Final draft ETSI ES 202 785 V1.7.1 (2020-02)

Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types

Reference
RES/MTS-202785v171

Keywords
conformance, testing, TTCN-3

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format at www.etsi.org/deliver.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Contents

Intellectual Property Rights 4

Foreword 4

Modal verbs terminology 4

1 Scope 5

2 References 5
2.1 Normative references 5
2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 6
3.1 Terms 6
3.2 Symbols 6
3.3 Abbreviations 6

4 Package conformance and compatibility 6

5 Package concepts for the core language 7
5.1 Extension to ETSI ES 201 873-1, clause 5 (Basic language elements) 7
5.2 Extension to ETSI ES 201 873-1, clause 6 (Types and values) 8
5.3 Extension to ETSI ES 201 873-1, clause 7 (Expressions) 14
5.4 Extension to ETSI ES 201 873-1, clause 8 (Modules) 14
5.5 Extension to ETSI ES 201 873-1, clause 10 (Declaring constants) 14
5.6 Extension to ETSI ES 201 873-1, clause 11 (Declaring variables) 14
5.7 Extension to ETSI ES 201 873-1, clause 15 (Declaring templates) 15
5.8 Extension to ETSI ES 201 873-1, clause 16 (Functions, altsteps and test cases) 15
5.9 Extension to ETSI ES 201 873-1, clause 19 (Basic program statements) 16
5.10 Extension to ETSI ES 201 873-1, clause 20 (Statements and operations for alternative behaviours) 17
5.11 Extension to ETSI ES 201 873-1, clause 21 (Configuration Operations) 17
5.12 Extension to ETSI ES 201 873-1, clause 26 (Module control) 18
5.13 Extension to ETSI ES 201 873-1, annex A (BNF and static semantics) 19
5.13.0 New keywords and TTCN-3 syntax BNF productions 19
5.13.1 Changes to ETSI ES 201 873-1, clause A.1.6 (TTCN-3 syntax BNF productions) 19

6 Package semantics 20
6.1 Replacements 20
6.2 Activate statement 20
6.3 Replacements in Execute statement's operational semantics 21
6.3.0 Execute statement 21
6.3.1 Flow graph segment <execute-without-timeout> 22
6.4 Replacements in Function call's operational semantics 23
6.4.0 Function call 23
6.4.1 Flow graph segment <user-def-func-call> 25
6.4.2 Flow graph segment <predef-ext-func-call> 25
6.5 Start component operation 25

7 TRI extensions for the package 28

8 TCI extensions for the package 28
8.1 Extensions to ETSI ES 201 873-6, clause 7 (TTCN-3 control interface and operations) 28
8.2 Extensions to ETSI ES 201 873-6, clause 8 (Java language mapping) 29
8.3 Extensions to ETSI ES 201 873-6, clause 9 (ANSI C language mapping) 30
8.4 Extensions to ETSI ES 201 873-6, clause 10 (C++ language mapping) 30
8.5 Extensions to ETSI ES 201 873-6, clause 11 (W3C XML mapping) 32

9 Extensions to TTCN-3 documentation comment specification for the package 42
9.1 Extensions to ETSI ES 201 873-10, annex A (where Tags can be used) 42

History 44

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

The use of underline (additional text) highlights the differences between base document and extended documents.

The present document relates to the multi-part standard ETSI ES 201 873 covering the Testing and Test Control Notation version 3, as identified in ETSI ES 201 873-1 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document defines the Behaviour Types package of TTCN-3. TTCN-3 can be used for the specification of all types of reactive system tests over a variety of communication ports. Typical areas of application are protocol testing (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of APIs, etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing. The specification of test suites for physical layer protocols is outside the scope of the present document.

TTCN-3 packages are intended to define additional TTCN-3 concepts, which are not mandatory as concepts in the TTCN-3 core language, but which are optional as part of a package which is suited for dedicated applications and/or usages of TTCN-3.

This package defines types for behaviour definitions in TTCN-3.

While the design of TTCN-3 package has taken into account the consistency of a combined usage of the core language with a number of packages, the concrete usages of and guidelines for this package in combination with other packages is outside the scope of the present document.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference/.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[2] ETSI ES 201 873-4: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics".

[3] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)".

[4] ETSI ES 201 873-6: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)".

[5] ISO/IEC 9646-1: "Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework; Part 1: General concepts".

[6] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization".

[7] ETSI ES 201 873-10: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".

[i.2] ETSI ES 201 873-8: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping".

