TTCN-3 Quick Reference Card

- PDF has links to TTCN-3 online Standards, browseable BNF and Quiz -

For TTCN-3 edition 4.9.1 (2017) and extensions (see also poster A0 format).
Designed and edited by Axel Rennoch, Claude Desroches, Theo Vassiliou and Ina Schieferdecker.

Contents

Document overview:

[1] ES 201 873-1 **(2017-05)** TTCN-3 part 1 (edition 4.9.1): ***Core Language (CL)*** [ITU-T Z.161]
[2] ES 201 873-2 **(2007-02)** TTCN-3 part 2 (edition 3.2.1): ***Tabular Presentation format (TFT)*** **(historical - not maintained!)** [ITU-T Z.162]
[3] ES 201 873-3 **(2007-02)** TTCN-3 part 3 (edition 3.2.1): ***Graphical Presentation format (GFT)*** [ITU-T Z.163]
[4] ES 201 873-4 **(2017-05)** TTCN-3 part 4 (edition 4.6.1): ***Operational Semantics (OS)*** [ITU-T Z.164]
[5] ES 201 873-5 **(2017-05)** TTCN-3 part 5 (edition 4.8.1): ***TTCN-3 Runtime Interface (TRI)*** [ITU-T Z.165]
[6] ES 201 873-6 **(2017-05)** TTCN-3 part 6 (edition 4.9.1): ***TTCN-3 Control Interface (TCI)*** [ITU-T Z.166]
[7] ES 201 873-7 **(2017-05)** TTCN-3 part 7 (edition 4.6.1): ***Using ASN.1 with TTCN-3*** [ITU-T Z.167]
[8] ES 201 873-8 **(2015-06)** TTCN-3 part 8 (edition 4.6.1): ***The IDL to TTCN-3 Mapping*** [ITU-T Z.168]
[9] ES 201 873-9 **(2017-05)** TTCN-3 part 9 (edition 4.8.1): ***Using XML schema with TTCN-3*** [ITU-T Z.169]
[10] ES 201 873-10 **(2013-04)** TTCN-3 part 10 (edition 4.5.1): ***TTCN-3 Documentation Comment Specification*** [ITU-T Z.170]
[11] ES 201 873-11 **(2017-06)** TTCN-3 part 11 (edition 4.7.1): ***Using JSON with TTCN-3*** NEW [ITU-T Z.171]

[12] ES 202 781 **(2017-05)** TTCN-3 Language Extensions (version 1.5.1): ***Configuration and Deployment Support*** [ITU-T Z.161.2]
[13] ES 202 782 **(2015-06)** TTCN-3 Language Extensions (version 1.3.1): ***TTCN-3 Performance and Real Time Testing*** [ITU-T Z.161.5]
[14] ES 202 784 **(2017-04)** TTCN-3 Language Extensions (version 1.6.1): ***Advanced Parameterization*** [ITU-T Z.161.3]
[15] ES 202 785 **(2017-08)** TTCN-3 Language Extensions (version 1.5.1): ***Behaviour Types*** [ITU-T Z.161.4]
[16] ES 202 786 **(2017-05)** TTCN-3 Language Extensions (version 1.4.1): ***Support of interfaces with continuous signals*** [ITU-T Z.161.1]
[17] ES 202 789 **(2015-06)** TTCN-3 Language Extensions (version 1.4.1): ***Extended TRI*** [ITU-T Z.165.1]
[18] ES 203 022 **(2017-07)** TTCN-3 Language Extensions (version 1.1.1): ***Advanced Matching*** NEW [ITU-T Z.161.6]
[19] TS 102 995 **(2010-11)** ***Proforma for TTCN-3 reference test suite*** (version 1.1.1)

A.1 MODULE STRUCTURES

MODULE, IMPORT, GROUP	*EXAMPLES*	DESCRIPTION	§ [1]
module *ModuleIdentifier* [**language** *FreeText* {"," *FreeText*}] "**{**"[*ModuleDefinitionsPart*] [*ModuleControlPart*] "**}**"	**module** *MyTypes* **language** "TTCN-3:2016" **{…}** **module** *MyConfig* **language** "TTCN-3:2015" **{…}**	version 4.8.1; version 4.7.1;	8.1
[*Visibility*] **import from** *ModuleId* ((**all** [**except** "**{**" *ExceptSpec* "**}**"]) |("**{**" *ImportSpec* "**}**")) [";"]	**public import from** *MyModule* **{type** *MyType*, **template all};**	definitions visible in defining and other (importing) module;	8.2.3 8.2.5
	friend import from *urn_3gpp_ns_cw_1_0* **language** "XSD" **all;**	definitions visible in defining and friend modules (namespace="urn:3gpp:ns:cw:1.0");	
	private import from *MyIPs* **all except {group** *myGroup***};**	definitions in *MyIPs* cannot be imported by other modules ("private" is default for "import");	
[**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	§ [1]
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	§ [1]
type component *ComponentTypeIdentifier* [**extends** *ComponentTypeIdentifier* {"," *ComponentTypeIdentifier*}] "**{**" { (*PortInstance* | *VarInstance* | *TimerInstance* | *ConstDef* | *TemplateDef* } "**}**"	**type component** *MyPtcA* **{port** *MyPortTypeA myPort*; **port** *MyPortTypeA myPorts*[3]; **var** *MyVarTypeA vc_var1* **};** **type component** *MyPtcB* **extends** *MyPtcA*, *MyPtcA2* **{timer** *tc_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 and MyPtcA2;* *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	§ [1]
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*;	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*; **map param (in integer** *p_p1*, **out** *MyType p_p2***)** **};**	synchronous communication; to call remote operation (get replies/exceptions); to get calls from other components (and sent replies/ exceptions);	

