TTCN-3 Quick Reference Card

- with links to TTCN-3 online tests -

For TTCN-3 edition 4.5.1 (2013-04) and extensions. *(PDF has direct links to online standards and browseable BNF)*

Designed and edited by Axel Rennoch, Claude Desroches, Theo Vassiliou and Ina Schieferdecker.

Contents

NOTE:

This Reference Card summarizes language features to support users of TTCN-3. The document is not part of a standard, not warranted to be error-free, and a 'work in progress'. For comments or suggestions please contact the editors via ttcn3-qrc@blukaktus.com.

Numbers in the right-hand column of the tables refer to sections or annex in ETSI standards ES 201873-x and language extensions ES 20278x. NEW Languages elements introduced in edition 4.4.1 (or later) have been marked.

Conventions

BNF DEFINITIONS		*TTCN-3 SAMPLES*	
::=	is defined to be;	**keyword**	identifies a TTCN-3 keyword;
abc xyz	*abc* followed by *xyz;*	"string"	user defined character string;
|	alternative;	// comments	user comments;
[abc]	0 or 1 instance of *abc;*	@desc	user documentation comments (T3DOC);
{abc}	0 or more instances of *abc;*	*Italic*	indicates literal text to be entered by the user;
{abc}+	1 or more instances of *abc;*	**[…]**	indicates an optional part of TTCN-3 source code;
(…)	textual grouping;	**…**	indicates additional TTCN-3 source code;
abc	the non-terminal symbol *abc;*	**<empty>**	string of zero length;
"abc"	the terminal symbol *abc;*		

Selected BNF definitions have links to browseable BNF provided by http://www.trex.informatik.uni-goettingen.de/trac/wiki/ttcn3-bnf.

A.1 Structuring

MODULE, IMPORT, GROUP	*EXAMPLES*	DESCRIPTION	§
module *ModuleIdentifier* [**language** *FreeText* {"," *FreeText*}] "**{**"[*ModuleDefinitionsPart*] [*ModuleControlPart*] "**}**"	**module** *MyTypes* **language** "TTCN-3:2013" **{…}** **module** *MyConfig* **language** "TTCN-3:2012" **{…}**	present version 4.5.1; version 4.4.1;	8.1
[*Visibility*] **import from** *ModuleId* ((**all** [**except** "**{**" *ExceptSpec* "**}**"]) |("**{**" *ImportSpec* "**}**")) ["**;**"]	**public import from** *MyModule* **language** "XSD" **all;**	definitions visible in defining and other (importing) module;	8.2.3
	friend import from *MyModule* **{type** *MyType*, **template all};**	definition visible in defining and friend modules;	
	private import from *MyIPs* **all except {group** *myGroup***};**	definitions in *MyIPs* cannot be imported by other modules;	8.2.5
[**public**] **group** *GroupIdentifier* "**{**" {*ModuleDefinition* ["**;**"]} "**}**"	**group** *myGroup* **{group** *mySubGroup* **{…}; …}**	groups can only have public visibility;	8.2.2
[**private**] **friend module** *ModuleIdentifier* {"," *ModuleIdentifier*} "**;**"	**friend module** *MyTestSuiteA***;**	this module is defined to be a friend to *MyTestSuiteA;*	8.2.4

GENERAL SYNTAX	*EXAMPLES*	DESCRIPTION	§
terminator (";")	*f_step1*() **;** *f_step2*() **;**	optional if construct ends with "}" or next symbol is "}"	A.1.2
identifiers	*v_myVariable*	case sensitive, must start with a letter (a-z, A-Z), may contain digits (0-9) and underscore (_)	A.1.3
free text comments	*/* block comment */* *f1*(); *// single line comment*	nested block comments not permitted; start with **//** and end with a newline;	A.1.4

A.2 Components and communication interfaces

COMPONENTS	*EXAMPLES*	DESCRIPTION	§
type component *ComponentTypeIdentifier* [**extends** *ComponentTypeIdentifier*] "**{**" { (*PortInstance* | *VarInstance* | *TimerInstance* | *ConstDef*) } "**}**"	**type component** *MyPtcA* **{port** *MyPortTypeA myPort***;** **port** *MyPortTypeA myPorts***[3];** **var** *MyVarTypeA v_var1* **};** **type component** *MyPtcB* **extends** *MyPtcA* **{timer** *t_myTimer***};** **private type component** *MyPtcC* **{…};**	declarations could be used in testcase, function, etc. that runs on *MyPtcA***;** array of three ports; in addition to the timer, *MyPtcB* includes all definitions from *MyPtcA;* *MyPtcC could not be imported from other modules;*	6.2.10
mtc **system** **self**	**mtc.stop;** **map(***myPtc***:** *myPort***, system:***portB***);** *myPort***.send(***m_temp***) to self;**	reference to main test component (executes testcase); reference to test system interface component; reference to actual component	6.2.11

PORTS	*EXAMPLES*	DESCRIPTION	§
type port *PortTypeIdentifier* **message** "**{**" [**address** *Type* "**;**"] [**map param** "**(**" {*FormalValuePar* [","]}+ "**)**"] [**unmap param** "**(**" {*FormalValuePar* [","]}+"**)**"] {(**in** | **out** | **inout**) {*MessageType* [","]}+ "**;**"} "**}**"	**type port** *MyPortA* **message {** **in** *MyMsgA***;** **out** *MyMsgB, MyMsgC***;** **inout** *MyMsgD***};** **type port** *MyPortB* **message {inout all};**	asynchronous communication; incoming messages to be queued; messages to send out; message allowed in both directions; all types allowed at *MyPortB*; NEW: with address type and param	6.2.9
type port *PortTypeIdentifier* **procedure** "**{**" [**address** *Type* "**;**"] [**map param** "**(**" {*FormalValuePar* [","]}+ "**)**"] [**unmap param** "**(**"{*FormalValuePar* [","]}+ "**)**"] {(**in** | **out** | **inout**) {*Signature* [","]}+ "**;**"} "**}**"	**type port** *MyPortA* **procedure {** **out** *MyProcedureB***;** **in** *MyProcedureA* **}**	synchronous communication; to call remote operation (get replies/exceptions); to get calls from other components (and sent replies/ exceptions); NEW: with address type and param	

PROCEDURE SIGNATURES	*EXAMPLES*	DESCRIPTION	§
signature *SignatureIdentifier* "**(**" {[**in** | **inout** | **out**] *Type ValueParIdentifier* [","]} "**)**" [(**return** *Type*) | **noblock**] [**exception** "**(**" *ExceptionTypeList* "**)**"]	**signature** *MyProcedureA* **(in integer** *p_myP1, …***) return** *MyType* **exception (***MyTypeA, MyTypeB***);** **signature** *MyProcedureB* **(inout integer** *p_myP2,* **…) noblock;**	caller blocks until a reply or exception is received; caller does not block;	14 22.1.2

A.3 Basic and user-defined data types

BASIC TYPES	*SAMPLE VALUES AND RANGES*		*SAMPLE SUB-TYPES*	SUBTYPES	§
boolean	**true, false**		**type boolean** *MyBoolean* **(true);**	list	6.1.0, 6.1.2
integer	(**-infinity..**-1), 0, 1, (2 **.. infinity**),		**type integer** *MyInteger* (-2, 0, 1..**3);**		
	(**!**-1 .. 30)	-1 excluded		list, range	
float	(**-infinity..** -2.783), 0.0, 2.3E-4,		**type float** *MyFloat* (1.1 .. **infinity**);		
	(1.0..3.0, **not_a_number**)	value greater than **infinity**			
charstring	"<empty>" , "any", "\""	single quote-symbol: ";	**type charstring** *MyISO646* **length**(1);	list, range, length	6.1.1, 6.1.2, E.2
	*v_ myCharstring***[0];** **lengthof(***v_ myCharstring***);**	first character of string; length of string			
universal charstring	**char**(0,0,3,179) & "more"	gamma (γ) in ISO/IEC 10646 (UTF32)	**type universal charstring** *bmpstring* (**char** (0,0,0,0) **.. char** (0,0,255,255))		
bitstring	**'<**empty**>**'B, '1'B, '0101'B		**type bitstring** *OneBit* **length**(1);	list, length	
hexstring	**'<**empty**>**'H, 'a'H, '0a'H, '123a'H, '0A'H		**type hexstring** *OneByte* **length**(2);		
octetstring	**'<**empty**>**'O, '00'O, '0a0b'O & '0A'O		**type octetstring** *MyOctets* ('AA'O,'BB'O);		

SPECIAL TYPES	*EXAMPLES*	DESCRIPTION	§
default	**var default** *v_myAltstep;*	manage use of altstep (activate/deactivate)	6.2.8
address	**var address** *v_myPointer;*	reference of component or SUT interface (global scope)	6.2.12
verdicttype	**var verdicttype** *v_myVerdict***;**	fixed values: none, pass, inconc, fail, error	6.1.0

