



**Methods for Testing and Specification (MTS);
The Testing and Test Control Notation version 3;
TTCN-3 Language Extensions:
Configuration and Deployment Support**

[ETSI ES 202 781 V1.9.1 \(2022-02\)](https://standards.iteh.ai/catalog/standards/sist/ca1d2984-7755-4fce-9a6a-bb72fbd0a03a/etsi-es-202-781-v1-9-1-2022-02)
<https://standards.iteh.ai/catalog/standards/sist/ca1d2984-7755-4fce-9a6a-bb72fbd0a03a/etsi-es-202-781-v1-9-1-2022-02>

ReferenceRES/MTS-202781v191

Keywordsprotocol, 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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

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/CommitteeSupportStaff.aspx><https://standards.iteh.ai/catalog/standards/sist/cal d2984-7755-4fc>**Notice of disclaimer & limitation of liability**

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.

The content of the PDF version shall not be modified without the written authorization of ETSI.

The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2022.

All rights reserved.

Contents

Intellectual Property Rights	6
Foreword.....	6
Modal verbs terminology.....	6
1 Scope	7
2 References	7
2.1 Normative references	7
2.2 Informative references.....	7
3 Definition of terms, symbols and abbreviations.....	8
3.1 Terms.....	8
3.2 Symbols.....	8
3.3 Abbreviations	8
4 Package conformance and compatibility.....	8
5 Package Concepts for the Core Language.....	9
5.0 General	9
5.1 Static configurations.....	10
5.1.1 The special configuration type: configuration	10
5.1.2 The configuration function	10
5.1.3 Starting a static test configuration.....	12
5.1.4 Destruction of static test configurations.....	12
5.1.5 Creation of static test components.....	13
5.1.6 Establishment of static connections and static mappings.....	14
5.1.7 Test case definitions for static test configuration.....	14
5.1.8 Executing test cases on static test configurations	15
5.1.9 Further restrictions.....	17
5.1.10 Logging the status of static configurations	17
5.2 Ports with translation capability.....	17
5.2.0 General.....	17
5.2.1 Translation capability in port type declaration.....	19
5.2.2 Mapping and connecting ports.....	20
5.2.3 Translation functions	20
5.2.4 Translation state	21
5.2.5 Sending	23
5.2.6 Receiving	23
5.2.7 Address	25
5.2.8 Clear, start, stop and halt operation	25
5.2.9 The outer port reference.....	26
5.3 Parallel Control Components	27
5.3.0 General.....	27
5.3.1 Component Operations	28
5.3.2 Port Operations	28
5.3.3 Alt and Interleave Statements	29
5.3.4 Test Case Execution.....	29
5.3.5 MTC Reference and Clause	29
5.3.6 Changes to the Test Control Interface TCI.....	29
6 Package Semantics	30
6.0 General	30
6.1 Replacement of short forms.....	31
6.2 Order of replacement steps.....	32
6.3 Flow graph representation of TTCN-3 behaviour	33
6.4 Flow graph construction procedure	33
6.5 Flow graph representation of configuration functions.....	34
6.6 Retrieval of start nodes of flow graphs.....	34
6.7 Module state	35

