

Document overview:

- [ES 201 873-1 \(2011-06\)](#) TTCN-3 part 1 (edition 4.3.1): *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 \(2010-07\)](#) TTCN-3 part 4 (edition 4.2.1): *Operational Semantics (OS)*
- [ES 201 873-5 \(2011-06\)](#) TTCN-3 part 5 (edition 4.3.1): *TTCN-3 Runtime Interface (TRI)*
- [ES 201 873-6 \(2011-06\)](#) TTCN-3 part 6 (edition 4.3.1): *TTCN-3 Control Interface (TCI)*
- [ES 201 873-7 \(2011-06\)](#) TTCN-3 part 7 (edition 4.3.1): *Using ASN.1 with TTCN-3*
- [ES 201 873-8 \(2011-06\)](#) TTCN-3 part 8 (edition 4.3.1): *The IDL to TTCN-3 Mapping*
- [ES 201 873-9 \(2011-06\)](#) TTCN-3 part 9 (edition 4.3.1): *Using XML schema with TTCN-3*
- [ES 201 873-10 \(2011-06\)](#) TTCN-3 part 10 (edition 4.3.1): *TTCN-3 Documentation Comment Specification*
- [ES 202 781 \(2010-08\)](#) TTCN-3 Language Extensions (version 1.1.1): *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 \(2011-05\)](#) TTCN-3 Language Extensions (version 1.2.1): *Advanced Parameterization*
- [ES 202 785 \(2011-05\)](#) TTCN-3 Language Extensions (version 1.2.1): *Behaviour Types*
- [TS 102 995 \(2010-11\)](#) *Proforma for TTCN-3 reference test suite* (version 1.1.1)

Upcoming event



Copyright 2012 www.blukactus.com. Forwarding and copying of this document is permitted for personal and educational purposes provided that authorship is retained and that the content is not modified. This work is not to be distributed for commercial advantage.



First TTCN-3 Quick Reference Card

-- NEW: with links to TTCN-3 surveys --

For TTCN-3 edition 4.3.1 (2011-06) and extensions. (PDF version has direct links to standards)

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

Contents

Static Declarations (NEW: click [survey A](#))

1.	Structuring	2
2.	Components and communication interfaces	2
3.	Basic and user-defined data types	2
4.	Data values and templates	3

Dynamic Behaviour (NEW: click [survey B](#))

5.	Statement blocks	4
6.	Typical Programming Constructs	4
7.	Port operations and external function	5
8.	Timer and alternatives	6
9.	Dynamic configuration	6

Supporting Definitions (NEW: click [survey C](#))

10.	Predefined functions and useful types	7
11.	Optional definitions: Control part and attributes	8
12.	Character pattern	8
13.	Preprocessing macros	8

Additional Documents (NEW: click [survey D](#))

14.	Generic Naming Conventions	9
15.	Documentation tags	9
16.	XML mapping	10
17.	Extensions	11

NOTE:

This Quick 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 ttn3-qrc@blukactus.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.

Conventions

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

1. Structuring

MODULE, IMPORT, GROUP	EXAMPLES	DESCRIPTION	§
module <i>ModuleIdentifier</i> [language <i>FreeText</i> { " ", <i>FreeText</i> }] [" (" [<i>ModuleDefinitionsPart</i>] [<i>ModuleControlPart</i>] ")"]	module <i>MyTypes</i> language "TTCN-3:2011" {...} module <i>MyConfig</i> language "TTCN-3:2010" {...}	present version 4.3.1; version 4.2.1;	8.1
[<i>Visibility</i>] import from <i>ModuleId</i> ((all [<i>except</i> (" (" <i>ExceptSpec</i> ")")] [(" (" <i>ImportSpec</i> ")")]);";"	public import from <i>MyModule</i> language "XML1.1" all; friend import from <i>MyModule</i> (<i>type MyType, template all</i>); private import from <i>MyIPs</i> all <i>except</i> { <i>group myGroup</i> };	definitions visible in defining and other (importing) module; definition visible in defining and friend modules; definitions in <i>MyIPs</i> cannot be imported by other modules;	8.2.3 8.2.5
[<i>public</i>] group <i>GroupIdentifier</i> [" (" [<i>ModuleDefinition</i> [";", "]"] ")"]	group <i>myGroup</i> { <i>group mySubGroup</i> {...}; ... }	groups can only have public visibility;	8.2.2
[<i>private</i>] friend module <i>ModuleIdentifier</i> [" (" [<i>ModuleIdentifier</i>]);";"	friend module <i>MyTestSuiteA</i> ;	this module is defined to be a friend to <i>MyTestSuiteA</i> ;	8.2.4
GENERAL SYNTAX	EXAMPLES	DESCRIPTION	§
terminator (;)	<i>f_step1</i> (); <i>f_step2</i> ();	optional construct ends with ";" or next symbol is "]"	A.1.2
identifiers	<i>v_myVariable</i>	case sensitive, must start with a letter (a-z, A-Z), may contain digits (0-9) and underscore (_)	A.1.3
free text comments	/* <i>block comment</i> */ <i>f</i> !(); // <i>single line comment</i>	nested block comments not permitted; start with // and end with a newline;	A.1.4