STRUCTURED TYPES AND ANYTYPE	*SAMPLE VALUES*	*SAMPLE USAGE*		SUBTYPES	§
type record *MyRecord* {**float** *field1*, *MySubrecord1 field2* **optional** }; **type** *MyRecord.field1 Field1*; **type** *MyRecord MyRecord2* ({*field1*:=1.0, *field2*:=**omit**});	**var** *MyRecord v_record* := {2.0, **omit**}; **var** *MyRecord v_record1* := {*field1* := 0.1}; **var** *MyRecord v_record2*:= {1.0, {*c_c1*, *c_c2*}}; **var** *Field1 v_float* := *v_record.field1*; **var** *MyRecord2 v_rec* := {1.0,**omit**};	*v_record.field1* **sizeof**(*v_ record1*) **ispresent**(*v_record .field2*) *v_record2*:= {1.0, **-**};	2.0 1 **false** (unchanged)	list	6.2.1 6.2.1.1 6.2.13.2
type record of integer *MyNumbers*; **type record length**(3) **of** **float** *MyThree*; **type integer** *MyArray* [2] [3];	**var** *MyNumbers v_myNumbers* := {1, 2, 3, 4}; **var** *MyThree v_three* := {1.0, 2.3, 0.4}; **var** *MyArray v_array* := { {1,2,3}, {4,5,6}}; **var** *MyNumbers*[**-**] *MyInteger* := 1;	**lengthof**(*v_myNumbers*) *v_ myNumbers* [1] *v_array* [0], *v_array* [1] [1]	4 2 {1,2,3} 5	list, length	6.2.3 6.2.7 6.2.3.2
type set *MyElements* {*MyRecord element1*, **float** *element2* **optional** }; **type set of** *OneBit MySet*;	**var** *MyElements v_myElements* := {*element1*:=*v_record*, *element2*:=**omit**}; **var** *MySet v_bits* := {'1'B, '0'B};	*v_ myElements.element2* *v_bits*[0]	 '1'B		6.2.2 6.2.3
type enumerated *MyKeywords* {*e_key1*, e_*key2*, e_ *key3*}; **type enumerated** *MyTags* {e_*keyA*(2), e_*keyB*(1)};	**var** *MyKeywords v_enum* := e_*key1*; **var** *MyTags v_enum2* := e_*keyA*;	**type** *MyKeywords MyShortList* {e_*key1*, e_*key3*}; /* e_*key1* < e_*key2* < e_*key3*, e_*keyA* > e_*keyB* */		list	6.2.4
type union *MyUnionType* {**integer** *alternative1*, **float** *alternative2* }	**var** *MyUnionType v_myUnion* := {*alternative1* := 1}	*v_myUnion.alternative2* // error **ischosen**(*v_myUnion.alternative1*) // *true*			6.2.5 C.3.2
anytype /* union of all types within a single module */ **type anytype** *MyAnyType* {(**integer**:= 22),(**boolean**:=**false**), ...}	**var anytype** *v_any* := {**integer**:= -1}; **var** *MyAnyType v_myAny* := {**integer**:= 22};	*v_any*.**integer**; *v_ myAny*.**integer**; *v_ myAny*.**boolean**;	*-1* 22 *n/a (error)*		6.2.6

A.4 Data values and templates

TEMPLATE	*EXAMPLES*	DESCRIPTION	§
template [*TemplateRestriction*] *Type TemplateIdentifier* ["("*TemplateOrValueFormalParList*")"] [**modifies** *TemplateRef*] ":=" *TemplateBody*	**template** *MyTwoInteger mw_subtemplate* (**template integer** *p_1*) := {1, *p_1*}; **var template** *MyRecord* *mw_template* := {**omit**, *mw_subtemplate*(1)}; **var** *MyRecord v_value* := **valueof** (*mw_template*); **isvalue**(*mw_template*); **template bitstring** *m_bits* := '010'**B** & **?** **length**(1);	template with parameter; parameter allows template expressions (e.g. wildcards); template variable; error in case of unspecific content, e.g. wildcards; returns true, if *mw_template* only contains concrete value or "omit"; concatenation results in string of four bits;	 15.3 15.10 C.3.3 15.11
	var template (**omit**) *MyType m_t1*; **var template** (**value**) *MyType m_t2;* **var template** (**present**) *MyType m_t3*;	contain specific values or omit; contain specific values or **omit** (complete template cannot resolve to **omit**); contains unspecific values, except **ifpresent;**	 15.8

TYPE	send/call/reply/raise *TEMPLATES* *(concrete values only)*	receive/getcall/getreply/getraise *TEMPLATES* *(can contain wildcards)*	§
type record *MyRecord* {**float** *field1*, *Mysubrecord1 field2* **optional** };	**template** *MyRecord m_record* := {*field1*:=*c_myfloat*, *field2*:= **omit** }; **template** *MyRecord md_record* **modifies** *m_record* := {*field1*:= 1.0 + *f_float1*(*v_var*)}; **template** *MyRecord m_record2* (**float** *p_f1*):= { *p_f1*} **with** {**optional** "implicit omit" };	**template** *MyRecord mw_record* := {(1.0..1.9), **?** }; **template** *MyRecord mw_record1* := {(1.0, 3.1), * }; **template** *MyRecord mdw_record1* **modifies** *mw_record* := { *field2*:= **omit** }; **template** *MyRecord mw_record2* := {**complement**(0.0), *mw_sub* **ifpresent** };	15.1 15.5 15.7 27.7
type record of integer *MyNums*; **type integer** *MyArray* [2] [5];	**template** *MyNums m_nums* := {0,1,2,3,4}; **template** *MyArray m_array* := { {1,2,3}, {1,2,3,4}};	**template** *MyNums mw_nums* := {0,1, **permutation**(2,3,4)}; **template** *MyArray mw_array* := { {1,2,?}, **?** **length**(2) };	15.6.3 15.7.3 15.7.4

type set *MySet* {**boolean** *field1*,
charstring *field2*};
type set of integer (1..3) *MyDigits*;
	template *MySet m_set* := {**true**, "c" };
	
	template *MyDigits m_digits* := {3,2};
		template *MySet mw_set* := {**false**, ("a"..."f")};
		
		template *MyDigits mw_digits* :=
		superset(**all from** *m_digits*); // NEW *argument*
		template *MyDigits mw_digits2* := **subset**(1,?);
			15.7.2
			
			B.1.2.6
			
			B.1.2.7

signature *MyProcedure* (**in integer** *p1*, **out integer** *p2*);	**template** *MyProcedure s_callProc*:= {1, **omit**}; **template** *MyProcedure s_replyProc*:= {**omit**, 2};	**template** *MyProcedure s_expectedCall* := {?, **omit**}; **template** *MyProcedure s_expectedReply* := {**omit**, ?};	15.2

CHARACTER STRING PATTERN	DESCRIPTION	§
template charstring *mw_template*:= **pattern** "ab??xyz*0" ; **template universal charstring** *mw_templ* := **pattern** "a*z" **length**(2..10);	string 'ab' followed by any two characters, 'xyz', none or any number of characters, followed by '0'; up to eight characters between first and last element;	B.1.5

DECLARATIONS	*EXAMPLES*	DESCRIPTION	§
const *Type* {*ConstIdentifier* [*ArrayDef*] ":=" *ConstantExpression* [","]} [";"]	**const integer** *c_myConst* := 5; **const float** *c_myFloat*[2] := {0.0, 1.2};	constants within type definitions need values at compile-time;	10
var *Type VarIdentifier* [*ArrayDef*] [":=" *Expression*] { [","]*VarIdentifier* [*ArrayDef*] [":=" *Expression*] } [";"]	**var boolean** *v_myVar2* := **true**, *v_myVar3* := **false**;	passed to both value and template-type formal parameters	11.1
var template [*TemplateRestriction*] *Type* *VarIdentifier* [*ArrayDef*] ":=" *TemplateBody* {[","] *VarIdentifier* [*ArrayDef*] ":=" *TemplateBody*} [";"]	**var template integer** *v_myUndefinedInteger* := ?; **var template** (**omit**) *MyRecord v_myRecord* := {*field1* := *c_v1*; *field2* := *v_my1*};	passed as actual parameters to template-type formal parameters	11.2
[*Visibility*] **modulepar** *ModuleParType* {*ModuleParIdentifier* [":=" *ConstantExpression*] ","} *ModuleParIdentifier* [":=" *ConstantExpression*] ";"	**modulepar integer** *PX_PARAM := c_default*; **private modulepar integer** *PX_PAR1*, *PX_PAR2* := 2;	test management value setting overwrites specified default; parameters not importable;	8.2.1

B.1 Functional blocks

TESTCASE	EXAMPLES	DESCRIPTION	§
testcase *TestcaseIdentifier* "**(**" [{(*FormalValuePar* | *FormalTemplatePar*) ["**,**"]}] "**)**" **runs on** *ComponentType* [**system** *ComponentType*] *StatementBlock*	**testcase** *TC_myTest* (**in integer** *p_myp1*, **out float** *p_myp2*) **runs on** *MyPtcA* **system** *MySUTinterface* **{const integer** *c_local*; **...}**	behaviour of the **mtc**; component type of **mtc**; test system interface comp.; *c_local* for local use only;	16.3