6.8	Accessing the module state	35
6.9	Configuration state	35
6.10	Accessing the configuration state	35
6.11	Entity states	36
6.12	Accessing entity states.....	38
6.13	Handling of connections among ports	39
6.14	Handling of port states	40
6.15	Void.....	41
6.16	Evaluation phases, general	41
6.17	Phase I: Initialization.....	41
6.18	Phase II: Update	42
6.19	Phase III: Selection.....	42
6.20	Phase IV: Execution	43
6.21	Global functions	43
6.22	Clear port operation.....	44
6.23	Configuration function call.....	44
6.24	Connect operation.....	45
6.25	Create operation	46
6.26	Flow graph segment <disconnect-all>.....	48
6.27	Flow graph segment <disconnect-comp>.....	49
6.28	Flow graph segment <disconnect-port>	50
6.29	Flow graph segment <disconnect-two-par-pairs>	50
6.30	Execute statement.....	51
6.31	Flow graph segment <execute-without-config>.....	52
6.32	Flow graph segment <execute-on-config>.....	52
6.33	Flow graph segment <execute-on-config-without-timeout>	52
6.34	Flow graph segment <execute-on-config-timeout>.....	53
6.35	Flow graph segment <statement-block>	55
6.36	Halt port operation.....	56
6.37	Kill component operation	57
6.38	Flow graph segment <kill-mtc>	59
6.39	Flow graph segment <kill-all-comp>	59
6.40	Kill execution statement.....	61
6.41	Kill configuration operation	62
6.42	Map operation	62
6.43	Start port operation.....	63
6.44	Stop component operation.....	64
6.45	Flow graph segment <stop-mtc>.....	66
6.46	Flow graph segment <stop-config>.....	67
6.47	Flow graph segment <stop-tc-config>.....	68
6.48	Stop port operation	69
6.49	Flow graph segment <unmap-all>.....	70
6.50	Flow graph segment <unmap-comp>.....	71
6.51	Flow graph segment <unmap-port>	72
7	TRI Extensions for the Package	72
7.1	Changes and extensions to clause 5.5.2 of ETSI ES 201 873-5 Connection handling operations.....	72
7.2	Extensions to clause 6 of ETSI ES 201 873-5 Java™ language mapping.....	74
7.3	Extensions to clause 7 of ETSI ES 201 873-5 ANSI C language mapping.....	74
7.4	Extensions to clause 8 of ETSI ES 201 873-5 C++ language mapping.....	74
7.5	Extensions to clause 9 of ETSI ES 201 873-5 C# language mapping.....	75
8	TCI Extensions for the Package	75
8.1	Extensions to clause 7.2.1.1 of ETSI ES 201 873-6 Management	75
8.2	Extensions to clause 7.3.1.1 of ETSI ES 201 873-6 TCI-TM required	75
8.3	Extensions to clause 7.3.1.2 of ETSI ES 201 873-6 TCI-TM provided	77
8.4	Extensions to clause 7.3.3.1 of ETSI ES 201 873-6 TCI-CH required	77
8.5	Extensions to clause 7.3.3.2 of ETSI ES 201 873-6 TCI CH provided	78
8.6	Extensions to clause 7.3.4 of ETSI ES 201 873-6 TCI-TL provided	80
8.7	Extensions to clause 8 of ETSI ES 201 873-6 Java™ language mapping.....	82
8.8	Extensions to clause 9 of ETSI ES 201 873-6 ANSI C language mapping.....	84
8.9	Extensions to clause 10 of ETSI ES 201 873-6 C++ language mapping.....	85

8.10	Extensions to clause 11 of ETSI ES 201 873-6 W3C® XML mapping.....	88
8.11	Extensions to clause 12 of ETSI ES 201 873-6 C# language mapping.....	90
8.12	Extensions to Annex A (normative) of ETSI ES 201 873-6 IDL Specification of TCI.....	92
Annex A (normative): BNF and static semantics		93
A.1	Additional TTCN-3 terminals	93
A.2	Modified TTCN-3 syntax BNF productions	93
A.3	Additional TTCN-3 syntax BNF productions	94
Annex B (informative): Library of useful types		96
B.1	Limitations	96
B.2	Useful TTCN-3 types	96
B.2.1	Status values for port states	96
History		97

iTech STANDARD PREVIEW (standards.iteh.ai)

[ETSI ES 202 781 V1.9.1 \(2022-02\)](https://standards.iteh.ai/catalog/standards/sist/ca1d2984-7755-4fce-9a6a-bb72fbd0a03a/etsi-es-202-781-v1-9-1-2022-02)
<https://standards.iteh.ai/catalog/standards/sist/ca1d2984-7755-4fce-9a6a-bb72fbd0a03a/etsi-es-202-781-v1-9-1-2022-02>

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

(standards.iteh.ai)

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.

<https://standards.iteh.ai/catalog/standards/sist/cal d2984->

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

2022-02

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 Configuration and Deployment Support 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 the TTCN-3 support for static test configurations.

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".

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".
- [i.4] 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".

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] and ISO/IEC 9646-1 [5] apply.

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], ISO/IEC 9646-1 [5] and the following apply:

MTC	Main Test Component
PTC	Parallel Test Component

4 Package conformance and compatibility

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

"TTCN-3:2009 Static Test Configurations" - 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 in ETSI ES 201 873-4 [2].