2. Components and communication interfaces

COMPONENTS	EXAMPLES	DESCRIPTION	§
type component <i>ComponentTypeIdentifier</i> [<i>extends</i> <i>ComponentTypeIdentifier</i>] [" (" [<i>PortInstance</i> <i>VarInstance</i> <i>TimerInstance</i> <i>ConstDef</i>] ")"]	type component <i>MyPtcA</i> { port <i>MyPortTypeA myPort1</i> ; port <i>MyPortTypeA myPort2</i> ; var <i>MyVarTypeA v_var1</i> ; type component <i>MyPtcB extends MyPtcA</i> { timer <i>t_myTimer</i> ; private type component <i>MyPtcC</i> {...}; }	declarations could be used in testcase, function, etc. that runs on <i>MyPtcA</i> ; in addition to the timer, <i>MyPtcB</i> includes all definitions from <i>MyPtcA</i> ; <i>MyPtcC</i> could not be imported from other modules;	6.2.10
mtc system self	stop . <i>mtc</i> ; map (<i>myPtc:portA</i> , system : <i>portB</i>); myPort.send (<i>m_temp</i>) to self ;	reference to main test component (executes testcases); reference to test system interface component; reference to actual component	6.2.11
PORTS	EXAMPLES	DESCRIPTION	§
type port <i>PortTypeIdentifier</i> message [" (" [<i>in</i> <i>out</i> <i>inout</i>] { [<i>Message</i> <i>Type</i> [";", "]"])+ ";", "]"	type port <i>MyPortA</i> message { in <i>MyMsgA</i> ; out <i>MyMsgB</i> , <i>MyMsgC</i> ; inout <i>MyMsgD</i> ;	asynchronous communication; incoming messages to be queued; messages to send out; message allowed in both directions;	6.2.9
type port <i>PortTypeIdentifier</i> procedure [" (" [<i>in</i> <i>out</i> <i>inout</i>] { [<i>Signature</i> [";", "]"])+ ";", "]"	type port <i>MyPortA</i> procedure { out <i>MyProcedureB</i> ; in <i>MyProcedureA</i> }	synchronous communication; to call remote operation (get replies/exceptions); to get calls from other components (and sent replies/exceptions);	
PROCEDURE SIGNATURES	EXAMPLES	DESCRIPTION	§
signature <i>SignatureIdentifier</i> [" (" [<i>in</i> <i>inout</i> <i>out</i>] <i>Type ValueParIdentifier</i> [";", "]"])+ (" (<i>return</i> <i>Type</i>) <i>noBlock</i>) [<i>exception</i> (" (" <i>ExceptionTypeList</i> ")")]	signature <i>MyProcedureA</i> (in <i>integer p_myP1</i> , ...) return <i>MyType</i> exception (<i>MyTypeA</i> , <i>MyTypeB</i>); signature <i>MyProcedureB</i> (inout <i>integer p_myP2</i> , ...) noBlock ;	caller blocks until a reply or exception is received; caller does not block;	14 22.1.2

3. Basic and user-defined data types

BASIC TYPES	SAMPLE VALUES AND RANGES	SAMPLE SUB-TYPES	SUBTYPES	§
boolean	true, false	type boolean <i>MyBoolean</i> (true);	list	6.1.0 6.1.2
integer	(-infinity..-1), 0, 1, (2 .. infinity), (1-1 .. 30)	type integer <i>MyInteger</i> (-2, 0, 1..3);	list, range	
float	(-infinity..-2.783), 0.0, 2.3E-4, (1.0..3.0, not a number)	type float <i>MyFloat</i> (1.1 .. infinity);		
charstring	'<empty>', 'any', '"', <i>v_myCharstring</i> [0]; lengthof (<i>v_myCharstring</i>);	single quote-symbol: "; first character of string; length of string	list, range, length	6.1.1 6.1.2 E.2
universal charstring	char (0,0,3,179) & "more"	gamma (γ) in ISO/IEC 10646 (UTF32)	type universal charstring <i>bmpstring</i> (char (0,0,0,0) .. char (0,0,255,255))	
bitstring	'<empty>'B, '1'B, '0101'B	type bitstring <i>OneBit</i> length(1);		
hexstring	'<empty>'H, 'a'H, '0a'H, '123a'H, '0a'H	type hexstring <i>OneByte</i> length(2);	list, length	
octetstring	'<empty>'O, '00'O, '0a0b'O & '0a'O	type octetstring <i>MyOctets</i> ('AA'O, 'BB'O');		
SPECIAL TYPES	EXAMPLES	DESCRIPTION	§	
default	var default <i>v_myAltstep</i> ;	manage use of altstep (activate/deactivate)		6.2.8
address	var address <i>v_myPointer</i> ;	reference of component or SUT interface (global scope)		6.2.12
verdicttype	var verdicttype <i>v_myVerdict</i> ;	fixed values: none, pass, inconc, fail, error		6.1.0