FUNCTION	EXAMPLES	DESCRIPTION	§

function *FunctionIdentifier*
"**(**" [{ (*FormalValuePar* | *FormalTimerPar*
| *FormalTemplatePar* | *FormalPortPar*) ["**,**"]}] "**)**"
[**runs on** *ComponentType*]
[**return** [**template**] *Type*]
StatementBlock
	function *f_myFunctionPtcA*
	(**template in** *MyTemplateType*) **runs on** *MyPtcA*
	return template *MyTemplateType*
	{timer *t_local* := 1.0; ...**};**
	
	function *f_myFctNoRunsOn* (**in integer** *p_myp*:=1)
	return template *MyTemplateType* {...**};**
	var template *MyTemplateType*
	v_generictempate := *f_myFctNoRunsOn*(**2**);
	v_generictempate := *f_myFctNoRunsOn*();
		invoke from components
		equivalent to *MyPtcA*, parameter
		allows wildcards;
		timer for local use only;
		
		can be called from any place
		(no "runs on");
		
		invoke *f_myFctNoRunsOn*;
		invoke with default of *p_myp*;
			16.1
			
			5.4.1.1

ALTSTEP	EXAMPLES	DESCRIPTION	§
altstep *AltstepIdentifier* "**(**" [{(*FormalValuePar* | *FormalTimerPar* | *FormalTemplatePar*| *FormalPortPar*) ["**,**"]}] "**)**" [**runs on** *ComponentType*] "**{**" {(*VarInstance* | *TimerInstance*| *ConstDef* |*TemplateDef*) ["**;**"]} *AltGuardList* "**}**"	**altstep** *a_default* (**in timer** *p_timer1*) **runs on** *MyPtcA* **{var boolean** *v_local*; [] *pco1*.**receive {repeat}** [] *p_timer1*.**timeout {break}** **...}** **var default** *v_firstdefault*; *v_firstdefault* := **activate**(*a_default*(*t_local*)); **deactivate**(*v_firstdefault*);	use definitions from *MyPtcA;* variable for local use only; start re-evaluation of altstep; exit from altstep; variable to handle default; (default) altstep activation;	16.2 20.3 19.12 6.2.8 20.5.2 20.5.3

B.2 Typical programming constructs

BRANCHES, LOOPS, ASSIGNMENTS	EXAMPLES	DESCRIPTION	§
if "**(**" *BooleanExpression* "**)**" *StatementBlock* {**else if** "**(**" *BooleanExpression* "**)**" *StatementBlock*} [**else** *StatementBlock*]	**if** (*v_myBoolean*) **{...} else {...};**		19.2
select "**(**" *SingleExpression* "**)**" "**{**" {**case** "**(**" { *SingleExpression* ["**,**"]} "**)**" *StatementBlock*} [**case else** *StatementBlock*] "**}**"	**select** (*v_myString*) **{** **case(** "blue" **) {...}** **case** ("red" **){...}** **case else {...}};**	selector can be of other type, e.g. integer, enumerated;	19.3
for "**(**" (*VarInstance*| *Assignment*) "**;**" *BooleanExpression* "**;**" *Assignment* "**)**" *StatementBlock*	**for** (**var integer** *v_ct* :=1; *v_ct* <8; *v_ct*:= *v_ct* +1) **{ ... };**	variable *v_ct* not defined outside of loop;	19.4
while "**(**" *BooleanExpression* "**)**" *StatementBlock*	**while**(*v_in* == *v_out*) **{...};**		19.5
do *StatementBlock* **while** "**(**" *BooleanExpression* "**)**"	**do** **{...; if(...){break};...}** **while** (*v_in*!=*v_out*) **{...};**		19.6
break;		exit from loop	19.12
continue;		next iteration of a loop	19.13
VariableRef **":="** (*Expression* | *TemplateBody*)	*v_myValue* := *v_myValue2;*	basic assignment	19.1
label *LabelIdentifier*	**label** *myLabel*;	define label location;	19.7
goto *LabelIdentifier*	**goto** *myLabel*;	jump to *myLabel;*	19.8
return [*Expression*]	**return** *v_myValue*;	terminates execution of functions or altsteps	19.10

AUXILLARY STATEMENTS	EXAMPLES	DESCRIPTION	§
match "**(**" *Expression* "**,**" *TemplateInstance* "**)**"	**if (match** ({0,(2..4)}, *mw_rec*)**) {...};**	evaluates expression against template;	15.9
log "**(**" { (*FreeText*| *TemplateInstance*) ["**,**"] } "**)**"	**log(**"expection:", *m_templ*, "ok"**);**	text output for test execution log;	19.11
action "**(**" {(*FreeText* | *Expression*) ["**&**"]} "**)**"	**action** ("Press Enter to continue"**);**	request external action during test execution;	25

VERDICT HANDLING	EXAMPLES	DESCRIPTION	§
verdicttype	**var verdicttype** *v_verdict*;	(none, inconc, pass, fail, error)	24
setverdict "**(**" *SingleExpression* { "**,**" (*FreeText* | *TemplateInstance*)} "**)**"	**setverdict(pass);**	initial value is **none;** change current verdict (could not be improved);	24.2
getverdict;	*v_verdict* := **getverdict;**	retrieve actual component verdict;	24.3

OPERATOR PRECEDENCE *(decreasing from left to right)*															§
par.	sign (unary)	arithmetic operators and string concatenation (&)		bitwise operators				shift rotate	relational operators		logical operators				
(...)	**+** **-**	***** **/** **mod** **rem**	**+** **-** **&**	**not4b**	**and4b**	**xor4b**	**or4b**	**<<** **>>** **<@** **@>**	**<** **>** **<=** **>=**	**==** **!=**	**not**	**and**	**xor**	**or**	7.1

B.3 Port operations and external function

ASYNCHRONOUS COMMUNICATION (send, receive)	EXAMPLES	DESCRIPTION	§
Port "." **send** "(" *TemplateInstance* ")" [**to** *Address*]	*myPort*.**send** (*MyType*: "any string"); *myPort*.**send** (*m_template*) **to** *my_ptc1*; *myPort*.**send** (*m_template*) **to** (*my_ptc1, my_ptc2*); *myPort*.**send** (*m_template*) **to all components**;	inline template; multicast; broadcast;	22.2.1
(*Port* | **any port**) "." **receive** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" *FieldOrTypeReference*][","]} ")"))] [**sender** *VariableRef*]]	*myPort*.**receive**(*MyType*:**?**) -> **value** *v_in*; *myPort*.**receive**(*mw_template*) **from** *v_interface*; *myPort*.**receive** -> **sender** *v_address*;	store incoming value; sender condition; store originator ref;	22.2.2

QUEUE INSPECTION	EXAMPLES	DESCRIPTION	§
(*Port* | **any port**) "." **trigger** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" *FieldOrTypeReference*]} ")"))] [**sender** *VariableRef*]]	*myPort*.**trigger**(*MyType*:**?**) -> **value** *v_income*;	removes all messages from queue (including the specified message with type *MyType*);	22.2.3
(*Port* | **any port**) "." **check** ["(" (*PortReceiveOp* | *PortGetCallOp* | *PortGetReplyOp* | *PortCatchOp*) | ([**from** *Address*] ["->" **sender** *VariableRef*]) ")"]	*myPort*.**check** (**receive**(*m_template*) **from** *v_myPtc*);	evaluates top element against expectation; no change of queue status;	22.4

SYNCHRONOUS COMMUNICATION (call, reply), EXCEPTIONS (raise, catch)	EXAMPLES	DESCRIPTION	§
Port "." **call** "(" *TemplateInstance* ["," *CallTimerValue*] ")" [**to** *Address*]	**signature** *MyProcedure* (**in integer** *p_myP1*, **inout float** *p_ myP2*) **return integer** **exception** (*ExceptionType*); *myPort*.**call** (*s_template*, 5.0) {... [] *myPort*.**getreply** (*s_template* **value** (1..9)) {...} [] *myPort*.**getreply** (*s_template2*) **from** *v_interface* -> **value** *v_ret* **param** (*v_myVar* := *p_ myP2*) **sender** *v_address* {...} ... [] *myPort*.**catch** (*ExceptionType*:?) {...} ... [] *myPort*.**catch** (**timeout**) {...} }; *myPort*.**getcall** (*s_myExpectation*); ... **if** (*v_failure*) { *myPort*.**raise**(*s_myError*);...}; ... *myPort*.**reply** (*s_myAnswer*)		22.3.1
(*Port* | **any port**) "." **getcall** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**param** "(" {(*VariableRef* ":=" *ParameterIdentifier*)"," } |{(*VariableRef* | "-") ","} ")"] [**sender** *VariableRef*]]		calling component: implicit timeout of 5 sec. return value must be 1..9;	22.3.2
(*Port* | **any port**) "." **getreply** ["(" *TemplateInstance* [**value** *TemplateInstance*] ")"] [**from** *Address*] ["->" [**value** *VariableRef*] [**param** "(" {(*VariableRef* ":=" *ParameterIdentifier*)"," } | {(*VariableRef* | "-") ","} ")"] [**sender** *VariableRef*]]		*v_ret* gets return value; *v_myVar* gets "out"-value of parameter *p_myP2;* remote exception raised; local timeout of implicit timer;	22.3.4
Port "." **reply** "(" *TemplateInstance* [**value** *Expression*] ")" [**to** *Address*]			22.3.3
Port "." **raise** "(" *Signature* "," *TemplateInstance* ")" [**to** *Address*]		called component:	22.3.5
(*Port* | **any port**) "." **catch** ["(" (*Signature* "," *TemplateInstance*) | "**timeout**"")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" *FieldOrTypeReference*] [","]} ")"))] [**sender** *VariableRef*]]		raise exception regular reply	22.3.6