The package presented in the present document is compatible to:

- ETSI ES 201 873-1 [1] version 4.9.1;
- ETSI ES 201 873-4 [2] version 4.6.1;
- ETSI ES 201 873-5 [3] version 4.8.1;
- ETSI ES 201 873-6 [4] version 4.9.1;
- ETSI ES 201 873-7 [i.1];
- ETSI ES 201 873-8 [i.2];

- ETSI ES 201 873-9 [i.3];
- ETSI ES 201 873-10 [i.4].

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.

5 Package Concepts for the Core Language

5.0 General

This package defines the TTCN-3 means to define *static test configurations*. A static test configuration is a test configuration with a lifetime that is not bound to a single test case. The test components of a static test configuration may be used by several test cases. This package realizes the following concepts:

- A special *configuration function* is introduced which can only be called in the control part of a TTCN-3 module to create *static test configurations*. The configuration function returns a handle of the predefined type **configuration** to access an existing static test configuration.
- A static test configuration consists of *static test components*, a test system interface, *static connections* and *static mappings*. These constituents have the following semantics:
 - A *static test component* is a special kind of test component that can only be created during the creation of a static test configuration and can only be destroyed during the destruction of a static test configuration. By definition, the MTC of a static test configuration is a static test component.
 - The test system interface of a static test configuration plays the same role as the test system interface of a test configuration created by a test case.
 - A *static connection* is a connection between static test components. It can only be established during the creation of a static test configuration and only be destroyed during the destruction of a static test configuration.
 - A *static mapping* is a mapping of a port of a static test component to a port of the test system interface of a static test configuration. Such a mapping can only be established during the creation of a static test configuration and only be destroyed during the destruction of a static test configuration.
- A static test configuration can be used by several test cases. For this the test case is started on a previously created static test configuration. This means:
 - The body of the test case is executed on the MTC of the static test configuration.
 - The MTC may start behaviour on other static test components of the static test configuration.
 - Static test components may create, start, stop and kill normal and alive test components. The lifetime of these components is bound to the actual test case that is executed on the static test configuration. In case that a normal and alive test component is not destroyed explicitly by another test component, it is implicitly destroyed when the test case ends.
 - During test case execution non-static connections and non-static mappings may be established. The lifetime of non-static connections and non-static mappings is bound to the actual test case that is executed on the static test configuration. In case that a non-static connection or a non-static mapping is not destroyed explicitly by another test component, it is implicitly destroyed when the test case ends.
- Component timers and variables of static test components are not reset or reinitialized when a test case is started on a static test configuration. They remain in the same state as when they were left after the creation of the static test configuration or after the termination of a previous test case. This allows to transfer information from one test case to another.

- Ports of static test components are not emptied or restarted when a test case is started on a static test configuration. For example, this allows a delayed handling of SUT responses like e.g. repetitive status messages, during the test campaign. In addition, all port operations (i.e. **clear**, **start**, **stop** and **halt**) are disallowed for ports of static test components. All ports of a static test component remain started during the whole lifetime of a static test configuration.
- In contrast to component timers, variables and ports, the verdict and the default handling is reset. This means all activated defaults are deactivated, all local verdicts and the global verdict are set to **none**.

5.1 Static configurations

5.1.1 The special configuration type: configuration

The special configuration type **configuration** is a handle for static test configurations. The special value **null** is available to indicate an undefined configuration reference, e.g. for the initialization of variables to handle a static test configuration.

Values of type **configuration** shall be the result of configuration functions, they can be checked for equality, e.g. to check if two variables store the same value, and they can be used in **execute** statements for starting a test case on an existing static test configuration and in **kill** configuration statements to destroy an existing static test configuration.

Each successful execution of a configuration function results in a different configuration value which is only equal to itself.

Restrictions

The following restrictions apply to usages of the **configuration** type:

- The **configuration** type cannot be subtyped or constrained.
- The **configuration** type is not a data type, therefore, the **anytype** does not include the configuration type.
- Module parameters shall not be of type **configuration**.
- Signature parameters shall not be of type **configuration**.
- Templates shall not be of type **configuration**.
- Templates shall not be of a structured type that contains fields or elements of type **configuration** on any level of nesting.
- External functions are not allowed to contain parameters or return values of type **configuration**.

EXAMPLES:

```

var configuration myStaticConfig := null; // Declaration and initialization of a
// configuration variable.

myStaticConfig := aStaticConfig(); // Assigns a value to the previously declared
// configuration variable. It is assumed that
// aStaticConfig() is a configuration function.

myStaticConfig.kill // Kills the static test configuration stored in
// variable myStaticConfig.