17. Extensions

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

BEHAVIOUR TYPES	EXAMPLES	DESCRIPTION	§
type function <i>BehaviourTypeIdentifier</i> [" (<" { <i>FormalTypePar</i> [";", "]"] ">"] [" (" [{ <i>FormalValuePar</i> <i>FormalTimerPar</i> <i>FormalTemplatePar</i> <i>FormalPortPar</i> } [";", "]"])" [runs on (<i>ComponentType</i> self)] [return template] <i>Type</i>]	type function <i>MyFuncType</i> (in <i>integer</i> <i>p1</i>); function <i>f_myFunc1</i> (in <i>integer</i> <i>p1</i>) {...}; ... var <i>MyFuncType</i> <i>v_func</i> ; <i>v_func</i> := <i>f_myFunc1</i> ; ... apply (" <i>v_func</i> (0)); <i>myComponent.start</i> (apply (<i>v_func</i> (1)));	new type definition (w/o body); concrete behaviour; define formal function variable; assign concrete function; execute <i>f_myFunc1</i> ; start PTC with <i>f_myFunc1</i>	5.2 (6.2.13.1) 5.8 5.11

CONFIGURATION AND DEPLOYMENT SUPPORT	EXAMPLES	DESCRIPTION	§
configuration <i>ConfigurationIdentifier</i> [" (" [{ <i>FormalValuePar</i> <i>FormalTemplatePar</i> } [";", "]"])" runs on <i>ComponentType</i> [system <i>ComponentType</i>] <i>StatementBlock</i>	configuration <i>f_StaticConfig</i> () runs on <i>MyMtcType</i> system <i>MySystemType</i> {...} myComponent := <i>MyPTCType.create</i> static ; map (<i>myComponent</i> : <i>PCO</i> , system : <i>PCO1</i>) static ; ...}	definition outside of testcases contains static configuration; creation of component; static mapping;	5.2
testcase <i>TestcaseIdentifier</i> [" (" [{ <i>FormalValuePar</i> <i>FormalTemplatePar</i> } [";", "]"])" (runs on <i>ComponentType</i> [system <i>ComponentType</i>] execute on <i>ConfigurationType</i>) <i>StatementBlock</i>	testcase <i>TC_test1</i> () execute on <i>f_StaticConfig</i> (...) control { var configuration <i>myStaticConfig</i> ; <i>myStaticConfig</i> := <i>f_StaticConfig</i> (); execute (<i>TC_test1</i> , 2.0, <i>f_StaticConfig</i>); <i>myStaticConfig.kill</i> ; ...}	testcase to be executed with static configuration; configuration setup; run test with static configuration; configuration down;	5.7 5.3 5.8

PERFORMANCE AND REAL-TIME TESTING	EXAMPLES	DESCRIPTION	§
type port <i>PortTypeIdentifier</i> message [realtime] [" (" [<i>in</i> <i>out</i> <i>inout</i>] { [<i>Message</i> <i>Type</i> [";", "]"])+ ";", "]"	module <i>MyModule</i> {... type port <i>MyPort</i> message [realtime] {...}; type component <i>MyPTC</i> (port <i>MyPort</i> <i>myP</i> ; ...); var float <i>v_specified_send_time</i> , <i>v_sendTimePoint</i> , <i>v_myTime</i> ; ... <i>myP.receive</i> (<i>m_expect</i>) [" (" <i>TemplateInstance</i> ")"] -> timestamp <i>v_myTime</i> ; [from <i>AddressRef</i>] [value <i>VariableRef</i>] [sender <i>VariableRef</i>] [timestamp <i>VariableRef</i>]]	port qualified for timestamp; time of message receipt; wait a specified time period (in sec.); get the actual time; timestamp precision of a msec.	5.2 5.3 5.4 5.1.1 5.1.2

16. 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](#).

