Ref**	**Comments**
The following parameters are used only in test cases for Valid behaviour			
VPAR1 VPAR2	INTEGER BOOLEAN	ref1 ref2	
The following parameters are used only in test cases for Invalid behaviour			
IPAR1 IPAR2	INTEGER BOOLEAN	ref3 ref4	

Figure 22: Use of collective comments

7.6 R_Type

This is a new predefined type associated with values of verdicts (pass, fail, inconc, none). It can be useful if verdicts need to be carried, say, in a Coordination message.

NOTE: This facility is less useful now that Concurrent TTCN supports the implicit passing of verdicts.

7.7 Test Suite Constants by Reference

Test Suite Constants that are defined externally in ASN.1 may now be referenced using the following proforma:

Test Suite Constant Declarations by Reference			
Constant Name	**Type**	**Value Reference**	**Comments**
TC1 TC2	INTEGER BOOLEAN	value_of_tc1 value_of_tc2	

Figure 23: Use of Test Suite Constant Declarations by Reference

7.8 PCO Type Declarations

PCO Types must be declared in a (new) single PCO Types Declaration table. In a Test Suite this table comes before PCO Declarations.

PCO Type Declarations		
Type Name	**Role**	**Comments**
TSAP SSAP	LT UT	

Figure 24: Declaration of Coordination Points

Two different PCOs, having the same PCO Type, may now use the same ASPs and/or PDUs.

7.9 ACTIVATE statement

The ACTIVATE statement allows Defaults to be selectively activated/deactivated during test case execution. This feature is powerful and should be used with some care.

History

Document history		
V1.1.1	December 1997	Membership Approval Procedure MV 9808: 1997-12-23 to 1998-02-20
V1.1.2	March 1998	Publication

ISBN 2-7437-2029-8
Dépôt légal : Mars 1998