PORT OPERATIONS	EXAMPLES	DESCRIPTIONS	§
(*Port* | (**all port**)) "." **start**	*myPortA*.**start**	clear queue and enables communication.	22.5
(*Port* | (**all port**)) "." **stop**	*myPortA*.**stop**	disables queue for sending and receiving events.	
(*Port* | (**all port**)) "." **halt**	*myPortA*.**halt**	disables sending and new incoming elements; current elements processed.	
(*Port* | (**all port**)) "." **clear**	*myPortA*.**clear**	removes all current elements from the port queue	
(*Port* | (**all port**) | (**any port**)) "." **checkstate** "(" *SingleExpression* ")"	*myPortA*.**checkstate** ("Mapped")	examine the state of a port, arguments: "Started", "Halted", "Stopped", "Connected", "Mapped", "Linked" NEW	22.5.5 E.2.2.4

EXTERNAL CALCULATION	EXAMPLES	DESCRIPTION	§
external function *ExtFunctionIdentifier* "(" [{(*FormalValuePar* | *FormalTimerPar* |*FormalTemplatePar*| *FormalPortPar*) [","]}] ")" [**return** *Type*]	**external function** *fx_myCryptoalgo* (**integer** *p_name, ...*) **return charstring**;	need implementation in platform adapter	16.1.3

B.4 Timer and alternatives

TIMER DEFINITIONS AND OPERATIONS	*EXAMPLES*	DESCRIPTION	§
timer {*TimerIdentifier* [*ArrayDef*] ":=" *TimerValue* [","] } [";"]	**timer** *t_myTimer* := 4.0; **timer** *t_myTimerArray*[2] := {1.0, 2.0};	declaration with default; array of two timers;	12 23.2
((*TimerIdentifier* | *TimerParIdentifier*) {"[" *SingleExpression* "]"}) "." **start** ["(" *TimerValue* ")"]	*t_ myTimer*.**start**(5.0); *t_ myTimer*.**start**;	timer started for 5 seconds; restart for 4 sec. (default);	23.4
(((*TimerIdentifier* | *TimerParIdentifier*) {"[" *SingleExpression* "]"}) | **all timer**) "." **stop**	**var float** *v_current* := *t_ myTimer*.**read**; *t_ myTimerArray*[1].**stop**;	get actual timer value; stop 2nd timer from array;	23.3
(((*TimerIdentifier* | *TimerParIdentifier*) {"[" *SingleExpression* "]"}) | **any timer**) "." **running**	**if (any timer.running)** **{...}**	any timer previously started;	23.5
(((*TimerIdentifier* | *TimerParIdentifier*) {"[" *SingleExpression* "]"}) | **any timer**) "." **timeout**	*t_ myTimer*.**timeout**;	awaits timeout of *t_ myTimer*	23.6

ALTERNATIVES	*EXAMPLES*	DESCRIPTION	§

alt "{"
{"[" [*BooleanExpression*] "]"
((*TimeoutStatement* | *ReceiveStatement* | *TriggerStatement*
| *GetCallStatement* | *CatchStatement* | *CheckStatement*
| *GetReplyStatement* | *DoneStatement* | *KilledStatement*)
StatementBlock) | (*AltstepInstance* [*StatementBlock*])
} ["[" **else** "]" *StatementBlock*]
"}"
	alt {
	[*v_flag*==**true**] *myPort*.**receive** {... }
	[] *myComponent*.**done** {...; **repeat** }
	[] *myComponent*.**killed** {...}
	[*v_integer* > 1] *a_myAltstep*() {...}
	[*t_timer1*.**running**]
	any timer.timeout {...}
	...
	[**else**] {...}
	}
		alternative with condition;
		start re-evaluation of alt;
		component not alive;
		altstep alternatives;
		condition that timer is running;
		
		if non of the previous alternatives matches;
			20.2

interleave "{" {"[]" (*TimeoutStatement* | *ReceiveStatement* | *TriggerStatement* | *GetCallStatement* | *CatchStatement* | *CheckStatement* | *GetReplyStatement* | *DoneStatement* | *KilledStatement*) *StatementBlock*} "}"	**interleave** { [] *myPort1*.**receive** {...} [] *myPort2*.**receive** {...} ...}	all alternatives must occur, but in arbitrary order	20.4

B.5 Dynamic configuration

COMPONENT MANAGEMENT	*EXAMPLES*	DESCRIPTION	§
ComponentType "." **create** ["("*Expression* ["," *Expression*] ")"] [**alive**]	**var** *MyPtc myInstance*, *myInstance2*; *myInstance* := *MyPtc*.**create alive**; *myInstance2* := *MyPtc*.**create**("ID"); *myInstance*.**start**(*f_myFunction* (*v_param1,c_param2*)); *myInstance*.**stop**; *myInstance*.**start**; *myInstance*.**kill**; **all component.kill**; **stop**; **self.stop**; **mtc.stop**;	initialize variables; allocate memory, "ID" for logging only; start with *f_myFunction* behaviour; stops (**alive** keeps resources); restart; stop and release resources; kills PTCs (remove resources); stops own component; stops testcase execution;	21.3.1 21.3.2 21.3.3 21.3.4
(*VariableRef* | *FunctionInstance*) "." **start** "(" *FunctionInstance*")"			
stop | ((*VariableRef* | *FunctionInstance*| **mtc** | **self**) "." **stop**) | (**all component** "." **stop**)			
kill | ((*VariableRef* | *FunctionInstance*| **mtc** | **self**) "." **kill**) | (**all component** "." **kill**)			
(*VariableRef* | *FunctionInstance*| **any component** | **all component**) "." (**alive** | **running** | **done** | **killed**)	*myInstance*.**alive**; *myInstance*.**running**; **all component.done**; **any component.killed**;	checks status of specific, **any** or **all** components;	21.3.5 21.3.6 21.3.7 21.3.8
testcase "." **stop** {(*FreeText* | *TemplateInstance*) [","]} ")"	**testcase.stop** ("Unexpected case");	stop with **error** verdict;	21.2.1

PORT ASSOCIATIONS	*EXAMPLES*	DESCRIPTION	§
map "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")" [**param** "(" [{*ActualPar* [","] }+] ")"]	**map**(*myPtc*:*portA*, **system**:*portX*); **map**(**mtc**:*portA*, **system**:*portB*);	assign to SUT via adapter	21.1.1
unmap [("(*ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")") | [**param** "(" [{*ActualPar* [","] }+] ")"] | ("(" *PortRef* ")") [**param** "(" [{ActualPar [","] }+] ")"] | ("(*ComponentRef* ":" **all port** ")") | ("(" **all component** ":" **all port** ")")]	**unmap**; **unmap**(**mtc**:*portA*, **system**:*portB*);	unmaps all own port; unmaps mtc *portA;*	21.1.2
connect "(" *ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")"	**connect**(**self**:*portA*, **mtc**:*portC*)	between components w/o adapter;	21.1.1
disconnect [("(*ComponentRef* ":" *Port* "," *ComponentRef* ":" *Port* ")") | ("(*PortRef* ")") | ("(*ComponentRef* ":" **all port** ")") | ("(" **all component** ":" **all port** ")")]	**disconnect**(**mtc**:*portA*, *myPtc*:*portB*); **disconnect**;	disconnect mtc portA; all connections of actual component;	21.1.2