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

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};	
(unnamed)	<union> <simpleType> <restriction base="xsd:string"/> </simpleType> <simpleType> <restriction base="xsd:float"/> </simpleType> </union>	type union MyType { XSD.String alt_1, // predefined fieldnames XSD.Float alt_1};	7.5.3
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	<xsd:complexType name="myType"> <xsd:sequence> <xsd:any namespace="##any"/> </xsd:sequence> </xsd:complexType>	type record MyType { XSD.String elem; // predefined fieldName };	7.7
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

STRUCTURED TYPES AND ANYTYPE	SAMPLE VALUES	SAMPLE USAGE	SUBTYPES	§
type record MyRecord { float field1, MySubrecord1 field2 optional };	var MyRecord v_record := {2.0, omit }; var MyRecord v_record1 := {field1 := 0.1}; var MyRecord v_record2 := {1.0, {c_c1, c_c2}};	v_record.field1 sizeof(v_record1) isPresent (v_record.field2) false	2.0 1 false	list 6.2.1
type record of integer MyNumbers; type record length (3) of float MyThree;	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}};	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
type integer MyArray [2] [3];	var MyElements v_myElements := { MyRecord element1, float element2 optional }; type set of OneBit MySet;	v_myElements.element2 v_bits[0]		6.2.2
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;	{element1:=v_record, element2:=omit}; var MySet v_bits := {'1'B, '0'B};	'1'B	6.2.3 6.2.4
type union MyUnionType { integer alternative1, float alternative2};	var MyUnionType v_myUnion := { alternative1 := 1};	type MyKeywords MyShortList { e_key1, e_key3}; /* e_key1 < e_key2 < e_key3, e_keyA > e_keyB */		list 6.2.5
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

4. Data values and templates

TEMPLATE	EXAMPLES	DESCRIPTION	§
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);	template with parameter; parameter allows template expressions (e.g. wildcards); template variable;	15.3
template [restriction] Type TemplateIdentifier ["[" TemplateFormalParList "]"] [modifies TemplateRef] ":=" TemplateBody	isvalue (mw_template);	error in case of unspecific content, e.g. wildcards;	15.10
var template (omit) MyType m_t1; var template (value) MyType m_t2;	var template (present) MyType m_t3;	returns true, if mw_template only contains concrete value or "omit"; 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 mdw_record1 := {{1.0, 3.1}, *}; template MyRecord mdw_record1 modifies mw_record := {field2:= omit}; template MyRecord m_record2 := { complement(0.0), mw_sub ifPresent };	15.1 15.5 15.7 27.7
type record of integer MyNums;	template MyNums m_nums := {0,1,2,3,4};	template MyNums mw_nums := {0,1, permutation(2,3,4)};	15.6.3 15.7.3
type integer MyArray [2] [5];	template MyArray m_array := {{1,2,3}, {1,2,3,4}};	template MyArray mw_array := {{1,2,?}, ? length (2)};	15.7.4
type set MySet {boolean field1, charstring field2};	template MySet m_set := {true, "c"};	template MySet mw_set := {false, ("a.."r")};	15.7.2
type set of integer (1..3) MyDigits;	template MyDigits m_digits := {3,2};	template MyDigits mw_digits := subset (1,7); template MyDigits mw_digits2 := superset (1);	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";	string with 'ab' followed by any two characters followed by 'xyz' followed by none or any number of characters, followed by '0';	B.1.5
template universal charstring mw_tmpl := pattern "a*" length (2..10);	up to eight characters between first and last string element;	

DECLARATIONS	EXAMPLES	DESCRIPTION	§
const Type [ConstIdentifier [ArrayDef] ":="] ConstanExpression [","] [":"]	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 [restriction] Type VarIdentifier [ArrayDef] ":=" TemplateBody [[":"] VarIdentifier [ArrayDef] ":="] TemplateBody [":"]	var template integer v_myUndefinedInteger := ?; var template (omit) v_myArray := { 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_PARI, PX_PAR2 := 2;	test management value setting overrides specified default; parameters not importable;	8.2.1

5. Statement blocks

TESTCASE	EXAMPLES	DESCRIPTION	§
testcase <i>TestcaseIdentifier</i> "{" [{ FormalValuePar FormalTemplatePar } [","]] }" runs on ComponentType [system ComponentType] StatementBlock	testcase <i>TC_myTest</i> (in integer <i>p_myp1</i> , out float <i>p_myp2</i>) runs on <i>myptcA</i> system <i>mySUTinterface</i> { const integer <i>c_local</i> ; ...}	behaviour of the mtc; test system interface comp.; <i>c_local</i> for local use only;	16.3
FUNCTION	EXAMPLES	DESCRIPTION	§
function <i>FunctionIdentifier</i> "{" [{ FormalValuePar FormalTimerPar FormalTemplatePar FormalPortPar } [","]] }" [runs on ComponentType] [return [template] Type] StatementBlock	function <i>f_myFunctionPtcA</i> (template in <i>MyPtcA</i>) runs on <i>MyPtcA</i> return template <i>MyTemplateType</i> { timer <i>t_local</i> ; ...}; function <i>f_myFctNoRunsOn</i> () return template <i>MyTemplateType</i> (...); var template <i>MyTemplateType</i> <i>v_generictemplate</i> := <i>f_myFctNoRunsOn</i> ();	invoke from components equivalent to <i>MyPtcA</i> , parameter allows wildcards; timer for local use only; can be called from any place (no ComponentType); invoke <i>f_myFctNoRunsOn</i> ;	16.1
ALTSTEP	EXAMPLES	DESCRIPTION	§
altstep <i>AltstepIdentifier</i> "{" [{ FormalValuePar FormalTimerPar FormalTemplatePar FormalPortPar } [","]] }" [runs on ComponentType] "{" { (VarInstance TimerInstance ConstDef TemplateDef) [","] } AltGuardList "}"	altstep <i>a_default</i> (in timer <i>p_timer1</i>) runs on <i>MyPtcA</i> { var boolean <i>v_local</i> ; [<i>pcol.receive</i> () { repeat } [<i>p_timer1.timeout</i> { break } ...} var default <i>v_firstdefault</i> ; <i>v_firstdefault</i> := <i>activate</i> (<i>a_default</i>); deactivate (<i>v_firstdefault</i>);	use definitions from <i>MyPtcA</i> ; 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