```

5.1.2 The configuration function

A configuration function allows the start of a static test configuration.

Syntactical Structure

```

configuration ConfigurationIdentifier
"(" [ { ( FormalValuePar | FormalTemplatePar) [","] } ] ")"
runs on ComponentType

```

```
[ system ComponentType ]
StatementBlock
```

Semantic Description

A configuration function allows the start of a static test configuration. A configuration function has to be defined in the definitions part of a TTCN-3 module and shall only be invoked in the control part of a TTCN-3 module. By definition, a configuration function returns a value of type **configuration** if the start of the configuration was successful, or **null** if the start of the configuration was not successful.

The invocation of a configuration function causes the creation of the MTC and the test system interface of the static test configuration. The types of MTC and test system interface shall be referenced in a **runs on** and a **system** clause. The **system** clause is optional and can be omitted, if the test system has exactly the same ports as the MTC and these ports are mapped one to one to each other.

The behaviour in the body of a configuration function shall be executed on the newly created MTC. During the start of a test configuration only behaviour on the MTC shall be executed and only static test components, static connections and static mappings shall be created or established. Communication with the SUT or with static PTCs is not allowed.

NOTE: The configuration function only returns a reference to a test configuration and no verdict. However, communication with the SUT might have to be checked. For this purpose, initial communication, e.g. for registration or coordination purposes, could be defined in form of a test case.

A static test configuration is successfully started if the behaviour of the corresponding configuration function has been executed till its end or if a **return** statement in the corresponding configuration function is reached. In case of a successful start, a reference to the newly created configuration is returned. The usage of a **stop** or a **kill** statement allows to specify an unsuccessful start of a static test configuration. In case of an unsuccessful start, the value **null** is returned.

PREVIEW
(standards.iteh.ai)

Restrictions

- a) The rules for formal parameter lists for the configuration function shall be followed as defined in clause 5.4 of ETSI ES 201 873-4 [2].
- b) Configuration functions shall only be invoked in the module control part.
- c) For the behaviour definition in the body of the configuration function and all functions directly or indirectly from the configuration function, the following restrictions shall hold:
 - Only static test components, static connections and static mappings shall be created or established. All created test components, connections and mappings during the execution of a configuration function are static.
 - Once created or established static test components, static connections and static mappings shall not be destroyed.
 - It is not allowed to start behaviour on newly created static test components.
 - Communication, timer and port operations are not allowed.

EXAMPLES:

```
// The following configuration function can be used to start a simple static test configuration
// which only consists of one MTC.
```

```
configuration simpleStaticConfig () runs on MyMTCtype{}
```

```
// The following configuration function starts a more complex static configuration.
// Configuration information is stored in MTC component variables. Further non-static
// connections and mappings may be established by the test cases that are executed
// on this configuration.
```

```
configuration aComplexStaticConfig (in integer NoOfPTCs) runs on MyMTCtype system MySystemType {
  var integer i;

  if (NoOfPTCs < 0) {
    log ("Negative number of PTCs");
```

```

    kill; // unsuccessful termination
}
else if (NoOfPTCs > MaxNoOfPTCs) { // MaxNoOfPTCs is a constant
    log ("Number of PTCs is too high");
    kill; // unsuccessful termination
}
else {
    for (i := 1, i <= NoOfPTCs, i := i + 1) {
        PTC[i] := PtcType.create static; // creation of static PTCs,
        // Array PTC[] is a component variable
        connect (mtc:SyncPort, PTC[i]:SyncPort) static; // static connection
    }
    map(mtc:PCO, system:PCO1) static; // static mapping of MTC.
    map(PTC[1]:PCO, system:PCO2); // some static mappings of PTCs,
    map(PTC[2]:PCO, system:PCO3); // further non-static mappings may be
    // established during test runs
}
return; // successful termination
}
}

```

5.1.3 Starting a static test configuration

A static test configuration is started by calling a configuration function in the control part of a TTCN-3 module. In case of a successful start, a reference to the newly created static test configuration is returned. In case of an unsuccessful start, the special value `null` is returned.