PROCEDURE SIGNATURES	*EXAMPLES*	DESCRIPTION	§ [1]
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	§ [1]
boolean	**true, false**		**type boolean** *MyBoolean* **(true);**	list	6.1.0, 6.1.2
integer	(**-infinity..**-1), 0, 1, (2 **.. infinity**), (!-1 .. 30)	-1 excluded	**type integer** *MyInteger* (-2, 0, 1..3**);**	list, range	
float	(**-infinity..** -2.783), 0.0, 2.3E-4, (1.0..3.0, **not_a_number**)	"invalid" value NaN (IEEE 754)	**type float** *MyFloat* (1.1 .. **infinity**);		
charstring	"<empty>" , "any", """" *v_ myCharstring*[0]; **lengthof (***v_ myCharstring***);**	7-bit coded chars (ITU T.50); single quote-symbol: ''; first character of string; length of string	**type charstring** *MyISO646* **length**(1); **type charstring** *MyChars* (**pattern** "A?"); **type charstring** *MyShortCharStrings* **length** (2..9);	list, range, length, pattern	6.1.1, 6.1.2, E.2
universal charstring	**char**(0,0,3,179) & "more"	gamma (γ) in ISO/IEC 10646 (default UTF32: 4 bytes)	**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	§ [1]
default	**var default** *v_myAltstep* **:= null;**	manage use of altstep (activate/deactivate); **null** is concrete **default** value;	6.2.8
address	**var address** *v_myPointer* **:= null;**	reference of component or SUT interface (global scope); **null** is concrete **address** value;	6.2.12
verdicttype	**var verdicttype** *v_myVerdict***;**	fixed values: none (default), pass, inconc, fail, error;	6.1.0
objid	**objid** { 0 4 0 }	values are the set of all syntactically correct object identifier values (use with ASN.1 only)	7.2 [7]

STRUCTURED TYPES, ANYTYPE	*SAMPLE VALUES*	*SAMPLE USAGE*		SUBTYPES	§ [1]
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*[-] *v_myField* := 1; //inner type	**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 *MyKeys* {*e_key1*, e_*key2*, e_ *key3*}; **type enumerated** *MyTags* {*e_keyA* (2), e_*keyB* (1)};	**var** *MyKeys v_enum* := e_*key1*; **var** *MyTags v_enum2* := e_*keyA*;	**type** *MyKeys MyShortList* {*e_key1*, e_*key3*};	e_*key1* < e_*key2* < e_*key3* e_ *keyA* > e_*keyB*		6.2.4
type union *MyUnionType* {**@default integer** *alt1*, **float** *alt2* }	**var** *MyUnionType v_myDefault* := 1; **var** *MyUnionType v_myUnion* := {*alt2* := 1.0};	 *v_myUnion.alt1*; **ischosen** (*v_myUnion.alt2*)	**NEW** opt. default *n/a (error)* **true**	list	6.2.5 C.3.2
anytype /* union of all data 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

DECLARATIONS	*EXAMPLES*	DESCRIPTION	§ [1]
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 [**@lazy** | **@fuzzy**] *Type VarIdentifier* [*ArrayDef* [":=" *Expression*] { [","]*VarIdentifier* [*ArrayDef*] [":=" *Expression*] } [";"]	**var boolean** *v_myVar2* := **true**, *v_myVar3* := **false**; **var @lazy integer** *v_myVar4*;	passed to both value and template-type formal parameters	11.1
var template [**@lazy**| **@fuzzy**] [*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 19.1
[*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

TEMPLATE	*EXAMPLES*	DESCRIPTION	§ [1]
template [*TemplateRestriction*] [**@fuzzy**] *Type TemplateIdentifier* ["(" *TemplateOrValueFormalParList*")"] [**modifies** *TemplateRef*] ":=" *TemplateBody*	**template** *MyTwoInteger mw_subtemplate* (**template integer** *p_1*:=4) := {1, *p_1*}; **var template** *MyRecord* *v_template* := {**omit**, *mw_subtemplate*(1)}; **var** *MyRecord v_value* := **valueof** (*v_template*); **isvalue**(*v_template*); **template bitstring** *m_bits* := '010'B **& ? length**(1);	template with parameter (default value 4, if missing); parameter allows template expressions (e.g. wildcards); template variable; in-parameters may be @lazy/@fuzzy **valueof**-operation: error in case of unspecific content (e.g. wildcards); returns true, if *v_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 (**value**) *MyType v_t2;* // for sending
	var template (**omit**) *MyType v_t1*; // for sending
	var template (**present**) *MyType v_t3*;
	// do not use for sending
		resolve to specific value (fields resolve to specific value or **omit**);
		template/fields resolve to specific value or **omit**;
		cannot resolve to **omit**, *, **ifpresent**, **complement** (fields contain any expectations, except **complement**);
			15.8