[i.3] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI ES 201 873-1 [1], ETSI ES 201 873-4 [2], ETSI ES 201 873-5 [3], ETSI ES 201 873-6 [4], ISO/IEC 9646-1 [5] and the following apply:

behaviour definition: definition of an altstep, function, or testcase that can be called explicitly

NOTE: A control part is not considered a behaviour definition, because it cannot be called explicitly.

behaviour type: type of behaviour definitions

NOTE: Behaviour types are of kind altstep, function, or testcase.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI ES 201 873-1 [1], ETSI ES 201 873-4 [2], ETSI ES 201 873-5 [3], ETSI ES 201 873-6 [4] and ISO/IEC 9646-1 [5] apply.

4 Package conformance and compatibility

The package presented in the present document is identified by the package tag:

`"TTCN-3:2009 Behaviour Types"` - *to be used with modules complying with the present document*

For an implementation claiming to conform to this package version, all features specified in the present document shall be implemented consistently with the requirements given in the present document; in ETSI ES 201 873-1 [1] and ETSI ES 201 873-4 [2]. All features marked [AdvancedParameterization] have to be implemented only in case that this package is used together with the Advanced Parameterization package [6].

The package presented in the present document is compatible with:

- ETSI ES 201 873-1 [1] (V4.5.1).
- ETSI ES 201 873-4 [2] (V4.4.1).
- ETSI ES 201 873-5 [3] (V4.5.1).
- ETSI ES 201 873-6 [4] (V4.5.1).
- ETSI ES 201 873-7 [i.1] (V4.5.1).
- ETSI ES 201 873-8 [i.2] (V4.5.1).
- ETSI ES 201 873-9 [i.3] (V4.5.1).
- ETSI ES 201 873-10 [7] (V4.5.1).

If later versions of those parts are available and should be used instead, the compatibility to the package presented in the present document has to be checked individually.

The package presented in the present document is also compatible with:

- ETSI ES 202 784 [6] Package Advanced Parameterization (V1.3.1);

and can be used together with this package.

If later versions of those packages are available and should be used instead, the compatibility to the package presented in the present document has to be checked individually.

5 Package concepts for the core language

5.1 Extension to ETSI ES 201 873-1, clause 5 (Basic language elements)

Clause 5.4 Parameterization

Values of behaviour types can be passed as parameters as indicated in table 2.

Table 2: Overview of parameterizable TTCN-3 objects

Keyword	Allowed kind of Parameterization	Allowed form of Parameterization	Allowed types in formal parameter lists
module	Value parameterization	Static at start of run-time	all basic types, all user-defined types and **`address`** type.
type (see note)	Value parameterization	Static at compile-time	all basic types, all user-defined types and **`address`** type.
template	Value and template parameterization	Dynamic at run-time	all basic types, all user-defined types, **`address`** type, **`template`**, and behaviour types.
function	Value, template, port and timer parameterization	Dynamic at run-time	all basic types, all user-defined types, **`address`** type, **`component`** type, **`port`** type, **`default`**, behaviour types, **`template`** and **`timer`**.
altstep	Value, template, port and timer parameterization	Dynamic at run-time	all basic types, all user-defined types, **`address`** type, **`component`** type, **`port`** type, **`default`**, behaviour types, **`template`** and **`timer`**.
testcase	Value, template, port and timer parameterization	Dynamic at run-time	all basic types and of all user-defined types, **`address`** type, **`template`**, and behaviour types.
signature	Value and template parameterization	Dynamic at run-time	all basic types, all user-defined types and **`address`** type, **`component`** type, and behaviour types.
NOTE: Record of, set of, enumerated, port, component and sub-type definitions do not allow parameterization.			

Clause 5.4.1.1 Formal parameters of kind value

Also, values of behaviour types can be passed as value parameters.

5.2 Extension to ETSI ES 201 873-1, clause 6 (Types and values)

Behaviour types such as **`altstep`**, **`function`**, and **`testcase`** may be used to define flexible behaviour of TTCN-3 libraries.

No subtyping is defined for behaviour types.

Clause 6.2 Structured Types and Values

Extend clause 6.2 Structured types and values by the following clause 6.2.13.

6.2.13 Behaviour Types

6.2.13.1 Behaviour Type Definitions

Behaviour types are the set of identifiers of altstep, function, and testcase definitions with a specific parameter list, runs on, system, mtc and return clauses. They denote those altsteps, functions, and testcases, respectively, defined in the test suite that have a compatible parameter list and compatible **`runs on`** or **`system`** clauses.

