ETSI ES 201 873-1 V4.14.1 (2022-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 1: TTCN‑3 Core Language

**ETSI Standard**

# 16 Functions, methods, altsteps and testcases

## 16.0 General

In TTCN‑3, functions, methods, altsteps and testcases are used to specify and structure test behaviour, define default behaviour and to structure computation in a module, etc. as described in the following clauses.

## 16.1 Functions

### 16.1.0 General

Functions are used in TTCN‑3 to express test behaviour, to organize test execution or to structure computation in a module, for example, to calculate a single value, to initialize a set of variables or to check some condition.

***Syntactical Structure***

**function** [ **@deterministic | @control** ] ( *FunctionIdentifier |* **control** )

"(" [ { ( *FormalValuePar* | *FormalTemplatePar* ) [","] } ] ")"

[ **extends** *Type* ]

[ **runs** **on** *ComponentType* ]

[ **mtc** *ComponentType* ]

[ **system** *ComponentType* ]

[ **return** [ *TemplateModifier* ] *Type* [ArrayDef] ]

*StatementBlock*

***Semantic Description***

Functions are portions of TTCN-3 behaviour, which perform a specific task and are relatively independent of the remaining behaviour.

Functions may return a value or a template. Value return is denoted by the **return** keyword followed by a type expression. Template return is denoted by the **return** keywords followed by a *TemplateModifier* and a type expression. Execution of a **return** statement in the body of the function causes evaluation of the return value or template, the function to terminate and to return the result to the location of the call of the function.

The behaviour of a function can be defined by using statements and operations described in clauses Error: Reference source not found to Error: Reference source not found.

Functions may be parameterized.

A function with an **extends** clause is called a *method*(see clause 16.1.5).

Functions may have an **mtc** clause. If a function has an **mtc** clause, the type referenced by this clause shall be mtc-compatible (see clause Error: Reference source not found) with the type of the **mtc** component reference. If the mtc clause is not present, the type of the **mtc** component reference is unknown in the scope of this function.

Functions may have a **system** clause. If a function has a **system** clause, the type referenced by this clause shall be system-compatible (see clause Error: Reference source not found) with the type of the **system** component reference**.** If the system clause is not present, the type of the **system** component reference is unknown in the scope of this function.

Using the **@deterministic** modifier, a function can be declared to be deterministic. Deterministic functions are safe to be used when called from specific places where non-determinism could lead to unexpected side effects (see clause Error: Reference source not found).

NOTE 0: The determination of determinism of a function is a semi-decidable problem and as such can and will not be exhaustively checked. As such, the annotation deterministic is mainly used for informational purposes and for allowing certain functions to be used during snapshot evaluation. Principally, a function can be seen as deterministic if it does not violate any of the restrictions from clause Error: Reference source not found which does not mean that violation of these restriction automatically leads to non-determinism.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause Error: Reference source not found, the following restrictions apply:

1. A function without **runs on** clause shall never invoke a function or altstep or activate an altstep as default with a **runs on** clause locally.
2. Functions started by using the **start** test component operation shall always have a **runs on** clause (see clause Error: Reference source not found) and are considered to be invoked in the component to be started, i.e. not locally. However, the **start** test component operation may be invoked within behaviour without a **runs on** clause.

NOTE 1: The restrictions concerning the **runs on** clause are only related to functions and altsteps and not to test cases.

1. Functions called directly or indirectly from a module control function shall have no **mtc** or **system** clause.

NOTE 2: Nevertheless, functions called directly or indirectly from the module control function are allowed to execute test cases.

1. The rules for formal parameter lists shall be followed as defined in clause 5.4.
2. For **return** *TemplateModifier* statements the restrictions specified in clause Error: Reference source not found shall apply.
3. Template **return** can be restricted to the matching mechanisms specific value and **omit**, see clause Error: Reference source not found.
4. A **return** statement in a value returning function shall always have a value expression compatible to the type specified in the function header return clause.
5. A **return** statement in a template returning function shall always have a template reference (including calling a value or template returning function)or template instance compatible to the type specified in the function header return clause. If the **return** clause has a template restriction, this restriction shall be adhered to by the returned template.
6. If the function header includes a **return** clause, the function, when terminating, shall do so by executing a **return** statement. The function will cause a test case error if it terminates (i.e. reaches the end of the function body) without executing a **return** statement.
7. If a function references the names of definitions that are defined inside a component type definition, the component type shall be referenced using the **runs on** keywords in the function header. The one exception to this rule is if all the necessary component-wide information is passed in the function as parameters.
8. The additional restrictions in clause 16.1.5 shall apply to all explicit control functions.The list of statements and operations that are allowed to be used by control functions is provided by table Error: Reference source not found.