TYPE	send/call/reply/raise *TEMPLATES* *(concrete values only)*	receive/getcall/getreply/getraise *TEMPLATES* *(can contain wildcards)*	§ [1]
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** }; **template** *MyRecord mw_record3* := { 1.0, *field2*:= **decmatch** *c_field2*};	15.1 15.5 15.7 27.7 15.7.2 B.1.2.9
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*); **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	§ [1]
template charstring *mw_template*:= **pattern** "ab??xyz*0" ; // use pattern from ch. C.3	string 'ab' followed by any two characters, 'xyz', none or any number of characters, followed by '0';	B.1.5
template universal charstring *mw_templ* := **pattern** **@nocase** "a*z" **length**(2..10);	up to eight characters between first and last element; @nocase modifier indicate case-insensitive way	B.1.5.6

B.1 BEHAVIOUR BLOCKS

TESTCASE	EXAMPLES	DESCRIPTION	§ [1]
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	§ [1]
function [**@deterministic**] *FunctionIdentifier* "(" [{ (*FormalValuePar* | *FormalTimerPar* | *FormalTemplatePar* | *FormalPortPar*) [","]}] ")" [**runs on** *ComponentType*] [**mtc** *ComponentType*] [**system** *ComponentType*] [**return** [**template**] *Type*] *StatementBlock*	**function** *f_myFunctionPtcA* **(in template** *MyTempType p_t1***) runs on** *MyPtcA* **mtc** *MyMtcType* **return template** *MyTempType* **{timer** *t_local* := 1.0; **var** *MtcType v_mtc* := *mtc;...***};** **function** *f_myFctNoRunsOn* **(in @lazy integer** *p_myp*:=1**)** **return template** *MyTempType* **{...};** **var template** *MyTempType* *v_genericTemplate* := *f_myFctNoRunsOn***(2);** *v_genericTemplate* := *f_myFctNoRunsOn***();**	invoke from components compatible to *MyPtcA*, parameter allows wildcards; timer for local use only; can be called from any place (no "runs on"); in-parameters may be @lazy/@fuzzy invoke *f_myFctNoRunsOn*; invoke with default of *p_myp*;	16.1 5.4.2 5.4.1.1

ALTSTEP	EXAMPLES	DESCRIPTION	§ [1]
altstep *AltstepIdentifier* "(" [{(*FormalValuePar* | *FormalTimerPar* | *FormalTemplatePar*| *FormalPortPar*) [","]}] ")" [**runs on** *ComponentType*] [**mtc** *ComponentType*] [**system** *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; re-evaluation of alt-statement; exit from altstep; in-parameters may be @lazy/@fuzzy 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

ASSIGNMENTS, BRANCHES, LOOPS	EXAMPLES	DESCRIPTION	§ [1]
VariableRef **":="** (*Expression* | *TemplateBody*)	*v_myValue* := *v_myValue2;* **var** *PTC1 v_ptc* := *PTC2*.**create**;	basic assignment; PTC2 compatible to PTC1 (*v_ptc* may have invisible resources)	19.1 6.3.3
if "(" *BooleanExpression* ")" *StatementBlock* {**else if** "(" *BooleanExpression* ")" *StatementBlock*} [**else** *StatementBlock*]	**if** (*v_myBoolean*) **{...} else {...};**		19.2
select "(" *SingleExpression* ")" "{" {**case** "(" {*TemplateInstance* [","]} ")" *StatementBlock*}+ [**case else** *StatementBlock*] "}"	**select** (*v_myString*) { **case** ("blue") **{...}** **case** ("red", "black"){...} **case else {...}};**	selector can be of any type (e.g. integer, enumerated) compatible to case expressions;	19.3.1
select union "(" *SingleExpression* ")" "{" {**case** "(" ({ *Identifier* [","]} | { *TemplateInstance* [","]}) ")" *StatementBlock* }+ [**case else** *StatementBlock*] "}"	**select union** (*mw_myTemplate*) **{...};**		19.3.2
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
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

VERDICT HANDLING	EXAMPLES	DESCRIPTION	§ [1]
verdicttype	**var verdicttype** *v_verdict*;	(none, inconc, pass, fail, error)	24
setverdict "(" *SingleExpression* { "," (*FreeText* | *TemplateInstance*)} ")"	**setverdict(pass, "my scenario was successful");**	initial value is **none;** change current verdict (could not be improved);	24.2
getverdict;	*v_verdict* := **getverdict;**	retrieve actual component verdict;	24.3

AUXILLARY CONSTRUCTS	EXAMPLES	DESCRIPTION	§ [1]
match "(" *Expression* "," *TemplateInstance* ")"	**if (match ({0,(2..4)},** *mw_rec***)) {...};**	evaluates expression against template;	15.9
log "(" { (*FreeText*| *TemplateInstance*) [","] } ")"	**log ("expectation:",** *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
ReferencedValue "=>" (*PredefinedType* | *TypeIdentifier* | ("(" *Type* ["," *Expression*] ")"))	**type record** *PDU* **{..., bitstring** *oPay*, **...};** **type record** *OPay* **{universal charstring** *iPay***};** *v_in* := *v_pdu*.*oPay* => *OPay*.*iPay*	NEW decoded field reference	7.3