C.1 Predefined functions and useful types

PREDEFINED CONVERSION FUNCTIONS	*EXAMPLES*		§
int2char| **int2unichar**|**int2str**|**int2float** **(in integer** *invalue***)** **return (**charstring| universal charstring|charstring|float) **int2bit**| **int2hex**| **int2oct** **(in integer** *invalue*, **in integer** *length***)** **return (**bitstring| hexstring| octetstring) **int2enum** **(in integer** inpar, **out** Enumerated_type outpar)	**int2str**(-66); **int2float**(4); **int2bit**(4,4); **int2bit**(4,2); **int2enum**(0, *v_myEnum*);	"-66" 4.0 '0100'B **error** *e_FirstElement*	16.1.2 C.1.1- C.1.8
float2int **(in float** *invalue***) return** integer	**float2int**(3.12345E2)	312	C.1.9
char2int| **char2oct** **(in charstring** *invalue***)** **return (**integer| octetstring)	**char2oct**("T")	'54'O	C.1.10 C.1.11
unichar2int **(in universal charstring** *invalue***) return** integer	**unichar2int**("T")	44	C.1.12
bit2int|**bit2hex**|**bit2oct**| **bit2str** **(in bitstring** *invalue***)** **return (**integer| hexstring| octetstring|charstring)	**bit2hex**('111010111'B)	'1D7'H	C.1.13- C.1.16
hex2int| **hex2bit**| **hex2oct**| **hex2str** **(in hexstring** *invalue***)** **return (**integer| bitstring| octetstring| charstring)	**hex2str**('AB801'H)	"AB801"	C.1.17- C.1.20
oct2int| **oct2bit**| **oct2hex**|**oct2str**|**oct2char** **(in octetstring** *invalue***)** **return (**integer|bitstring|hexstring|charstring| charstring)	**oct2bit** ('D7'O)	'11010111'B	C.1.21- C.1.25
str2int|**str2hex**|**str2oct**| **str2float** **(in charstring** *invalue***)** **return (**integer| hexstring |octetstring | float)	**str2oct**("1D7")	'01D7'O	C.1.26- C.1.29
enum2int **(in** Enumerated_type inpar**) return** integer	**enum2int**(*e_FirstElement*)	0	C.1.30

OTHER PREDEFINED FUNCTIONS	*EXAMPLES*		DESCRIPTION	§
lengthof **(in template (present)** any_string_or_list_type *inpar***)** **return** integer	**lengthof**('1??1'B)	4	return length of value or template of any string type, **record of**, **set of** or **array**	C.2.1
sizeof **(in template (present)** any_record_set_type *inpar***)** **return** integer	**sizeof**(*MyRec*:{1,**omit**}) **sizeof**(*MyRec*:{1,**2**})	1 2	return number of elements in a value or template of **record** or **set**	C.2.2
ispresent **(in template** any_type *inpar***)** **return** boolean	**ispresent**(*v_myRecord.field1*)		true if optional field *inpar* is present (**record** or **set** only)	C.3.1
ischosen **(in template** any_union_type *inpar***)** **return** boolean	**ischosen**(*v_receivedPDU.field2*)		true if *inpar* is chosen within the **union** value/template	C.3.2
isvalue **(in template** any_type *inpar***) return** boolean**;**	**isvalue**(*MyRec*:{1,**omit**})	**true**	true if concrete value	C.3.3
isbound **(in template** any_type *inpar***) return** boolean**;**	**isbound** (*v_myRecord*)		true if *inpar* is partially initialized NEW	C.3.4
rnd **([in float** *seed***]) return** float**;**	*v_randomFloat* **:= rnd** ();		random float number	C.6.1
testcasename **() return charstring**;	*v_TCname* **:= testcasename** ();		current executing test case	C.6.2

STRING MANIPULATION	*EXAMPLES*	DESCRIPTION	§
substr **(in template (present)** any_string_or_sequence_type *inpar***,** **in integer** index**, in integer** count **)** **return** input_string_or_sequence_type	**substr** ("test", 1, 2)	equal to "es"	C.4.2
replace (in any_string_or_sequence_type *inpar***,** **in integer** *index***, in integer** *len***,** **in** any_string_or_sequence_type *repl* **)** **return** any_string_or_sequence_type	**replace** ("test", 1, 2,"se")	equal to "tset"	C.4.3
regexp **(in template (value)** any_character_string_type *inpar***,** **in template (present)** any_character_string_type *expression***,** **in integer** *groupno***)** **return** any_character_string_type	*v_string* := " alp beta gam delta "; *v_template* := "(?+)(gam)(?+)"; **regexp** (*v_string*, *v_template*, 2);	three groups; group #2 is equal to " delta "	C.4.1
encvalue(in template (value) any_type *inpar***)** **return bitstring**	*p_messagelen* **:=** **lengthof(encvalue(***m_payload***));**	functions require codec implementation;	C.5.1
decvalue **(inout bitstring** encoded_value, **out any_type** decoded_value **)** **return integer**	**decvalue**(*v_in*, *v_out*)	**decvalue** return indicates success (0), failure (1), uncompletion (2);	C.5.2

TYPE DEFINTIONS FOR SPECIAL CODING (USE WITH OTHER NOTATIONS: ASN.1, IDL, XSD)	DESCRIPTION	§
type charstring *char646* **length (1);**	single character from ITU T.50	E.2.4.1
type universal charstring *uchar* **length (1);**	single character from ISO/IEC 10646	E.2.4.2
type bitstring *bit* **length (1);**	single binary digit	E.2.4.3
type hexstring *hex* **length (1);**	single hexdigit	E.2.4.4
type octetstring *octet* **length (1);**	single pair of hexdigits	E.2.4.5
type integer *byte* **(**-128 .. 127**) with {variant** "8 bit"**};** **type integer** *unsignedbyte* **(**0 .. 255**) with {variant** "unsigned 8 bit"**};**	to be encoded / decoded as they were represented on 1 byte	E.2.1.0
type integer *short* **(**-32768 .. 32767**) with {variant** "16 bit"**};** **type integer** *unsignedshort* **(**0 .. 65535**) with {variant** "unsigned 16 bit"**};**	to be encoded / decoded as they were represented on 2 bytes	E.2.1.1
type integer *long* **(**-2147483648 .. 2147483647**) with {variant** "32 bit"**};** **type integer** *unsignedlong* **(**0 .. 4294967295**) with {variant** "unsigned 32 bit"**};**	to be encoded / decoded as they were represented on 4 bytes	E.2.1.2
type integer *longlong* **(**-9223372036854775808 .. 9223372036854775807**) with {variant** "64 bit"**};** **type integer** *unsignedlonglong* **(**0 .. 18446744073709551615**) with {variant** "unsigned 64 bit"**};**	to be encoded / decoded as they were represented on 8 bytes	E.2.1.3
type float *IEEE754float* **with {variant** "IEEE754 float"**};** **type float** *IEEE754double* **with {variant** "IEEE754 double"**};** **type float** *IEEE754extfloat* **with {variant** "IEEE754 extended float"**};** **type float** *IEEE754extdouble* **with {variant** "IEEE754 extended double"**};**	to be encoded / decoded according to the IEEE 754	E.2.1.4
type universal charstring utf8string **with {variant** "UTF-8"**};**	encode / decode according to UTF-8	E.2.2.0
type universal charstring *iso8859string* **(char** (0,0,0,0) **.. char** (0,0,0,255)**) with {variant** "8 bit"**};**	all characters defined in ISO/IEC 8859-1	E.2.2.3
type universal charstring *bmpstring* **(char** (0,0,0,0) **.. char** (0,0,255,255)**) with {variant** "UCS-2"**};**	BMP character set of ISO/IEC 10646	E.2.2.1
type record *IDLfixed* **{unsignedshort** *digits***, short** *scale***, charstring** *value_***}** **with {variant** "IDL:fixed FORMAL/01-12-01 v.2.6"**};**	fixed-point decimal literal as defined in the IDL Syntax and Semantics version 2.6	E.2.3.0

C.2 Optional definitions: Control part and attributes

CONTROL PART	EXAMPLES	DESCRIPTION	§

control "{"
{ (*ConstDef* | *TemplateDef* | *VarInstance*| *TimerInstance*
|*TimerStatements* |*BasicStatements* | *BehaviourStatements*
| *SUTStatements* | **stop**) [";"]}
"}"
[*WithStatement*] [";"]
	control {
	...
	var verdicttype *v_myverdict1* :=
	execute (*TC_testcase1(c_value)*, 3.0);
	if (**execute** (*TC_testcase2*() != **pass**)
	{**stop**};
	}
		implicit timeout value 3.0 s;
		termination of control part;
			26.2

execute "(" *TestcaseRef* "(" [{*ActualPar* [","]}] ")" ["," *TimerValue* ["," *HostId*]] ")"			26.1

ATTRIBUTES	EXAMPLES	DESCRIPTION	§
with "{" { (**encode** | **variant** | **display** | **extension** | **optional**) [**override**] ["(" *DefinitionRef* | *FieldReference* | *AllRef* ")"] *FreeText* [";"]} "}"	**group** *g_typeGroup* { **type integer** *MyInteger* **with** {**encode** "rule 2"}; **type record** *MyRec* {**integer** *field1*, **integer** *field2* **optional**} **with** {**variant** (*field1*) "rule3"}; **type integer** *MyIntType* **with** { **display** "mytext for GFT"} } **with** (**encode** "rule1"; **extension** "Test purpose 1");	apply rule2 to *MyInteger*; apply rule3 to *field1*; GFT format details; apply rule1/extension to group;	27.2
	template *MyRec m_myRec* := {*field1* := *1*} **with** {**optional** "implicit omit"}; **template** *MyRec m_myRec2* := {*field1* := *1*} **with** {**optional** "explicit omit"};	*field2* is set to omit; *field2* is undefined;	27.7