***Examples***

EXAMPLE 1: Function with return:

 // Definition of f\_myFunction which has no parameters

 **function** f\_myFunction() **return** **integer**

 {

 **return** 7; // returns the integer value 7 when the function terminates

 }

EXAMPLE 2: Function with template return:

 // Definition of functions which may return matching symbols or templates

 **function** f\_myFunction2() **return** **template** **integer**

 {

 :

 **return** ?; // returns the matching mechanism AnyValue

 }

 **function** f\_myFunction3() **return** **template** **octetstring**

 {

 :

 **return** 'FF??FF'O; // returns an octetstring with AnyValue inside it

 }

EXAMPLE 3: Function with runs on clause:

 **function** f\_myFunction3() **runs on** MyPTCType {

 // f\_myFunction3 does not return a value, but

 **var integer** v\_myVar := 5; // does make use of the port operation

 pCO1.**send**(v\_myVar); // send and therefore requires a runs on

// clause to resolve the port identifiers

 } // by referencing a component type

EXAMPLE 4: Parameterized function:

 **function** f\_myFunction2(**inout integer** p\_myPar1) {

 // f\_myFunction2 does not return a value

 p\_myPar1 := 10 \* p\_myPar1; // but changes the value of p\_myPar1 which

 } // is passed in by reference

EXAMPLE 5: Function without return statement:

 **function** f\_myFunction5(**inout integer** p\_myPar1) **return** **integer** {

 **if** (p\_myPar1 > 5) {

 p\_myPar1 := 5;

 **return** p\_myPar1;

 }

 // in case of p\_myPar1 <= 5, f\_myFunction5 does not terminate in a return statement

 // and will cause a test case error

 }

EXAMPLE 6: Function with system and mtc:

 **type component** MtcType { ... }

 **type component** SystemType **{ ... }**

 **function** f\_myFunction6() **runs** **on** MyPtcType **mtc** MtcType **system** SystemType {

 **var** MtcType v\_mtc := **mtc**;

 **var** SystemType v\_system := **system**;

 f\_myFunction3(); // allowed, f\_myFunction3() has no mtc and system clause

 f\_myFunction6(); // allowed, f\_myFunction6() has compatible mtc and system clause

 }

 **function** f\_myFunction7() **runs** **on** MyPtcType **system** SystemType {

 **var** MtcType v\_mtc := **mtc**; // not allowed, mtc type unknown

 f\_myFunction6(); // possible runtime error, no mtc clause of f\_myFunction7

 }

 **function** MyFunction8() **runs** **on** MyPtcType **mtc** MtcType {

 **var** SystemType v\_system := **system**; // not allowed, system type unknown

 f\_myFunction6(); // possible runtime error, no system clause of f\_myFunction8

 }

### 16.1.3 External functions

A function may be defined within a module or be declared as being defined externally (i.e. **external**).

***Syntactical Structure***

**external** **function** [ **@deterministic** | **@control** ] *ExtFunctionIdentifier*

"(" [ { ( *FormalValuePar* | *FormalTemplatePar* ) [","] } ] ")"

[ **extends** *Type* ]

[ **return** [ *TemplateModifier* ] *Type* ]

***Semantic Description***

For an external function only the function interface has to be provided in the TTCN‑3 module. The realization of the external function is outside the scope of the present document.

Using the **@deterministic** modifier, an external function can be declared to be deterministic. Deterministic functions are safe to be used when called from specific places where non-determinism could lead to unexpected side effects (see clause Error: Reference source not found).

The **@control** modifier is used in the same way as described in the clause 16.1.5.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause Error: Reference source not found, the following restrictions apply:

1. Restrictions on invoking functions from specific places are described in clause Error: Reference source not found.

NOTE: External functions should only exchange information with the test system via return values and parameter passing. Side-effects that change the status of the test system and may influence the test outcome should be avoided. Such side-effects can occur if an external function contains default handling, configuration, communication or timer operations.

***Examples***

 **external** **function** fx\_myFunction4() **return** **integer**; // External function without parameters

 // which returns an integer value

 **external** **function** fx\_initTestDevices(); // An external function which only has an

 // effect outside the TTCN‑3 module

### 16.1.5 Methods

A method is a function or an external function with an **extends** clause. The type specified by the **extends** clause is called the receiver type.

A method definition binds behaviour to a receiver type. The method name is local to the receiver type and shall be unique within the receiver type (but does not have to be globally unique). Methods of embedded fields can become promoted methods (see clause 6.2.1.4). Methods bound to a receiver type do not affect compatibility rules.

Methods may not be imported explicitly. Methods are implicitly imported together with their receiver type according to the visibility rules specified in 8.2.5.

Methods of a base type are automatically bound to its type synonyms (see clause 6.4). It is allowed to bind additional methods to type synonyms. These methods shall have different names than those present in the base type (i.e. overriding is not allowed). Methods bound to a type synonym are not bound to the base type and other type synonyms of the same base type.