OPERATORS ***(precedence decreasing from left to right)*** — § [1]

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	§ [1]
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** | **any from** *PortArrayRef*) "." **receive** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" [**@decoded** ["(" *Expression* ")"]] *FieldOrTypeReference*]["."]} ")"))] [**sender** *VariableRef*] [**@index value** *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; @decoded value assignment @index refer port array;	22.2.2

QUEUE INSPECTION	*EXAMPLES*	DESCRIPTION	§ [1]
(*Port* | **any port** | **any from** *PortArrayRef*) "." **trigger** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" [**@decoded** ["(" *Expression* ")"]] *FieldOrTypeReference*]} ")"))] [**sender** *VariableRef*] [**@index value** *VariableRef*]]	*myPort*.**trigger** (*MyType*:?) -> **value** *v_income*;	removes all messages from queue (including the specified message with type *MyType*); @decoded value assignment @index refer port array;	22.2.3
(*Port* | **any port** | **any from** *PortArrayRef*) "." **check** ["(" (*PortReceiveOp* | *PortGetCallOp* |*PortGetReplyOp* | *PortCatchOp*) | ([**from** *Address*] ["->"[**sender** *VariableRef*] [**@index value** *VariableRef*]]) ")"]	*myPort*.**check** (**receive** (*m_template*) **from** *v_myPtc*);	evaluates top element against expectation; no change of queue status; @index refer port array	22.4

SYNCHRONOUS COMMUNICATION (call, reply), EXCEPTIONS (raise, catch)	*EXAMPLES*	DESCRIPTION	§ [1]
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**) {...} };		22.3.1
(*Port* | **any port** | **any from** *PortArrayRef*) "." **getcall** ["(" *TemplateInstance* ")"] [**from** *Address*] ["->" [**param** "(" {(*VariableRef* ":=" [**@decoded** ["(" *Expression* ")"]] *ParameterIdentifier*) "," } |{(*VariableRef* | "-") ","} ")"] [**sender** *VariableRef*] [**@index value** *VariableRef*]]		calling component: implicit timeout of 5 sec.; return value must be 1..9; *v_ret* gets return value; *v_myVar* gets "out"-value of parameter *p_myP2*;	22.3.2
(*Port* | **any port** | **any from** *PortArrayRef*) "." **getreply** ["(" *TemplateInstance* [**value** *TemplateInstance*] ")"] [**from** *Address*] ["->" [**value** *VariableRef*] [**param** "(" {(*VariableRef* ":=" [**@decoded** ["(" *Expression* ")"]] *ParameterIdentifier*) "," } | {(*VariableRef* | "-") ","} ")"] [**sender** *VariableRef*] [**@index value** *VariableRef*]]		remote exception raised; local timeout of implicit timer;	22.3.4
Port "." **reply** "(" *TemplateInstance* [**value** *Expression*] ")" [**to** *Address*]	*myPort*.**getcall** (*s_myExpectation*); ... **if** (*v_failure*) { *myPort*.**raise** (*s_myError*); ...}; ... *myPort*.**reply** (*s_myAnswer*)	called component: raise exception; regular reply; @index refer port array; @decoded value assignment	22.3.3 22.3.5 22.3.6
Port "." **raise** "(" *Signature* "," *TemplateInstance* ")" [**to** *Address*]			
(*Port* | **any port** | **any from** *PortArrayRef*) "." **catch** ["(" (*Signature* "," *TemplateInstance*) | "**timeout**"")"] [**from** *Address*] ["->" [**value** (*VariableRef* | ("(" {*VariableRef* [":=" [**@decoded** ["(" *Expression* ")"]] *FieldOrTypeReference*] [","]} ")"))] [**sender** *VariableRef*] [**@index value** *VariableRef*]]			

PORT OPERATIONS	*EXAMPLES*	DESCRIPTIONS	§ [1]
(*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"	22.5.5 E.2.2.4

EXTERNAL CALCULATION	*EXAMPLES*	DESCRIPTION	§ [1]
external function [**@deterministic**] *ExtFunctionIdentifier* "(" [{(*FormalValuePar* | *FormalTimerPar* |*FormalTemplatePar*| *FormalPortPar*) [","]}] ")" [**return** [**template** [*Restriction*]] *Type*]	**external function** *fx_myCryptoalgo* (**integer** *p_name*, ...) **return charstring**;	need implementation in platform adapter; in-parameters may @lazy/@fuzzy; NEW return of templates	16.1.3