C.3 Character pattern

META-CHARACTER	DESCRIPTION		EXAMPLES	§
?	single character		a, 1	B.1.5
*	any number of any characters		1, 1111saaa	
\d	single numerical digit		0, 1, ... 9	
\w	single alphanumeric character		0,... 9, a,...z, A,...Z	
\q{*a,b,c,d*} **\t** **\n** **\r** **\s**	any universal character	characters according to ITU-T Recommendation T.50	**char**(0,0,3,179) = gamma (γ) in ISO/IEC 10646 HT(9) LF(10), VT(11), FF(12), CR(13) CR HT(9), LF(10), VT(11), FF(12), CR(13), SP(32)	A.1.5.1
	control character HT(9)			
	newline character			
	control character CR			
	whitespace character			
\b	word boundary (any graphical character except SP or DEL is preceded or followed by any of the whitespace or newline characters)			
\" **""**	one double-quote-character		**\"**, **""**	
****	Interpret a meta-character as a literal		***a*** refers to the characters ***a*** only **\[** refers to the single character **[** only	
{reference}	reference to existing definitions, e.g. **const charstring** *c_mychar* := "ac";		*"{c_mychar}"* refers to "ac"	
\N{reference}	reference to existing character set definitions e.g. **const charstring** *c_myset*:= ("a.."c");		*"{c_myset}"* refers to "a", "b" or "c" only	
[ ]	any single character of the specified set		[1s3] allows 1, s, 3	
-	range within a specified set		[a-d] allows a,b,c, d	
^	exclude ranges within a specified set		[^a-d] allows any character except a,b,c,d	
|	Used to denote two alternative expressions		(a|b) indicates "a" or "b"	
( )	Used to group an expression			
#(n, m)	repetition of previous expression	min. n-times, max. m-times	*d#(2,4)* indicates "dd", "ddd" or "dddd"	
#n		n-times repetition	*d#3* indicates "ddd"	
+		optional	*d+* indicates "d", "dd", "ddd", ...	

C.4 Preprocessing macros

MACRO NAME	DESCRIPTION	EXAMPLES	§

__MODULE__	occurrences are replaced with module name (charstring value)	**module** *MyTest* {
		const charstring
		c_myConst := **__MODULE__** & ":" & **__FILE__**;
		// becomes "MyTest:/home/mytest.ttcn"
		const charstring
		c_myConst2 := **__LINE__** & ":" & **__BFILE__**;
		// becomes "6:mytest.ttcn"
		}
			D.1 - D.4

__FILE__	occurrences are replaced with full path and basic file name (charstring value)		
__BFILE__	occurrences are replaced with basic file name (charstring value without path)		
__LINE__	occurrences are replaced with the actual line number (integer value)		

__SCOPE__	occurrences are replaced with (charstring value):
	• *module* name
	• **'control'**
	• *component* type
	• *testcase* name
	• *altstep* name
	• *function* name
	• *template* name
	• *type* name
	if lowest named scope unit is...
	...module definitions part
	...module control part
	...component type definition
	...test case definition
	...altstep definition
	...function definition
	...template definition
	...user-defined type
		module *MyModule* {
		const charstring *c_myConst* := **__SCOPE__**;
		// value becomes "MyModule"
		template charstring
		m_myTemplate := **__SCOPE__**;
		// value becomes "m_myTemplate"
		function *f_myFunc* () := {
		log (**__SCOPE__**)} // output "f_myFunc"
		}
			D.5

D.1 Generic Naming Conventions

The following table is derived from ETSI TS 102 995 (see http://www.ttcn-3.org/index.php/development/naming-convention for more examples).

LANGUAGE ELEMENT			PREFIX	EXAMPLES	NAMING CONVENTION
module			*none*	*MyTemplates*	upper-case initial letter
data type (incl. component, port, signature)				*SetupContents*	
group within a module				*messageGroup*	lower-case initial letter(s)
port instance				*signallingPort*	
test component instance				*userTerminal*	
module parameters				*PX_MAC_ID*	all upper case letters (consider test purpose list)
test case			***TC_***	*TC_G1_SG3_N2_V1*	
TSS group			***TP_***	*TP_RT_PS_TR*	
template	message		***m_***	*m_setupInit*	lower-case initial letter(s)
		with wildcard or matching expression	***mw_***	*mw_anyUserReply*	
	modifying		***md_***	*md_setupInit*	
		with wildcard or matching expression	***mdw_***	*mdw_anyUserReply*	
	signature		***s_***	*s_callSignature*	
constants			***c_***	*c_maxRetransmission*	
function			***f_***	*f_authentication()*	
	external		***fx_***	*fx_calculateLength()*	
altstep (incl. default)			***a_***	*a_receiveSetup()*	
variable			***v_***	*v_macId*	
	(defined within a component type)		***vc_***	*vc_systemName*	
timer			***t_***	*t_wait*	
	(defined within a component type)		***tc_***	*tc_authMin*	
formal parameters			***p_***	*p_macId*	
enumerated values			***e_***	*e_syncOk*	

D.2 Documentation tags

The following tables provide summaries only; the complete definitions are provided in ES 201873-10.
Documentation blocks may start with /** and end with */ or start with //* and end with the end of line.

GENERAL TAGS FOR *ALL* OBJECTS	EXAMPLES	DESCRIPTION	§
@author [freetext]	//* **@author** My Name	a reference to the programmer	6.1
@desc [freetext]	//* **@desc** My description about the TTCN-3 object	any useful information on the object	6.3
@remark [freetext]	//* **@remark** This is an additional remark from Mr. X	an optional remark	6.8
@see *Identifier*	//* **@see** *MyModuleX.mw_messageA*	a reference to another definition	6.10
@since [freetext]	//* **@since** version_0.1	indicate a module version when object was added	6.11
@status *Status* [freetext]	//* **@status** *deprecated* because of new version A	samples: *draft, reviewed, approved, deprecated*	6.12
@url uri	//* **@url** http://www.ttcn-3.org	a valid URI, e.g.: file, http, shttp, https	6.13
@version [freetext]	//* **@version** version_0.1	the version of the documented TTCN-3 object	6.15
@reference [freetext]	//* **@reference** ETSI TS xxx.yyy section zzz	a reference for the documented TTCN-3 object	6.18

TESTCASE SPECIFIC TAGS	EXAMPLES	DESCRIPTION	§
@config [freetext]	/** ---------- * **@config** intended for our configuration A * **@priority** *high* * **@purpose** SUT send msg A due to receipt of msg B * **@requirement** requirement A.x.y * ---------- */ **testcase** *TC_MyTest* **() {...}**	a reference to a test configuration	6.2
@priority *Priority*		individual priority	6.16
@purpose [freetext]		explains the testcase purpose	6.7
@requirement [freetext]		a link to a requirement document	6.17

OBJECT SPECIFIC TAGS	EXAMPLES	USED FOR			§
@exception *Identifier* [freetext]	//* **@exception** *MyExceptionType* due to event A	signature			6.4
@param *identifier* [freetext]	//* **@param** *p_param1* input parameter of procedure **signature** *MyProcedure* **(in integer** *p_param1***);**		template,	function, altstep, testcase	6.6
@return [freetext]	//* **@return** this procedure returns an octetstring			function	6.9
@verdict *Verdict* [freetext]	//* **@verdict** *fail* due to invalid parameter **function** *f_myfct1***() {...}**			function, altstep, testcase	6.14
@member *identifier* [freetext]	/** **@member** *tc_myTimer* the timer within *MyPTC* type */ **type component** *MyPTC* **{timer** *tc_myTimer***; ...}**	struct data type, component, port, modulepar, const,	template		6.5

D.3 ASN.1 mapping

The following tables present selected introduction examples only (ASN.1 values are omitted); complete definitions are provided in ES 201873-7.
Additional rules: Replace all "-" with "_", ASN.1 definitions using TTCN-3 keywords append "_" to used keywords

ASN.1 TYPE	TTCN-3 TYPE	*ASN.1 EXAMPLE*	*TTCN-3 EQUIVALENT*	§

BOOLEAN	boolean	Message ::= SEQUENCE {
		version [0] IMPLICIT INTEGER(0..99),
		mId [1] MId,
		messageBody CHOICE {
		messageError [2] ErrorDescriptor,
		transactions [3] SEQUENCE OF Transaction
		} }
			type record *Message* {
			integer *version* (0..99),
			MId mId,
			MessageUnion messageBody
			}
			
			type union *MessageUnion* {
			ErrorDescriptor
			messageError,
			record of *Transaction*
			transactions
			}
				8.1