6. Typical programming constructs

BRANCHES, LOOPS, ASSIGNMENTS	EXAMPLES	DESCRIPTION	§														
if (" BooleanExpression ") StatementBlock { else if (" BooleanExpression ") StatementBlock } [else StatementBlock]	if (<i>v_myBoolean</i>) { ... } else { ...};		19.2														
select (" SingleExpression ") {" [case (" SingleExpression [","] ") StatementBlock] [case else StatementBlock] "}"	select (<i>v_myString</i>) { 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 <i>v_ct</i> := 1; <i>v_ct</i> < 8; <i>v_ct</i> := <i>v_ct</i> + 1) { ... }; while (<i>v_in</i> == <i>v_out</i>) { ...}; do { ... if (...) { break }; ... } while (<i>v_in</i> != <i>v_out</i>) { ...};	variable <i>v_ct</i> not defined outside of loop;	19.4														
while (" BooleanExpression ") StatementBlock			19.5														
do StatementBlock while (" BooleanExpression ")			19.6														
break ;		exit from loop	19.12														
continue ;		next iteration of a loop	19.13														
VariableRef := " (Expression TemplateBody)	<i>v_myValue</i> := <i>v_myValue2</i> ;	basic assignment	19.1														
label LabelIdentifier	label <i>myLabel</i> ;	define label location;	19.7														
goto LabelIdentifier	goto <i>myLabel</i> ;	jump to <i>myLabel</i> ;	19.8														
AUXILIARY STATEMENTS	EXAMPLES	DESCRIPTION	§														
match (" Expression ";" TemplateInstance ")	if (match {(0,(2..4)), <i>mw_rec</i> } { ...};	evaluates expression against template;	15.9														
log (" (FreeText TemplateInstance) [","] ")	log ("expectation"; <i>m_tmpl</i> , "ok");	text output for test case 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 <i>v_verdict</i> ;	(could be none, inconc, pass, fail, error)	24														
setverdict (" SingleExpression " " (FreeText TemplateInstance) ")	setverdict (pass);	initial value is none;	24.2														
getverdict ;	<i>v_verdict</i> := getverdict ;	change current verdict (could not be improved); 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											
(...)	+	*	+	not4b	<	<<	not	and	xor	or							7.1
	-	/	-	and4b	>	>>											
		mod	&	xor4b	<@	<<@											
		rem		or4b	@>	@>											

14. Generic Naming Conventions

The following table is derived from [ETSI TS 102 995](#); Proforma for TTCN-3 Test Suite.

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

15. 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: <i>draft</i> , <i>reviewed</i> , <i>approved</i> , <i>deprecated</i>	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	a reference to a test configuration	6.2
@priority Priority	* @priority high	individual priority	6.16
@purpose [freetext]	* @purpose SUT send msg A due to receipt of msg B * requirement requirement A.x.y	explains the testcase purpose	6.7
@requirement [freetext]	* ----- */ testcase <i>TC_MyTest</i> () { ... }	a link to a requirement document	6.17
OBJECT SPECIFIC TAGS	EXAMPLES	USED FOR	§
@exception Identifier [freetext]	/** @exception MyExceptionType due to event A		6.4
@param Identifier [freetext]	/** @param <i>p_param1</i> input parameter of procedure signature MyProcedure (in integer <i>p_param1</i>);	signature	6.6
@return [freetext]	/** @return this procedure returns an octetstring	template, function, altstep, testcase	6.9
@verdict Verdict [freetext]	/** @verdict fail due to invalid parameter function <i>f_myfct1</i> () { ... }	function, altstep, testcase	6.14
@member Identifier [freetext]	/** @member <i>tc_myTimer</i> the timer within <i>MyPTC</i> type */ type component MyPTC (timer <i>tc_myTimer</i> ; ...)	struct data type, component, port, modulepar, const,	6.5