B.4 TIMER and ALTERNATIVES

TIMER DEFINITIONS AND OPERATIONS	EXAMPLES	DESCRIPTION	§ [1]
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
((*TimerIdentifier* | *TimerParIdentifier*) {**"["** *SingleExpression* **"]"**}) **"."** **start** [**"("** *TimerValue* **")"**]	*t_ myTimer*.**start** (5.0); *t_ myTimer*.**start**;	timer started for 5 seconds; restart for 4 sec. (default);	23.2 23.4
((*TimerIdentifier* | *TimerParIdentifier*) {**"["** *SingleExpression* **"]"**}) **"."** **read**	**var float** *v_current* := *t_ myTimer*.**read**;	get actual timer value;	23.3
(((*TimerIdentifier* | *TimerParIdentifier*) {**"["** *SingleExpression* **"]"**}) | **all timer**) **"."** **stop**	*t_ myTimerArray*[1].**stop**;	stop 2nd timer from array;	
(((*TimerIdentifier* | *TimerParIdentifier*) {**"["** *SingleExpression* **"]"**}) | **any timer** | **any from** *TimerArrayRef*) **"."** **running** [**@index value** *VariableRef*]	**if** (**any timer**.**running**) {…}	any timer previously started; @index refer timer array	23.5
(((*TimerIdentifier* | *TimerParIdentifier*) {**"["** *SingleExpression* **"]"**}) | **any timer** | **any from** *TimerArrayRef*) **"."** **timeout** [**@index value** *VariableRef*]	*t_ myTimer*.**timeout**;	awaits timeout of *t_ myTimer;* @index refer timer array	23.6

ALTERNATIVES	EXAMPLES	DESCRIPTION	§ [1]

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 none 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	§ [1]
ComponentType **"."** **create** [**"("***Expression* [**","***Expression*] **")"**] [**alive**]	**var** *MyPtc myInstance*, *myInstance2*; **var** *MyPtc myPtcs*[3]; *myInstance* := *MyPtc*.**create alive**; *myInstance2* := *MyPtc*.**create** (**"ID"**);	initialize variable identifiers; allocate memory, "ID" for logging only;	21.3.1
(*VariableRef* | *FunctionInstance*) **"."** **start** **"("** *FunctionInstance***")"**	*myInstance*.**start** (*f_myFunction* (*v_param1,c_param2*));	start with *f_myFunction* behaviour;	21.3.2
stop | ((*VariableRef* | *FunctionInstance*| **mtc** | **self**) **"."** **stop**) | (**all component** **"."** **stop**)	*myInstance*.**stop**; *myInstance*.**start** (…); *myInstance*.**kill**; **all component**.**kill**;	stops (**alive** keeps resources); restart with new function; stop and release resources; kills PTCs (remove resources);	21.3.3
kill | ((*VariableRef* | *FunctionInstance*| **mtc** | **self**) **"."** **kill**) | (**all component** **"."** **kill**)	**stop**; **self**.**stop**; **mtc**.**stop**;	stops own component; stops testcase execution;	21.3.4
(*VariableRef* | *FunctionInstance*| **any component** | **all component** | **any from** *ComponentArrayRef*) **"."** (**alive** | **running**) [**"->"** **@index value** *VariableRef*]	*myInstance*.**alive**; *myInstance*.**running**;	checks status of specific, **any** or **all** components;	21.3.5 21.3.6
(*VariableRef* | *FunctionInstance* | **any component** | **all component** | **any from** *ComponentArrayRef*) **"."** (**done** | **killed**) [**"->"** [**value** *VariableRef*] **@index value** *VariableRef*]	**all component**.**done**; **any component**.**killed**; *myInstance*.**done** -> **value** *v_verdict*;	@index refer component array; verdict value assignment	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	§ [1]
map **"("** *ComponentRef* **":"** *Port* **","** *ComponentRef* **":"** *Port* **")"** [**param** **"("** [{*ActualPar* [**","**] }+] **")"**] **unmap** [(**"("***ComponentRef* **":"** *Port* **","** *ComponentRef* **":"** *Port* **")"**) | [**param** **"("** [{*ActualPar* [**","**] }+] **")"**] | (**"("** *PortRef* **")"**) [**param** **"("** [{ActualPar [**","**] }+] **")"**] | (**"("***ComponentRef* **":"** **all port** **")"**) | (**"("** **all component** **":"** **all port** **")"**)]	**map** (*myPtc*:*portA*, **system**:*portX*); **map** (**mtc**:*portA*, **system**:*portB*) **param** (*v_myConfig*); **unmap**; **unmap** (*myPtc*:*portA*); **unmap** (**mtc**:*portA*, **system**:*portB*);	assign to SUT via adapter provide values to adapter; unmaps all own port; unmaps *myPtc portA;* unmaps **mtc** *portA;*	21.1.1 21.1.2
connect **"("** *ComponentRef* **":"** *Port* **","** *ComponentRef* **":"** *Port* **")"** **disconnect** [(**"("***ComponentRef* **":"** *Port* **","** *ComponentRef* **":"** *Port* **")"**) | (**"("***PortRef* **")"**) | (**"("***ComponentRef* **":"** **all port** **")"**) | (**"("** **all component** **":"** **all port** **")"**)]	**connect** (**self**:*portA*, **mtc**:*portC*) **disconnect** (**mtc**:*portA*, *myPtc*:*portB*); **disconnect** (**mtc**:**all port**); **disconnect**;	between components w/o adapter; disconnect mtc portA; all connections of mtc; all connections of actual component;	21.1.1 21.1.2