INTEGER	integer			
REAL	float			
OBJECT IDENTIFIER	objid			
BIT STRING	bitstring			
OCTET STRING	octetstring			
SEQUENCE	record			
SEQUENCE OF	record of			
SET	set			
SET OF	set of			
ENUMERATED	enumerated			
CHOICE	union			
NULL	type enumerated <identifier> { NULL }	MyType ::= NULL	type enumerated *MyType* { NULL }	9 *(21)*

ASN.1 TYPE	TTCN-3 TYPE		§
BMPString	universal charstring (char(0,0,0,0) .. char(0,0,255,255));		9 *(15)*
UTF8String	universal charstring		
NumericString	charstring	constrained to set of characters given in clause 41.2 of ITU-T X.680	
TeletexString	universal charstring	constrained to the set of characters given in clause 41.4 of ITU-T X.680	
T61String			
VideotexString			

ASN.1 TYPE	TTCN-3 TYPE	§
GraphicString	universal charstring	9 *(15)*
GeneralString		
OPEN TYPE	anytype	
VisibleString	charstring	8.1
IA5String		
UniversalString	universal charstring	

D.4 XML mapping

The following tables present introduction examples only (e.g. attributes are omitted); complete definitions are provided in ES 201873-9. The TTCN-3 module containing type definitions equivalent to XSD built-in types is given in Annex A.

USER TYPES	*XML EXAMPLE*	*TTCN-3 EQUIVALENT*	§
sequence elements	<sequence> <**element** name="*my1*" type="integer"/> <**element** name="*my2*" type="string"/> </sequence>	type record *MyType* {XSD.Integer *my1*, XSD.String *my2* };	7.3
global attribute	<attribute name="*myType*" **type**="*BaseType*"/>	type *BaseType MyType*;	7.4.1
list	<**list** itemType="float"/>	type record of XSD.Float *MyType;*	7.5.2
union (named)	<xsd:**union** memberTypes="xsd:string xsd:boolean"/>	type union *MyTypememberlist* { XSD.String *string*, XSD.Boolean *boolean_*};	7.5.3
(unnamed)	<**union**> <simpleType> <restriction base="xsd:string"/> </simpleType> <simpleType> <restriction base="xsd:float"/> </simpleType> </**union**>	type union *MyType* { XSD.String *alt_*, // predefined fieldnames XSD.Float *alt_1* };	

complex type	<**complexType** name="*baseType*"> <sequence>
	<element name="*my1*" type="string"/>
	<element name="*my2*" type="string"/>
	</sequence>
	<attribute name="*my3*" type="integer"/> </**complexType**>
	<**complexType** name="*myType*"> <**complexContent**>
	<extension base="ns:*baseType*">
	<sequence> <element name="*my4*" type="integer"/> </sequence>
	<attribute name="*my5*" type="string"/>
	</extension>
	</**complexContent**> </**complexType**>
		type record *MyType* {
		XSD.Integer *my3* **optional**,
		XSD.String *my5* **optional**,
		// elements of base type
		XSD.String *my1*,
		XSD.String *my2*,
		// extending element and group reference
		XSD.Integer *my4*,
		};
			7.6.2

all content	<complexType name="*myType*"> <**all**> <element name=" *my1*" type="integer"/> <element name=" *my2*" type="string"/> </**all**></complexType>	type record *MyType* { record of enumerated {*my1, my2*} *order*, //predef. XSD.Integer *my1*, XSD.String *my2* };	7.6.4
choice	<complexType name="*myType*"> <**choice**> <element name="*my1*" type="integer"/> <element name="*my2*" type="float"/> </**choice**></complexType>	type record *MyType* { union {XSD.Integer *my1*, XSD.Float *my2*} *choice* // predefined fieldname };	7.6.5
sequence	<complexType name="*myType*"> <**sequence**> <element name="*my1*" type="integer"/> <element name="*my2*" type="float"/> </**sequence**> </complexType>	type record *MyType* {XSD.Integer *my1*, XSD.Float *my2*};	7.6.6
any	<xs:complexType name="*myType*"> <xs:sequence> <xs:**any** namespace="##any"/> </xs:sequence> </xs:complexType>	type record *MyType* {XSD.String *elem*}; // predefined fieldname	7.7
group	<xs:**group** name="*myType*"> <xs:sequence> <xs:element name="*myName*" type="xs:string"/> </xs:sequence> </xs:**group**>	type record *MyType* {XSD.String *myName*};	7.9

XML FACETS	XML EXAMPLE	TTCN-3 EQUIVALENT	§
length restrictions	<**length** value="10"/>	**type** XSD.String *MyType* **length**(10);	6.1.1
	<**minLength** value="3"/>	**type** XSD.String *MyType* **length**(3 .. **infinity**);	6.1.2
	<**maxLength** value="5"/>	**type** XSD.String *MyType* **length**(0 .. 5);	6.1.3
pattern	<**pattern** value="abc??xyz*0 "/>	**type** XSD.String *MyType* (**pattern** "abc??xyz*0");	6.1.4
enumeration	<xsd:**enumeration** value="*yes*"/> <xsd:**enumeration** value="*no*"/>	**type enumerated** *MyEnum* {*yes*, *no*};	6.1.5
value restrictions	<**minInclusive** value="-5"/>	**type** XSD.Integer *MyType* (-5 .. **infinity**);	6.1.7/8
	<**maxExclusive** value="10"/>	**type** XSD.PositiveInteger *MyType* (1 .. !10);	6.1.9/10
list boundaries	<element name="*my1*" type="integer" **minOccurs**="0"/> <element name="*my2*" type="integer" **minOccurs**="5" **maxOccurs**="10"/>	XSD.Integer *my1* **optional** **record length**(5..10) **of** XSD.Integer *my2_list*;	7.1.4

D.5 Extensions

ADVANCED PARAMETERIZATION (ES 202 784)	EXAMPLES	DESCRIPTION	§
FormalTypeParList ::= "<" *FormalTypePar* {"," *FormalTypePar* } ">"	**type record** *MyData* <**in type** *p_PayloadType*> {*Header p_hdr*, *p_PayloadType p_payload*};	type definition with formal type parameter;	5.2 *(5.4.1.5)*
FormalTypePar ::= ["**in**"] [*Type* |"**type**"] *TypeParIdentifier* [":=" *Type*]	**var** *MyData* <**charstring**> *v_myMsg* :={*c_hdr*,"ab"};	instantiation second field;	5.5
	function *f_myfunction* <**in type** *p_MyType* > (**in** *MyList*<*p_MyType*> *p_list*, **in** *p_ MyType p_elem*) **return** *p_ MyType* {…**return** (*p_list*[0] + *p_elem*);}	function definition with formal type and two parameters (2nd parameter type not fixed); function body;	
can appear in definitions of type, template, and statement blocks	*f_myfunction* <**integer**> ({1,2,3,4}, 5)	apply function with concrete type and parameter values	

BEHAVIOUR TYPES (ES 202 785)	EXAMPLES	DESCRIPTION	§
type function *BehaviourTypeIdentifier* ["<" {*FormalTypePar* [","] } ">"] "(" [{ (*FormalValuePar* | *FormalTimerPar* | *FormalTemplatePar* | *FormalPortPar*) [","]}] ")" [**runs on** (*ComponentType* | **self**] [**return** [**template**] *Type*]	**type function** *MyFuncType* (**in integer** *p1*); **function** *f_myFunc1* (**in integer** *p1*) {…}; … **var** *MyFuncType v_func*; *v_func* := *f_myFunc1*; …	new type definition (w/o body); concrete behaviour; define formal function variable; assign concrete function;	5.2 *(6.2.13. 1)*
apply "(" *Value* "(" [{ (*TimerRef* | *TemplateInstance* | *Port* | *ComponentRef* | "-") [","]}] ")" ")"	**apply** (*v_func* (0)); *myComponent*.**start**(**apply**(*v_func* (1));	execute *f_myFunc1;* start PTC with *f_myFunc1*	5.8 5.11

CONFIGURATION AND DEPLOYMENT SUPPORT (ES 202 781)	EXAMPLES	DESCRIPTION	§
configuration *ConfigurationIdentifier* "(" [{(*FormalValuePar* | *FormalTemplatePar*) [","] }] ")" **runs on** *ComponentType* [**system** *ComponentType*] *StatementBlock*	**configuration** *f_StaticConfig* () **runs on** *MyMtcType* **system** *MySystemType* {… *myComponent* := *MyPTCType*.**create static**; **map** (*myComponent* :*PCO*, **system**:*PCO1*) **static**; …}	definition outside of testcases contains static configuration; creation of component; static mapping;	5.2

testcase *TestcaseIdentifier*
"(" [{(*FormalValuePar* |
FormalTemplatePar) [","]}] ")"
(**runs on** *ComponentType* [**system** *ComponentType*] |
execute on *ConfigurationType*)
StatementBlock
	testcase *TC_test1* () **execute on** *f_StaticConfig* {…}
	
	control {
	var configuration *myStaticConfig*;
	myStaticConfig := *f_StaticConfig*();
		testcase to be executed with static configuration;	5.7
			