During method invocation the receiver value is passed as the implicit **inout** parameter **this**.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5, and the restrictions on invoking functions given in clause 16.1.1, the following restrictions apply:

1. A *method* and its *receiver* *type* shall be defined in the same module.

***Examples***

**module** M1 {

 **type** **integer** A;

 **type** **integer** B;

 **function** method1() **extends** A {}

 **function** method1() **extends** B {}

 **private** **function** method2() **extends** B { **log**("receiver value:", **this**) }

}

**module** M2 {

 **import** **from** M1 **all**;

 **function** method1() **extends** M1.A {} *// ERROR: method not in same module as receiver type.*

 **type** M1.A C; // type synonym

 **function** method1() **extends** C {} // ERROR: not allowed to override a method from the base type

 **function** method3() **extends** C {} // additional method for the type synonym

 **function** F() {

 **var** A a := 1;

 **var** B b := 2;

 **var** C c := 3;

 a := c; *// a and c are compatible.*

 a.method1(); *// regular method invocation.*

 b.method2(); *// ERROR: method2 is not visible in module M2.*

 c.method1(); *// type synonym invokes a method of the base type*

 c.method3(); // regular method invocation

 a.method3(); // ERROR: method3 is available only in the type synonym C

 }

}

### A.1.5.0 General

TTCN‑3 terminal symbols and reserved words are listed in tables A.2, A.3 and A.5.

Table A.1: List of TTCN‑3 special terminal symbols

|  |  |
| --- | --- |
| Begin/end block symbols | **{ }** |
| Begin/end list symbols | **( )**  |
| Element specifier symbols | **[ ]** |
| Range symbol | **..** |
| Line and block comments | **/\* \*/**  **//** |
| Statement separator symbol | **;** |
| Arithmetic operator symbols | **+ / - \*** |
| Concatenation operator symbol | **&** |
| Relational operator symbols | **!= == >= <= < >** |
| Shift operator symbols | **<< >>** |
| Rotate operator symbols | **<@ @>** |
| String enclosure symbols | **" '**  |
| Wildcard/matching symbols | **? \***  |
| Assignment symbol | **:=**  |
| Communication operation assignment  | **->** |
| Bitstring, hexstring and Octetstring values | **B H O**  |
| Float exponent | **E** |
| List element separator symbol | **,** |
| Field reference | **.** |
| Decoded field reference | **=>** |

The predefined function identifiers defined in table Error: Reference source not found and described in annex C shall also be treated as reserved words.

Table A.2: List of TTCN‑3 terminals which are reserved words

|  |  |  |  |
| --- | --- | --- | --- |
| **action****activate****address****alive****all****alt****altstep****and****and4b****any****anytype****bitstring****boolean****break****case****call****catch****char****charstring****check****clear****complement****component****connect****const** **continue****control****create****deactivate****decmatch****default****disconnect****display****do****done****else****encode****enumerated****error****except****exception****execute****extends****extension****external** | **fail****false****float****for****friend****from****function****getverdict****getcall****getreply****goto****group****halt****hexstring****if****ifpresent****import****in****inconc****infinity****inout****integer****interleave****isbound****ischosen****ispresent****isvalue****kill****killed****label****language****length****log****map****match****message****mixed****mod****modifies****module****modulepar****mtc** | **noblock****none****not****not\_a\_number****not4b****nowait****null****octetstring****of****omit****on****optional****or****or4b****out****override****param****pass****pattern****permutation****port****present****private****procedure****public****raise****read****receive****record****recursive****rem****repeat****reply****return****running****runs** | **select****self****send****sender****set****setencode****setverdict****signature****start****stop****subset****superset****system****template****testcase****this****timeout****timer****to****trigger****true****type****union****universal****unmap****value****valueof****var****variant****verdicttype****while****with****xor****xor4b** |

The TTCN‑3 terminals listed in table A.3 shall not be used as identifiers in a TTCN‑3 module. These terminals shall be written in all lowercase letters.

Additionally, there are special TTCN-3 terminals consisting of an @-symbol, directly followed by an identifier. These terminals shall also be written in all lowercase letters.

NOTE: These terminals can be used in combination with the @-symbol, which results in a specific semantics for the annotated language element. They can also be used like any other identifier without any special meaning.

Table A.3: List of TTCN‑3 terminals which are modifiers

|  |  |  |  |
| --- | --- | --- | --- |
| **@abstract****@control** | **@decoded****@default****@deterministic****@fuzzy** | **@index****@lazy****@local** | **@nocase****@nodefault** |

Table A.4: List of TTCN‑3 terminals which are reserved words in extension packages

|  |  |  |  |
| --- | --- | --- | --- |
| **apply****assert****at****configuration****conjunct****cont****delta****disjunct****duration****finished** | **history****implies****inv****mode****notinv****now****onentry****onexit** | **par****prev****realtime****seq****setstate****static****stepsize****stream** | **timestamp****until****values****wait** |

The TTCN‑3 terminals listed in table A.4 are used as keywords inside the TTCN-3 extension packages. These terminals shall not be used as identifiers in a TTCN‑3 module..