Syntactical Structure

```
type  function  BehaviourTypeIdentifier
[ "<" { FormalTypePar [","] } ">" ] NOTE 1
"(" [ { ( FormalValuePar | FormalTimerPar | FormalTemplatePar | FormalPortPar ) [","] } ] ")"
[ runs on ( ComponentType | self ] NOTE 2
[ system ComponentType ]
[ mtc ComponentType ]
[ return [ template ] Type ]

type altstep BehaviourTypeIdentifier
[ "<" { FormalTypePar [","] } ">" ] NOTE 1
"(" [ { ( FormalValuePar | FormalTimerPar | FormalTemplatePar | FormalPortPar ) [","] } ] ")"
[ runs on ( ComponentType | self ] NOTE 2
[ system ComponentType ]
[ mtc ComponentType ]

type testcase BehaviourTypeIdentifier
[ "<" { FormalTypePar [","] } ">" ] NOTE 1
"(" [ { ( FormalValuePar | FormalTemplatePar ) [","] } ] ")"
runs on ComponentType
[ system ComponentType ]
```

Semantic Description

Behaviour types define prototypes of altsteps, functions, and testcases.

NOTE 1: [AdvancedParameterization] If the advanced parameterization package [6] is also supported, behaviour types can have type parameters.

NOTE 2: **`runs on self`** indicates a specific compatibility check, see extension of clause 6.3.

Restrictions

a) The rules for formal parameter lists shall be followed as defined in the TTCN-3 Core Language [1] clause 5.4 and extended in clause 5.1 of the present document.

b) Behaviour types of kind **`altstep`** may have a **`runs on`**, **`system`** or **`mtc`** clause, behaviour types of type **`function`** may have a **`runs on`**, **`system`**, **`mtc`** or **`return`** clause, behaviour types of kind **`testcase`** shall have a **`runs on`** clause and may have a **`system`** clause.

c) **`runs on self`** shall not be used for test cases.

Examples

EXAMPLE 1: Function type with one parameter and a return value.

```
type function MyFunc1 ( in integer p1 ) return boolean;
```

EXAMPLE 2: Function type with one parameter, a **runs on** clause and a return value.

```
type function MyFunc2 ( in integer p1 ) runs on MyCompType return boolean;
```

EXAMPLE 3: Altstep type with a type parameter and a **runs on** clause.

```
type altstep MyAltstep1<type T> ( in T p1 ) runs on MyCompType;
```

EXAMPLE 4: Testcase type without parameter, with a **runs on** clause and a **system** clause.

```
type testcase MyTestcase1 ( ) runs on MyCompType system MySysType;
```

6.2.13.2 Behaviour Values

The values of a behaviour type are the identifiers of altsteps, functions, and testcases with compatible parameters and **runs on, system**, **mtc** and **return** clauses. Both predefined and user-defined, including external, functions can be used as values. Type compatibility of behaviour types is defined in the extension to clause 6.3.5 within the present document.

Syntactical Structure

```
VariableRef | FunctionInstance | FunctionRef | AltstepRef | TestcaseRef | null
```

Semantic Description

The literal behaviour values are the identifiers of the predefined and user-defined altsteps, (external) functions, and testcases and the special value **null**. The special value **null** is available to indicate an undefined behaviour value, e.g. for the initialization of variables. Behaviour values can be passed around as parameters and behaviour values can be stored. Behaviour values can be used, together with a corresponding list of actual parameters, to invoke the behaviours in statements and expressions. Behaviour values can also be used, again together with a corresponding list of actual parameters, in **activate**, **start**, and **execute** statements, respectively.

The only operators (see clause 7.1 of the TTCN-3 core language [1]) on behaviour values that are defined are the check for equality and inequality.

Restrictions

a) Values of a behaviour type with a **runs on self** clause shall not be sent to another test component.

b) Values of a behaviour type with a **runs on self** clause shall not be used in a start test component operation.

c) The special value **null** shall not be used to invoke a behaviour.

Examples

```
type function MyFunc3 ( in integer p1 ) return charstring;
function blanks (in integer p1) return charstring {
    // return a charstring of p1 blank characters
}
var MyFunc3 myVar1 := blanks;
var MyFunc3 myVar1 := int2char;
```

6.2.13.3 Deferred Behaviour Type Definitions

Deferred behaviour types are the set of altstep, function, and testcase behaviours with possible runs on, system and mtc clauses paired with their actual parameter list which can be passed to the **activate**, **start** and **execute** operations, respectively. They denote those instantiations of altsteps, functions, and testcases defined in the test suite that have compatible **runs on**, **system** or **mtc** clauses.