PORT CONFIGURATION	EXAMPLES	DESCRIPTION	§ [1]
(*Port* | (**all port**) | **self**) **"."** **setencode** **"("** *Type* **","** *SingleExpression* **")"** **optional**	**type record** *MyPDU* {…, *MyField field*, …} **with** {**encode** "CD1"; **encode** "CD2"}; *portA*.**setencode** (*MyPDU.field*, "CD2");	NEW dynamic selection of encode attributes;	27.9

C.1 PREDEFINED FUNCTIONS and USEFUL TYPES

TYPE CONVERSION FUNCTIONS	*EXAMPLES*		§ [1]
int2char | **int2unichar** | **int2str**| **int2float** **(in integer** *invalue*) **return (**charstring | universal charstring | charstring | float)	**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
int2bit | **int2hex**| **int2oct** **(in integer** *invalue*, **in integer** *length*) **return (**bitstring | hexstring | octetstring)			C.1.1- C.1.8
int2enum (in integer inpar, **out** Enumerated_type outpar)			
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 | octetstring)	**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)	**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
oct2unichar (in octetstring *invalue,* **in charstring** *string_encoding* := "UTF-8") **return (**universal charstring)	**oct2unichar**('C384'O, "UTF-8")	"Ä"	C.1.31
unichar2oct (in universal charstring *invalue,* **in charstring** *string_encoding* := "UTF-8")) **return** (octetstring)	**unichar2oct**("Ä", "UTF-8")	'C384'O	C.1.32
any2unistr (in template any_type *invalue*) **return (**universal charstring)	**var integer** *v_int* := 1; **any2unistr**(*v_int*) "1"	conversion of value or template	C.1.33

STRING MANIPULATION	*EXAMPLES*	DESCRIPTION	§ [1]
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" type of *inpar*;	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" type of *inpar*;	C.4.3
regexp [**@nocase**] **(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 delt a"; *mw_pattern* := "(?+)(gam)(?+a)"; **regexp** (*v_string, mw_pattern,* 2);	charstring or universal charsting; three groups identified by "(" and ")"; group #2 ["(?+a)"] resolves to " delt a", return "" if mismatch (type of *inpar*); @nocase modifier indicates case-insensitive	C.4.1
encvalue (in template (value) any_type *inpar*, **in universal charstring** *encoding_info* := "", **in universal charstring** *dynamic_encoding* := "") **return bitstring**	*p_messagelen* := **lengthof (encvalue(***m_payload***));**	functions require codec implementation; **NEW** dynamic selection of encode attribute;	C.5.1
decvalue (inout bitstring *encoded_value*, **out** any_type *decoded_value*, **in universal charstring** *encoding_info* := "", **in universal charstring** *dynamic_encoding* := "") **return integer**	**decvalue**(*v_in, v_out*)	**decvalue** return indicates success (0), failure (1), incompletion (2); **NEW** dynamic selection of encode attribute	C.5.2
encvalue_unichar (in template (value) any_type *inpar*, **in charstring** *string_serialization* :="UTF-8", **in universal charstring** *encoding_info* := "", **in universal charstring** *dynamic_encoding* := "") **return universal charstring**	**encvalue_unichar** ('C3'O,"UTF-8")	encodes to universal charstring; **NEW** dynamic selection of encode attribute	C.5.3
decvalue_unichar (inout universal charstring *encoded_value*, **out** any_type *decoded_value*, **in charstring** *string_serialization* := "UTF-8", **in universal charstring** *encoding_info* := "", **in universal charstring** *dynamic_encoding* := "") **return integer**	**decvalue_unichar** (*v_uchar, v_out*,"UTF-8")	decodes universal charstring, return value 0 indicates success; **NEW** dynamic selection of encode attribute	C.5.4
encvalue_o (in template (value) any_type *inpar*, **in universal charstring** *encoding_info* := "") **return octetstring**	**encvalue_unichar** ('1100'B) *// returns 'C'O*	**NEW** encoding to octetstring	C.5.5
decvalue_o (inout octetstring *encoded_value*, **out** any_type *decoded_value*, **in universal charstring** *encoding_info* := "") **return integer**	**decvalue_o**(*v_in, v_out*)	**NEW** decoding of octetstring, return value indicates success (0), failure (1), incompletion (2);	C.5.6
get_stringencoding (in octetstring *encoded_value*) **return charstring**	**get_stringencoding ('**6869C3**'O)** *// returns "UTF-8"*	retrieve type of string encoding	C.5.7
remove_bom (in octetstring *encoded_value*) **return octetstring**	**remove_bom ('**FEFF0068006900**'O)** *// returns '0068006900'O*	remove 'byte order mark' of 'Universal Coded Character Set' encoding schemes	C.5.8

OTHER PREDEFINED FUNCTIONS	*EXAMPLES*		DESCRIPTION	§ [1]
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	C.3.4
istemplatekind (in template any_type *inpar*, **in charstring** *kind*) **return** boolean;	**istemplatekind** (*mw_t*, "*"); **istemplatekind** (*mw_t*, "list");		true if *inpar* contains specified matching mechanism	C.3.5 E.2.2.5
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
hostId (in charstring *idkind*:="Ipv4orIPv6") **return** charstring;	**hostId** ("Ipv4"); *// "127.0.0.1"*		function returns host ID	C.6.3