11. Optional definitions: Control part and attributes

CONTROL PART	EXAMPLES	DESCRIPTION	§
<pre>control "t" { (ConstDef TemplateDef VarInstance TimerInstance TimerStatements BasicStatements BehaviourStatements SUTStatements stop) [";"] } "</pre>	<pre>control { ... var verdictType v_myverdict1 := execute (TC_testcase1(c_value), 3.0); if (execute (TC_testcase2() != pass) {stop; } }</pre>	implicit timeout value 3.0; termination of control part;	26 26.1
ATTRIBUTES	EXAMPLES	DESCRIPTION	§
<pre>with "t" { (encode variant display extension optional) [override] ["(DefinitionRef FieldReference AllRef)"] [FreeText [";"]] } "</pre>	<pre>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"); } template MyRec m_myRec {field1 := 1} with (optional "implicit omit");</pre>	apply rule2 to MyInteger; apply rule3 to field1; GFT format details; apply rule1/extension to group;	27 27.2

12. 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}	any universal character	char(0,0,3,179) = gamma (γ) in ISO/IEC 10646	
\t	control character HT(9)	HT(9)	
\n	newline character	LF(10), VT(11), FF(12), CR(13)	
\r	control character CR	CR	
\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 characters \a only \\ refers to the single character \ only	
{reference}	reference to existing definitions, e.g. const charstrng c_mychar := "ac";	"{c_mychar}" refers to "ac"	
\N(reference)	reference to existing character set definitions e.g. const charstrng 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	dH(2,4) indicates "dd", "ddd" or "dddd"	
#n	n-times repetition	d#3 indicates "ddd"	
+	optional	d+ indicates "d", "dd", "ddd", ...	

13. Preprocessing macros

MACRO NAME	DESCRIPTION	EXAMPLES	§
__MODULE__	occurrences are replaced with module name (charstrng value)	module MyTest { const charstrng c_myConst := __MODULE__ & " & __FILE__; // becomes "MyTest:/home/mytest.ttcn"	D.1 -D.4
__FILE__	occurrences are replaced with full path and basic file name (charstrng value)		
__BFILE__	occurrences are replaced with basic file name (charstrng value without path)	const charstrng c_myConst2 := __LINE__ & " & __BFILE__; // becomes "6:mytest.ttcn"	
__LINE__	occurrences are replaced with the actual line number (integer value)		
__SCOPE__	occurrences are replaced with (charstrng 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	D.5

7. Port operations and external function

ASYNCHRONOUS COMMUNICATION (send, receive)	EXAMPLES	DESCRIPTION	§
<pre>Port "t" send (" TemplateInstance ") [to Address]</pre>	<pre>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;</pre>	inline template; multicast; broadcast;	22.2.1
<pre>(Port any port) "t" receive [(" TemplateInstance ")] [from Address Ref] ["-> [value (VariableRef (" ({ VariableRef [":" FieldOrTypeReference] [","] })"))] [sender VariableRef]]</pre>	<pre>myPort.receive(MyType:?) -> value v_in; myPort.receive(mw_template) from v_interface; myPort.receive -> sender v_address;</pre>	store incoming value; sender condition; store originator ref;	22.2.2
QUEUE INSPECTION	EXAMPLES	DESCRIPTION	§
<pre>(Port any port) "t" trigger (" TemplateInstance ") [from Address] ["-> [value (VariableRef (" ({ VariableRef [":" FieldOrTypeReference] [","] })"))] [sender VariableRef]]</pre>	<pre>myPort.trigger(MyType:?) -> value v_income;</pre>	removes all messages from queue (including the specified message with type MyType);	22.2.3
<pre>(Port any port) "t" check ["((PortReceiveOp PortGetCallOp PortGetReplyOp PortCatchOp) [from Address] ["->" sender VariableRef])]]";</pre>	<pre>myPort.check (m_template) from v_myPtc;</pre>	evaluates top element against expectation; no change of queue status;	22.4

SYNCHRONOUS COMMUNICATION (call, reply), EXCEPTIONS (raise, catch)	EXAMPLES	DESCRIPTION	§
<pre>Port "t" call (" TemplateInstance [" " CallTimerValue] ") [to Address]</pre>	<pre>signature MyProcedure (in integer p_myP1, inout float p_myP2) return integer exception (ExceptionType);</pre>	calling component; implicit timeout of 5 sec.	22.3.1
<pre>(Port any port) "t" getcall [(" TemplateInstance ")] [from Address] ["-> [param (" ({ VariableRef ":" ParameterIdentifier } ", ") / { (VariableRef [":"] " ") " " })] [sender VariableRef]]</pre>	<pre>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 {...} }</pre>	return value must be 1..9;	22.3.2
<pre>(Port any port) "t" getreply ["(TemplateInstance [value TemplateInstance] ")] [from Address] ["-> [value VariableRef] [param (" ({ VariableRef ":" ParameterIdentifier } ", ") / { (VariableRef [":"] " ") " " })] [sender VariableRef]]</pre>	<pre>... [] myPort.catch (ExceptionType:?) {...} ... [] myPort.catch (timeout) {...} };</pre>	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
<pre>Port "t" reply (" TemplateInstance [value Expression] ") [to Address]</pre>	<pre>myPort.getcall (s_myExpectation);</pre>	called component;	22.3.3
<pre>Port "t" raise (" Signature ", " TemplateInstance ") [to Address]</pre>	<pre>... if (v_failure) { myPort.raise(s_myError);...; ... myPort.reply (s_myAnswer)</pre>	raise exception	22.3.5
<pre>(Port any port) "t" catch ["(" Signature " TemplateInstance) TimeoutKeyword ")] [from Address] ["-> [value (VariableRef (" ({ VariableRef [":" FieldOrTypeReference] [","] })"))] [sender VariableRef]]</pre>		regular reply	22.3.6