Syntactical Structure

```
type  function  BehaviourTypeIdentifier
[ runs on ComponentType ]
[ system ComponentType ]
[ mtc ComponentType ]

type altstep BehaviourTypeIdentifier
[ runs on ComponentType ]
[ system ComponentType ]
[ mtc ComponentType ]

type testcase BehaviourTypeIdentifier
runs on ComponentType
[ system ComponentType ]
```

Semantic Description

Deferred behaviour types define references to executable behaviours. Deferred function behaviour types define the set of behaviours that can be started on a ptc. Deferred altstep behaviour types define the set of behaviours that can be activated as default alternatives. Deferred testcase behaviour types define the set of behaviours that can be executed from the control part.

The actual parameters of a deferred behaviour value are evaluated when creating the value.

Restrictions

a) The same restrictions apply for the start, activate and execute operations that apply for functions, altsteps and testcases used directly in these statements in a non-deferred way.

Examples

EXAMPLE 1: Deferred function type with.

```
type function MyFunc1;
```

EXAMPLE 2: Deferred function type with a **runs on** clause.

```
type function MyFunc2 runs on MyCompType;
```

EXAMPLE 3: Deferred altstep type with a **runs on** clause.

```
type altstep MyAltstep1 runs on MyCompType;
```

EXAMPLE 4: Deferred testcase type with a **runs on** clause and a **system** clause.

```
type testcase MyTestcase1 runs on MyCompType system MySysType;
```


6.2.13.4 Deferred Behaviour Values

The values of a deferred behaviour type are the un-executed instantiations of altsteps, functions, and testcases together with their compatible actual parameters which have compatible runs on, system and mtc clauses to those in the behaviour type. Type compatibility of deferred behaviour types is defined in the extension to clause 6.3.5 within the present document.

Syntactical Structure

```
VariableRef | FunctionInstance
```

Semantic Description

The literal deferred behaviour values are the function instance expressions of the user-defined altsteps, (external) functions, and testcases. Deferred behaviour values can be passed around as actual parameters or stored in a variable. Deferred altstep, function or testcase behaviour values can be used in **`activate`**, **`start`**, and **`execute`** statements, respectively.

The only operators (see clause 7.1 of the TTCN-3 core language [1]) on deferred behaviour values that are defined are the check for equality and inequality.

Restrictions

a) Only in functions, altsteps and tescases without inout or out parameters shall be used as deferred behaviour values.

b) For the actual parameters of a function behaviour value instance, restrictions from clause 21.3.2 of the core language specification shall apply.

c) For the actual parameters of an alstep behaviour value instance, restrictions from clause 20.5.2 of the core language specification shall apply.

d) For the actual parameters of a testcase behaviour value instance, restriction from clause 26.1 of the core language specification shall apply.

Examples

EXAMPLE 1:

```
type function MyFunctionBehaviour;
function MyFunction (in integer p1) {
    // …
}
var MyFunctionBehaviour v_myBehaviour := MyFunction(4); // does not invoke MyFunction(4)
var PtcType ptc := PtcType.create;
ptc.start(myBehaviour); // starts MyFunction(4) on component ptc
```

EXAMPLE 2:

```
type altstep MyAltstepType runs on PtcType;
altstep MyAltstep(integer i) runs on PtcType {
[] …
}
var MyAltstepType v_myAltstep := MyAltstep(3);
activate(v_myAltstep);
```

EXAMPLE 3:

```
type testcase MyTestcaseBehaviour runs on PtcType;
testcase MyTestcase(integer i) runs on PtcType { … }
function runTestCase(MyTestcaseBehaviour p_myTestcase) {
    execute(p_myTestcase, 1.0);
}
control {
    var MyTestcaseBehaviour v_myTestcase := MyTestcase(3);
    runTestCase(v_myTestcase);
    runTestCase(MyTestcase(4)); // execution is deferred to runTestCase
}
```


Clause 6.3 Type compatibility

Clause 6.3 Type compatibility is extended by:

6.3.5 Type compatibility of behaviour types

Altsteps are only compatible to **`altstep`** behaviour types, functions are only compatible to **`function`** behaviour types, testcases are only compatible to **`testcase`** behaviour types. A behaviour object (an altstep, function or testcase) is a value of a given behaviour type, if the parameter lists are compatible, if the return clause is compatible, if the **`runs on`**, **`mtc`** and **`system`** clauses are compatible, provided they exist, and if modifiers (optionally) declared for the behaviour object and the given behaviour type are identical.