			5.3

execute "(" *TestcaseRef* "(" [{*TemplateInstance* [","]}] ")" ["," *TimerValue*] ["," *ConfigurationRef*] ")"	**execute**(*TC_test1*, 2.0, *f_StaticConfig*); *myStaticConfig*.**kill**; …}	configuration setup; run test with static configuration; configuration down;	5.8
type port *PortTypeId* **message** [**map to** {*OuterPortType*[","]}+] [**connect to** …] "{" {(**in** {*InnerInType* [**from** {*OuterInType* **with** *InFunction*"()" [","]}+] [","]}+ |**out** {*InnerOutType* [**to** {*OuterOutType* **with** *OutFunction*"()" [","]}+] [","]}+ |**inout** … |**address** …|**map param** … |**unmap param** … |*VarInstance*) ";"}+ "}"	**type port** *DPort1* **message map to** *TPort2* {**in** *DType1* **from** *TType2* **with** *f_T2toD1*(); **out** *DPort1* **to** *TType2* **with** *f_D1toT2* (); … }	message port definition with translation functions NEW	5.10
function *FunctionIdentifier* "(" **in** *FormalValuePar* "," **out** *FormalValuePar* ")" [**port** *PortTypeId*] *StatementBlock*	**function** *f_T2toD1* (**in** *TType2 p_in*, **out** *DType1 p_out*) **port** *DPort1* {… **port**.**setstate**(TRANSLATED); …}	translation function (states: TRANSLATED, NOT_TRANSLATED, FRAGMENTED, PARTIALLY_TRANSLATED)	
port.setstate "(" *SingleExpression* { "," (*FreeText* | *TemplateInstance*) } ")"			

PERFORMANCE AND REAL -TIME TESTING (ES 202 782)	EXAMPLES	DESCRIPTION	§
type port *PortTypeIdentifier* **message** [**realtime**] "{" {(**in** | **out** | **inout**) {MessageType [","]}+ ";"} "}"	**module** *MyModule* {… **type port** *MyPort* **message realtime** {…}; **type component** *MyPTC* {**port** *MyPort myP*; …}; **var float** *v_specified_send_time*, *v_sendTimePoint*, *v_myTime*; …	port qualified for timestamp;	5.2
(*Port* | **any port**) "." **receive** ["(" *TemplateInstance* ")"] [**from** *AddressRef*] [-> [**value** *VariableRef*] [**sender** *VariableRef*] [**timestamp** *VariableRef*]]	*myP* .**receive**(*m_expect*) -> **timestamp** *v_myTime*; …	time of message receipt;	5.3
	wait (*v_specified_send_time*); *myP*.**send**(*m_out*); *v_sendTimePoint* := **now**; …	wait a specified time period (in sec.); get the actual time;	5.4 5.1.1
	} **with** {**stepsize** "0.001"};	timestamp precision of a msec.	5.1.2

INTERFACES WITH CONTINUOUS SIGNALS (ES 202 786)	*EXAMPLES*	DESCRIPTION	§
type port *PortTypeIdentifier* **stream** "**{**" (**in** | **out** | **inout**) *StreamValueType* "**}**" **port** *StreamPortTypeReference* {*StreamPortIdentifier* ["**:=**" *StreamDefaultValue*] ["**,**"]}+ ["**;**"]	**type port** *MyStream* **stream {in** *MyData* **};** **type record** *Sample* **{***MyData v*, **float** *d***};** **type record of** *Sample MySamples*; **type component** *MyPTC* **{port** *MyStream myIn := myDef*; **…};**	incoming data stream; a stream sample is a pair of a value and time (delta); *myDef* is default value of *myIn;*	5.2.2.1 5.2.2.2
(*StreamPortReference* | *StreamPortSampleReference*) "**.**" (**value** | **timestamp** | **delta**) *StreamPortReference* "**.**" **prev** ["**(**" *PrevIndex* "**)**"] *StreamPortReference* "**.**" **at** ["**(**" *Timepoint* "**)**"] *StreamPortReference* "**.**" (**history** | **values**) "**(**" *StartTime* "**,**" *EndTime* "**)**" *StreamPortReference* "**.**" **apply** "**(**" *Samples* "**)**" **assert** "**(**" *Predicate* {"**,**" *Predicate*} "**)**"	*myIn*.**value;** *myIn*.**timestamp;** *myIn*.**delta** := 0.001**;** *myIn*.**prev(***i***).value;** *myIn*.**at(***tim***).value;** **var** *MySamples myRec* **:=** *myIn*.**history**(0.0, **now**); **var record of** *MyData myValues* **:=** *myIn*.**values**(0.0, **now**); *myOut*.**apply**(*myRec*); **assert (***myIn*.**value** == 4.0, *myIn*.**timestamp** == 5.0**);**	current stream value; time info (float) actual sample; set stepsize (float) of a stream; previous (*i* steps before) value; value at specified time *tim*; stream subpart with time (delta) info; stream samples w/o time info; send out stream samples; setverdict(fail) if any condition is false;	5.2.3.1 5.2.3.2 5.2.3.3 5.2.4.1 5.2.4.2 5.2.5.1 5.2.5.2 5.2.5.3 5.3
mode *ModeName* ["**(**" { (*FormalValuePar* | *FormalTimerPar* | *FormalTemplatePar* | *FormalPortPar* | *FormalModePar*) ["**,**"] } "**)**"] [**runs on** *ComponentType*] (**cont** | **par** | **seq**) "**{**" {*Declaration*} [**onentry** *StatementBlock*] [**inv** "**{**" *Predicate* {"**,**" *Predicate*} "**}**"] *Body* [**onexit** *StatementBlock*] "**}**" [**until** "**{**" {"**[**" [*Guard*] "**]**" [*TriggerEvent*] [*StatementBlock*] [*GotoTarget*]} "**}**"]	**module** *MyModule* **{** **cont {…** **onentry {***var1***:= 1; …}** **inv {***a* > 1.0**}** **…** **onexit {***var1***:= 7; …}** **…}** **until** **{[***a* > *b***] {…; repeat}** **[] {…; continue}** **…}** **…** **wait**(1.0) **} with {stepsize** "0.001"**};**	declaration of mode instance; at activation of mode; condition during block execution; assignments or asserts (body); at termination of mode; end of mode; restart mode; next step of mode (par/seq/cont); suspend execution for 1 sec. timestamp precision of a msec.	5.4.1 5.4.1.3 5.4.1.2 5.4.1.4 5.4.1.1.3 5.4.1.1.4 5.5 5.1.2

Document overview:

ES 201 873-1 (2013-04) TTCN-3 part 1 (edition 4.5.1): *Core Language (CL)*
ES 201 873-2 (2007-02) TTCN-3 part 2 (edition 3.2.1): *Tabular Presentation format (TFT)* (historical - not maintained!)
ES 201 873-3 (2007-02) TTCN-3 part 3 (edition 3.2.1): *Graphical Presentation format (GFT)* (historical - not maintained!)
ES 201 873-4 (2012-04) TTCN-3 part 4 (edition 4.4.1): *Operational Semantics (OS)*
ES 201 873-5 (2013-04) TTCN-3 part 5 (edition 4.5.1): *TTCN-3 Runtime Interface (TRI)*
ES 201 873-6 (2013-04) TTCN-3 part 6 (edition 4.5.1): *TTCN-3 Control Interface (TCI)*
ES 201 873-7 (2013-04) TTCN-3 part 7 (edition 4.5.1): *Using ASN.1 with TTCN-3*
ES 201 873-8 (2013-04) TTCN-3 part 8 (edition 4.5.1): *The IDL to TTCN-3 Mapping*
ES 201 873-9 (2013-04) TTCN-3 part 9 (edition 4.5.1): *Using XML schema with TTCN-3*
ES 201 873-10 (2013-04) TTCN-3 part 10 (edition 4.5.1): *TTCN-3 Documentation Comment Specification*

ES 202 781 (2013-06) TTCN-3 Language Extensions (version 1.2.1): *Configuration and Deployment Support*
ES 202 782 (2010-07) TTCN-3 Language Extensions (version 1.1.1): *TTCN-3 Performance and Real Time Testing*
ES 202 784 (2013-04) TTCN-3 Language Extensions (version 1.3.1): *Advanced Parameterization*
ES 202 785 (2013-04) TTCN-3 Language Extensions (version 1.3.1): *Behaviour Types*
ES 202 786 (2012-04) TTCN-3 Language Extensions (version 1.1.1): *Support of interfaces with continuous signals*
ES 202 789 (2013-04) TTCN-3 Language Extensions (version 1.2.1): *Extended TRI*
TS 102 995 (2010-11) *Proforma for TTCN-3 reference test suite* (version 1.1.1)

Upcoming event