EXTERNAL CALCULATION	EXAMPLES	DESCRIPTION	§
<pre>external function ExtFunctionIdentifier (" ({ FormalValuePar FormalTimerPar FormalTemplatePar FormalPortPar } [","]) ") [return Type]</pre>	<pre>external function fx_myCryptoalgo (integer p_name, ...) return charstrng;</pre>	need implementation in adapter	16.1.3

8. Timer and alternatives

TIMER DEFINITIONS AND OPERATIONS	EXAMPLES	DESCRIPTION	§
timer { TimerIdentifier [ArrayDef] := " TimerValue ["] ["] }	timer t_myTimer := 4.0;	declaration with default;	12
((TimerIdentifier TimerParIdentifier) { " SingleExpression " }) " start { " TimerValue " }	timer t_myTimerArray[2] := {1.0,2.0};	array of two timers;	23.2
((TimerIdentifier TimerParIdentifier) { " SingleExpression " }) " read	t_myTimer.start(5.0); myTimer.start;	timer started for 5 seconds; restart for 4 sec. (default);	23.4
((TimerIdentifier TimerParIdentifier) { " SingleExpression " }) " all timer	var float v_current := t_myTimer.read;	get actual timer value;	23.3
" stop	t_myTimerArray[2].stop;	stop 2 nd timer from array;	
(((TimerIdentifier TimerParIdentifier) { " SingleExpression " }) " any timer	if (any timer.running)	any timer previously started;	23.5
" running	{...}		
(((TimerIdentifier TimerParIdentifier) { " SingleExpression " }) " any timer	t_myTimer.timeout;	awaits timeout of t_myTimer	23.6
" timeout			
ALTERNATIVES	EXAMPLES	DESCRIPTION	§
alt "t" { [" 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;	20.2
interleave "t" { ["] (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

9. Dynamic configuration

COMPONENT MANAGEMENT	EXAMPLES	DESCRIPTION	§
ComponentType " create [" Expression " Expression "] [alive]	var MyPtc myInstance, myInstance2; myInstance := MyPtc.create alive;	initialize variables;	21.3.1
(VariableRef FunctionInstance) " start " (FunctionInstance)"	myInstance2 := MyPtc.create("ID"); myInstance.start(f_myFunction (v_param1,c_param2));	allocate memory; "ID" for logging only; start with f_myFunction behaviour;	21.3.2
stop ((VariableRef FunctionInstance mtc self) " stop) (all component " stop)	myInstance.stop; myInstance.start;	stops (alive keeps resources); restarts;	21.3.3
kill ((VariableRef FunctionInstance mtc self) " kill) (all component " kill)	myInstance.kill; all component.kill; stop; self.stop; mtc.stop;	stop and release resources; kills PTCs (remove resources); stops own component; stops testcase execution;	21.3.4
(VariableRef FunctionInstance any component all component) " "	myInstance.alive; myInstance.running;	checks status of specific, any or all components;	21.3.5 - 21.3.8
(alive running done killed)	any component.killed;		
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	
unmap " (" ComponentRef " Port " ComponentRef " Port ") (" PortRef ") (" ComponentRef " all port ") (" all component " all port ") [param " [{ ActualPar ["] }] "]	unmap; unmap(mtc:portA, system:portB);	unmaps all own port; unmaps mtc portA;	21.1
connect " (ComponentRef " Port " ComponentRef " Port ")	connect(self:portA, mtc:portC)	between components w/o adapter;	
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;	
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 on port and new incoming elements; but current queue elements are processed.	
(Port (all port)) " clear	myPortA.clear	removes all current elements from the port queue	