The parameter list of an object is compatible with the parameter list of a type if the order of the parameters is identical, as well as direction, kind, type, name of the parameters, optional modifiers and whether a default exists. If the parameter is of kind **`template`**, then also potential template restrictions have to be identical. Compatibility of parameter lists applies to the type parameter list, if exists (i.e. when the advanced parameterization package [6] is also supported and a type parameter list is defined), and the value parameter list separately.

The return clause of a function is compatible with the return clause of a function type if it is either absent in case the function type does not have a return clause, or it is of identical kind and type if the function type has a return clause. In case the return clause is of kind **`template`**, then potential template restrictions have to be identical, too.

The **runs on** clause of an object is compatible with the **runs on** clause of a behaviour type, if it is either absent in the object or, if the **runs on** clause exists, the component type in the **runs on** clause of the object is runs on compatible (see ETSI ES 201 873-1 [1], clause 6.3.3, section 2)) with the type specified in the **runs on** clause of the behaviour type. According to the first condition, an object without a **runs on** clause is compatible with a behaviour type that has a **runs on** clause.

The **system** clause of a **testcase** object is compatible with the **system** clause of a **testcase** type, if the component type in the **system** clause of the **testcase** object is system compatible (see ETSI ES 201 873-1 [1], clause 6.3.3, section 4)) with the type specified in the **system** clause of the **testcase** type. If the **testcase** object or the **testcase** behaviour type does not have a **system** clause, then the component type of the **runs on** clause is used instead (see ETSI ES 201 873-1 [1], clause 9.2).

The **system** clause of a function or altstep object is compatible with the **system** clause of a function or altstep behaviour type, if it is either absent in the function or altstep object or, if the **system** clause exists, the component type in the **system** clause of the function or altstep object is system compatible (see ETSI ES 201 873-1 [1], clause 6.3.3, section 4)) with the type specified in the **system** clause of the function or altstep behaviour. According to the first condition, a function or altstep object without a **system** clause is compatible with a function or altstep behaviour type that has a **system** clause.

The **mtc** clause of an object is compatible with the **mtc** clause of a behaviour type, if it is either absent in the object or, if the **mtc** clause exists, the component type in the **mtc** clause of the object is mtc compatible (see ETSI ES 201 873-1 [1], clause 6.3.3, section 3)) with the type specified in the **mtc** clause of the behaviour type. According to the first condition, an object without an **mtc** clause is compatible with a behaviour type that has an **mtc** clause.

For functions it does not matter whether the function is an external function. If the parameter lists are compatible, then an external function is also compatible with the function behaviour type.

EXAMPLE:

```
// Given
type component MyCompType0 { };
type component MyCompType1 { integer a };
type component MyCompType2 { integer a;
                             float   b };
type function MyFunc ( in integer p1 ) runs on MyCompType1 return boolean;
type function MyFuncNoRunsOn ( in integer p1 ) return boolean;

//compatible with MyFunc, identical parameterlist, runs on clause, return type
function f1 (in integer p1) runs on MyCompType1 return boolean { /*…*/ };
//compatible with MyFunc, component type in type extends the one of the function
function f2 (in integer p1) runs on MyCompType0 return boolean { /* … */ }
//compatible with MyFunc, function does not have runs on clause
function f3 (in integer p1) return boolean { /* … */ }

//incompatible with MyFunc, additional parameter without default value
function g1 (in integer p1, in boolean p2) runs on MyCompType1 return boolean { /*…*/ };
//incompatible with MyFunc, missing return clause
function g2 (in integer p1) runs on MyCompType1 { /*…*/ };
//incompatible with MyFunc, different kind of parameter
function g3 (in template integer p1) runs on MyCompType1 return boolean { /*…*/ };
//incompatible with MyFunc, component type of function is an extension of the function type
function g4 (in integer p1) runs on MyCompType2 return boolean { /* … */ }
//incompatible with MyFuncNoRunsOn, function has runs on clause, type does not have one
function g5 (in integer p1) runs on MyCompType1 return boolean { /* … */ }
```

6.3.6 Type compatibility of behaviour types with runs on self

Function and altstep types can be defined with a **runs on self** clause. If a specific value is assigned to a variable or parameter of such a behaviour type, then the **runs on** clause of the enclosing definition is used in the compatibility check.

A value of a behaviour type with a **runs on self** clause can always be used in a function or altstep invocation.