TYPE DEFINITIONS FOR SPECIAL CODING (USE WITH OTHER NOTATIONS: ASN.1, IDL, XSD)	DESCRIPTION	§ [1]
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** "UTF-16"**};**	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	§ [1]
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	§ [1]
with "{" { (**encode** | **variant** | **display** | **extension** | **optional**) [**override** | **@local**] ["(" *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
(*Type* | *TemplateInstance*) "." (**display** | **encode** | **variant** | **extension** | **optional**) ["(" *Expression* ")"]	**type record** *MyPDU* **{ ... } with {encode** "CD1"**};** **type record of universal charstring** *MyRec*; **var** *MyRec v_enc* **:=** *v_pdu*.**encode;**	NEW retrieving of attribute values; if *v_pdu* is of type *MyPDU*, then *v_enc* will contain {"CD1"};	27.8

C.3 CHARACTER PATTERN

META-CHARACTER	DESCRIPTION		*EXAMPLES*	§ [1]
?	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***}** **\q{U***xxxx***,U***xxxx* ...**}**	any universal character; any universal character ("UCS sequence identifier"-like syntaxes)		**char**(0,0,3,179): "γ" (gamma) **char**(**U**4E2D, **U**56FD): "中国" ISO/IEC 10646	
\t	control character HT(9): horizontal tab	characters according to ITU-T Recommendation T.50	HT(9)	A.1.5.1
\n	newline character		LF(10), VT(11), FF(12), CR(13)	
\r	control character CR: carriage return		CR(13)	
\s	whitespace character		HT(9), LF(10), VT(11), FF(12), CR(13), SP(32)	
\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 two characters C\a" only **\[** refers to the single character "[" only	
{*reference***}** **{***reference***}** **\N{***reference***}**	reference to existing definitions, e.g. **const charstring** *c_mychar* := "ac?"; reference to existing character set definitions e.g. **type charstring** *c_myset* ("a"..."c");		"{\c_mychar}" refers to string "ac?"; "{c_mychar}" refers to "ac" followed by any character; "\N{c_myset}" refers to "a", "b" or "c" only	
[ ] **-** **^**	any single character of the specified set range within a specified set exclude ranges within a specified set		[1s3] allows 1, s, 3 [a-d] allows a,b,c, d [^a-d] allows any character except a,b,c,d	
| **( )**	Used to denote two alternative expressions Used to group an expression		(a|b) indicates "a" or "b"	
#(*n***,** *m***)** **#***n* **+**	repetition of previous expression	min. n-times, max. m-times n-times repetition; shorthands: #(,) #(0,) optional	*d***#(***2,4***)** indicates "dd", "ddd" or "dddd" *d***#***3* indicates "ddd" *d***+** indicates "d", "dd", "ddd", ...	

C.4 PREPROCESSING MACROS

MACRO NAME	DESCRIPTION	EXAMPLES	§ [1]

__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 [19] (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 [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	§ [10]
@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	§ [10]
@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			§ [10]
@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*; ...}	structured 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*	§ [7]

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		§ [7]
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	§ [7]
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 [9].

USER TYPES	*XML EXAMPLE*	*TTCN-3 EQUIVALENT*	§ [9]
sequence elements	<complexType name="*myType*"><**sequence**> <**element** name="*my1*" type="integer"/> <**element** name="*my2*" type="string"/> </**sequence**></complexType>	type record *MyType* {XSD.Integer *my1*, XSD.String *my2*};	7.3 7.6.6
global attribute	<attribute name="*myType*" **type**="*BaseType*"/>	type *BaseType MyType*;	7.4.1
list	<element name="*myType*"><simpleType> <**list** itemType="float"/> </element></simpleType>	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)	<element name="*myType*"><simpleType><**union**> <simpleType> <restriction base="xsd:string"/> </simpleType> <simpleType> <restriction base="xsd:float"/> </simpleType> </**union**></element></simpleType>	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*, //predefined name 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
any	<element name="*myType*"><complexType> <sequence> <**any** namespace="##any"/> <sequence> <complexType></element>	type record *MyType* {XSD.String *elem*}; // predefined fieldname	7.7.1
group	<xsd:**group** name="*myType*"> <xsd:sequence> <xsd:element name="*myName*" type="xsd:string"/> </xsd:sequence> </xsd:**group**>	type record *MyType* {XSD.String *myName*};	7.9

XML FACETS	XML EXAMPLE	TTCN-3 EQUIVALENT	§ [9]
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
total digits	<restriction base="decimal">	**type** XSD.Decimal *MyType* (-9999.0 .. 9999.0)	6.1.11
fraction digits	<**totalDigits** value="4"/><**fractionDigits** value="1"/></restriction>		6.1.12
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

CONFIGURATION AND DEPLOYMENT SUPPORT (ES 202 781)	EXAMPLES	DESCRIPTION	§ [12]
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;	5.1.2
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*(); **execute**(*TC_test1*, 2.0, *f_StaticConfig*); *myStaticConfig*.**kill**; **...}**	testcase to be executed with static configuration; configuration setup; run test with static configuration; configuration down;	5.1.7 5.1.3 5.1.8
execute "**(**" *TestcaseRef* "**(**" [{*TemplateInstance* [","]}] "**)**" ["," *TimerValue*] ["," *ConfigurationRef*] "**)**"			
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 5.1.10	
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	§ [13]
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