These terminals shall be written in all lowercase letters.

## A.1.6 TTCN-3 syntax BNF productions

#### A.1.6.1.4 Function definitions

1FunctionDef ::= [FunctionKeyword](#TFunctionKeyword) [ [DeterministicModifier](#TDeterministicModifier) | [ControlModifier](#TControlModifier) ]

 [IdentifierOrControl](#TIdentifierOrControl)

 "(" [[FunctionFormalParList](#TFunctionFormalParList)] ")" [ReceiverSpec] [[RunsOnSpec](#TRunsOnSpec)] [[MtcSpec](#TMtcSpec)]

 [[SystemSpec](#TSystemSpec)] [[ReturnType](#TReturnType)] [StatementBlock](#TStatementBlock)

2FunctionKeyword ::= "function"

3FunctionFormalParList ::= [FunctionFormalPar](#TFunctionFormalPar) {"," [FunctionFormalPar](#TFunctionFormalPar)}

4FunctionFormalPar ::= [FormalValuePar](#TFormalValuePar) |

 [FormalTemplatePar](#TFormalTemplatePar)

5ReturnType ::= [ReturnKeyword](#TReturnKeyword) [[TemplateModifier](#TTemplateModifier)] [Type](#TType) [ArrayDef]

6ReturnKeyword ::= "return"

7RunsOnSpec ::= [RunsKeyword](#TRunsKeyword) [OnKeyword](#TOnKeyword) [ComponentType](#TComponentType)

8RunsKeyword ::= "runs"

9OnKeyword ::= "on"

10ReceiverSpec ::= Extends[Keyword](#TMTCKeyword) [Type](#TComponentType)

11MtcSpec ::= [MTCKeyword](#TMTCKeyword) [ComponentType](#TComponentType)

12MTCKeyword ::= "mtc"

13StatementBlock ::= "{" [[FunctionDefOrStatementList](#TFunctionStatementList)] "}"

14FunctionDefOrStatementList ::= {( [FunctionBodyDef](#TFunctionBodyDef) | [FunctionStatement](#TFunctionStatement)) [[SemiColon](#TSemiColon)]}+

15FunctionBodyDef ::= (FunctionLocalDef | FunctionLocalInst) [WithStatement]

16FunctionLocalInst ::= [VarInstance](#TVarInstance) | [TimerInstance](#TTimerInstance)

17FunctionLocalDef ::= [ConstDef](#TConstDef) | [TemplateDef](#TTemplateDef)

18FunctionStatement ::= [ConfigurationStatements](#TConfigurationStatements) |

 [TimerStatements](#TTimerStatements) |

 [CommunicationStatements](#TCommunicationStatements) |

 [BasicStatements](#TBasicStatements) |

 [BehaviourStatements](#TBehaviourStatements) |

 [SetLocalVerdict](#TSetLocalVerdict) |

 [SUTStatements](#TSUTStatements) |

 [TestcaseOperation](#TTestcaseOperation)

19FunctionInstance ::= [FunctionRef](#TFunctionRef) [ "(" [[ActualParList](#TActualParList)] ")" ]

/\* STATIC SEMANTICS – the part is only optional if the *FunctionRef* uses the *ControlKeyword* and the referenced control function has no formal parameters \*/

20FunctionRef ::= [[Identifier](#TIdentifier) [Dot](#TDot)] ([Identifier](#TIdentifier) | [PreDefFunctionIdentifier](#TPreDefFunctionIdentifier) | [ControlKeyword](#TControlKeyword) )

21PreDefFunctionIdentifier ::= [Identifier](#TIdentifier) [[CaseInsenModifier](#TCaseInsenModifier)]

/\* STATIC SEMANTICS - The *Identifier* shall be one of the pre-defined TTCN-3 function identifiers from Annex C of ES 201 873-1. *CaseInsenModifier* shall be present only if *Identifier* is "regexp". \*/

/\* STATIC SEMANTICS – if a value parameter is used, an in-line template shall evaluate to a value \*/

#### A.1.6.1.10 External function definitions

22ExtFunctionDef ::= [ExtKeyword](#TExtKeyword) [FunctionKeyword](#TFunctionKeyword) [[DeterministicModifier](#TDeterministicModifier) | [ControlModifier](#TControlModifier)]

 [Identifier](#TIdentifier) "(" [[FunctionFormalParList](#TFunctionFormalParList)] ")" [ReceiverSpec] [[ReturnType](#TReturnType)]

23ExtKeyword ::= "external"