10. Predefined functions and useful types

PREDEFINED CONVERSION FUNCTIONS	EXAMPLES	DESCRIPTION	§	
int2char, int2unichar, int2str, int2float (in integer invalue) return (charstring universal charstring charstring float)	int2str(-66); int2float(4); int2bit(4,4); int2bit(4,2);	"-66" 4.0 "0100"B error	16.1.2 , C.1-C.7	
int2bit, int2hex, int2oct (in integer invalue, in integer length) return (bitstring hexstring octetstring)	float2int(3.12345E2)	312	C.8	
float2int (in float invalue) return integer	char2int, char2oct (in charstring invalue) return (integer octetstring)	54'O	C.9/C.10	
char2int, char2oct (in charstring invalue) return (integer octetstring)	unichar2int("T")	44	C.11	
unichar2int (in universal charstring invalue) return integer	bit2int, bit2hex, bit2oct, bit2str (in bitstring invalue) return (integer hexstring octetstring charstring)	"D7'H	C.12 - C.15	
bit2int, bit2hex, bit2oct, bit2str (in bitstring invalue) return (integer hexstring octetstring charstring)	hex2int, hex2bit, hex2oct, hex2str (in hexstring invalue) return (integer bitstring octetstring charstring)	"AB801'H	C.16 - C.19	
hex2int, hex2bit, hex2oct, hex2str (in hexstring invalue) return (integer bitstring octetstring charstring)	oct2int, oct2bit, oct2hex, oct2str, oct2char (in octetstring invalue) return (integer bitstring hexstring charstring charstring)	"1101011'B	C.20 - C.24	
oct2int, oct2bit, oct2hex, oct2str, oct2char (in octetstring invalue) return (integer bitstring hexstring charstring charstring)	str2int, str2hex, str2oct, str2float (in charstring invalue) return (integer hexstring octetstring float)	"01D7'O	C.25 - C.29	
str2int, str2hex, str2oct, str2float (in charstring invalue) return (integer hexstring octetstring float)	enum2int (in Enumerated_type inpar) return integer	0	C.37	
enum2int (in Enumerated_type inpar) return integer	int2enum (in integer inpar, out Enumerated_type outpar)	e_FirstElement	C.42	
int2enum (in integer inpar, out Enumerated_type outpar)	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.29
	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.30
	ispresent (in template any_type inpar) return boolean	ispresent(v_myRecord.field1)	true if optional field inpar is present (record or set only)	C.31
	ischosen (in template any_union_type inpar) return boolean	ischosen(v_receivedPDU.field2)	true if inpar is chosen within the union value/template	C.32
	isvalue (in template any_type inpar) return boolean;	isvalue(MyRec:1,omit) true	true if concrete value	C.38
	rnd (in float seed) return float;	v_randomFloat := rnd ();	random float number	C.36
	testcasename () return charstring;	v_TName := testcasename ();	current executing test case	C.41

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.34
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.35
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.33
encvalue (in template (value) any_type inpar) return bitstring	p_messageLen := lengthof(encvalue(m_payload));	functions require codec implementation;	C.39
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.40

TYPE DEFINITIONS FOR SPECIAL CODING (USE WITH OTHER NOTATIONS: ASN.1, IDL, XSD)	DESCRIPTION	§
type charstring char64 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" }; type integer short (-32768 .. 32767) with { variant "16 bit" }; type integer unsignedshort (0 .. 65535) with { variant "unsigned 16 bit" }; type integer long (-2147483648 .. 2147483647) with { variant "32 bit" }; type integer unsignedlong (0 .. 4294967295) with { variant "unsigned 32 bit" }; type integer longlong (-9223372036854775808 .. 9223372036854775807) with { variant "64 bit" }; type integer unsignedlonglong (0 .. 18446744073709551615) with { variant "unsigned 64 bit" }; type float IEEE754float with { variant "IEEE754 float" }; type float IEEE754double with { variant "IEEE754 double" }; type float IEEE754extendedfloat with { variant "IEEE754 extended float" }; type float IEEE754extendeddouble with { variant "IEEE754 extended double" }; type universal charstring utf8string with { variant "UTF-8" }; type universal charstring iso8859string (char (0,0,0) .. char (0,0,0,255)) with { variant "8 bit" }; type universal charstring bmpstring (char (0,0,0,0) .. char (0,0,255,255)) with { variant "UCS-2" }; type record IDLfixed { unsignedshort digits, short scale, charstring value. } with { variant "IDL:fixed FORMAL/01-12-01 v.2.6" };	to be encoded / decoded as they were represented on 1 byte E.2.1.0 to be encoded / decoded as they were represented on 2 bytes E.2.1.1 to be encoded / decoded as they were represented on 4 bytes E.2.1.2 to be encoded / decoded as they were represented on 8 bytes E.2.1.3 to be encoded / decoded according to the IEEE 754 E.2.1.4 encode / decode according to UTF-8 E.2.2.0 all characters defined in ISO/IEC 8859-1 E.2.2.3 BMP character set of ISO/IEC 10646 E.2.2.1 fixed-point decimal literal as defined in the IDL Syntax and Semantics version 2.6 E.2.3.0	