ADVANCED PARAMETERIZATION (ES 202 784)	EXAMPLES	DESCRIPTION	§ [14]
FormalTypeParList ::= "<" *FormalTypePar* {"," *FormalTypePar* } ">"	**type record** *MyData* <**in type** *p_PayloadType*> {*Header p_hdr*, *p_PayloadType p_payload*}; **var** *MyData* <**charstring**> *v_myMsg* **:=**{*c_hdr*,"ab"};	type definition with formal type parameter; instantiation second field;	5.2 *(5.4.1.5)*
FormalTypePar ::= ["**in**"] [*Type* |"**type**"] *TypeParIdentifier* [":=" *Type*] can appear in definitions of type, template, and statement blocks	**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*);} *f_myfunction* <**integer**> ({1,2,3,4}, 5)	function definition with formal type and two parameters (2nd parameter type not fixed); function body; apply function with concrete type and parameter values	5.5

BEHAVIOUR TYPES (ES 202 785)	EXAMPLES	DESCRIPTION	§ [15]
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

INTERFACES WITH CONTINUOUS SIGNALS (ES 202 786)	*EXAMPLES*	DESCRIPTION	§ [16]
type port *PortTypeIdentifier* **stream** "{" (**in** | **out** | **inout**) *StreamValueType* [";"] (**map param** "(" {*FormalValuePar* [","]}+ ")" [";"]) | (**unmap param** "(" {*FormalValuePar* [","]}+ ")" [";"]) "}"	**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 v_myRec* := *myIn*.**history**(0.0, **now**); **var record of** *MyData v_myValues* := *myIn*.**values**(0.0, **now**); *myOut*.**apply**(*v_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** {*v_var1*:= 1; ...} **inv** {*a* > 1.0} ... **onexit** {*v_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

ADVANCED MATCHING (ES 203 022) NEW	*EXAMPLES*	DESCRIPTION	§ [18]
@dynamic (*StatementBlock* | *FunctionRef*)	**template** *MyType mw_t* := **@dynamic** {... **if** (*my_f*(**value**)) {**return false**};... **return true**;}	**value** represents *mw_t*(*v_arg*) argument; **false** implies: template match fails; **true** implies: template matches	5.1
TemplateInstance "->" *VariableRef*	**template integer** *mw_t* := ((1..2)->*v_var1*, (1..5)->*v_var2*))	*v_var1* retrieves value if 1 or 2, *v_var2* retrieves value if 3, 4 or 5	5.2.1
template [*restriction*] [**@fuzzy**] *Type TemplateIdentifier* ["(" *TemplateFormalParList* ")"] [**modifies** *TemplateRef*] ":=" *TemplateBody*	**template float** *mw_t*(**out boolean** *p_a*):= {*field1*:=7, *field2*:=?->*p_a*)}; **match**(*c_value*, *mw_t*(*v_b*));	 *v_b* retrieves value if template match	5.2.2
conjunct "(" { (*TemplateInstance* | **all from** *TemplateInstance*) [","] } ")"	**template integer** *mw_t* := **conjunct** ((1..200), **@dynamic** *f_myfunc*);	all templates from the list must match	5.3.1
TemplateInstance **implies** *TemplateInstance*	**template charstring** *mw_t* := ? **length**(5..**infinity**) **implies pattern** "*ac";	match precondition and implied template	5.3.2
TemplateInstance **except** *TemplateInstance*	**template integer** *mw_t* := (1..100) **except** (48, 64);	all integer between 1 and 100 except 48 and 64	5.3.3
disjunct "(" { (*TemplateInstance* | **all from** *TemplateInstance*) [","] } ")"	**template** *MyRecOf mw_t* := {0, **disjunct** (*mw_a*, *mw_b*, ? **length**(2..4)), *}	match if one of the alternatives matches maximal subset of unmatched elements	5.3.4
TemplateInstance "#" "(" [*SingleExpression*] ["," [*SingleExpression*]] ")"	**template** *RoN mw_a* := { { { *givenName* := "John", *surname* := ?} } #(2, 5) };	matches values containing 2..5 elements; the givenName field of each element has to be equal to "John"	5.4

NOTES:

This Reference Card summarizes language features to support users of TTCN-3. The document is not part of a standard, not warranted 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/203022.

NEW Language elements introduced in edition 4.9.1 have been marked.

CONVENTIONS

BNF DEFINITIONS ([1] §A.1.1)		*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;
{abc}#(n,m)	n to m instances of *abc;*	**...**	indicates additional TTCN-3 source code;
(...)	textual grouping;	<empty>	string of zero length;
abc	the non-terminal symbol *abc;*		
"abc"	the terminal symbol *abc;*		

Selected BNF definitions have links to browseable BNF generated by https://bnftools.informatik.uni-goettingen.de.

Copyright 2010 - 2018 www.blukaktus.com. Forwarding and copying of this document is permitted for personal and educational purposes if authorship is retained and that the content is not modified. This work is not to be distributed for commercial advantage.