EXAMPLES:

```

control {
    var configuration myStaticConfig := null; // Declaration and initialization of a
    // configuration variable.

    myStaticConfig := aStaticConfig(); // Assigns a value to the previously declared
    // configuration variable. It is assumed that
    // aStaticConfig() is a configuration function.

    if (myStaticConfig == null) {
        stop; // Stop test campaign due to an unsuccessful start
    }
    else {
        execute(MyTestCase(), myStaticConfig); // Successful start, continuation of test campaign
        ...
    }
}

```

5.1.4 Destruction of static test configurations

A static test configuration can be destroyed by executing a **kill** configuration operation.

Syntactical Structure

ConfigurationReference.kill

Semantic Description

The execution of a **kill** configuration operation causes the destruction of a static test configuration. The destruction is similar to stopping a test case by killing the MTC. This means, resources of all static PTCs shall be released and the PTCs shall be removed. The only difference is that no test verdict is calculated and returned. After executing the **kill** configuration operation, it is not possible to execute a test case on the killed static test configuration.

Executing the kill configuration operation with the special value `null` shall have no effect, executing a kill configuration operation with a reference to a non-existing static test configuration shall cause a runtime error.

Restrictions

- a) The **kill** configuration operation shall only be executed in the control part of a TTCN-3 module.

EXAMPLES:

```
control {
  var configuration myStaticConfig := null; // Declaration and initialization of a
                                           // configuration variable.

  myStaticConfig := aStaticConfig();      // Assigns a value to the previously declared
                                           // configuration variable. It is assumed that
                                           // aStaticConfig() is a configuration function.

  myStaticConfig.kill                     // Destruction of the previously started static
                                           // test configuration.
}
```

5.1.5 Creation of static test components

All create operations invoked directly or indirectly from configuration functions create static test components. The creation of static test components can be indicated by the additional optional keyword **static** in the **create** operation. The extension of the **create** operation in clause 21.2.1 of ETSI ES 201 873-4 [2] required for the creation of static test components is described in this clause.

Syntactical Structure

```
ComponentType "." create [ "(" (Name | "-") [ "," HostId ] ")" ] [ alive | static ]
```

Semantic Description

The **create** operation in combination with the keyword **static** shall only be used to create static test components. Static test components can only be created by executing a configuration function and by functions directly or indirectly invoked by configuration functions. The keyword **static** in a **create** operation shall not be used in combination with the keyword **alive**.

NOTE 1: During the lifetime of a static test configuration, a static component behaves like an alive component.

Static test components are created in the same manner as normal test components that are not declared as alive components. Further details on this can be found in clause 21.2.1 of ETSI ES 201 873-4 [2].

NOTE 2: Static test components can only be created directly or indirectly by a configuration function. This may be checkable at runtime and therefore the keyword **static** may not be required, but for having an explicit specification of static test configurations and for keeping the feature of static test configurations extendible, the keyword **static** has been introduced.

Restrictions

- The **create** operation in combination with the keyword **static** shall only be invoked in configuration functions and in function that may be directly or indirectly called by such a configuration function.
- The keyword **static** in a **create** operation shall not be used in combination with the keyword **alive**.

EXAMPLES:

```
// This example declares variables of type MyComponentType, which are used to store the
// references of newly created static component instances of type MyComponentType.
// An associated name is allocated to some of the created component instances.
:
var MyComponentType MyNewComponent;
var MyComponentType MyNewestComponent;
:
MyNewComponent := MyComponentType.create static;
MyNewestComponent := MyComponentType.create("Newest") static;
```

5.1.6 Establishment of static connections and static mappings

The **map** and **connect** operations called directly or indirectly from configuration functions establish static connections and static mappings. This can be indicated by the additional optional keyword **static** in **connect** and the **map** operations. The extension of the **connect** and **map** operation in clause 21.1.1 of ETSI ES 201 873-4 [2] required for the establishment of static connections and mapping is described in this clause.

Syntactical Structure

```
connect "(" ComponentRef ":" Port "," ComponentRef ":" Port ")" [ static ]

map "(" ComponentRef ":" Port "," ComponentRef ":" Port ")"
  [ param "(" [ { ActualPar [ "," ] }+ ] ")" ] [ static ]
```

Semantic Description

The **connect** and **map** the operation in combination with the keyword **static** shall only be used to establish static connections and static mappings. Static connections and static mappings can only be established by executing the creator function of a configuration type and by functions directly or indirectly invoked by the creator functions of configuration type.

Static connections and static mappings are established in the same manner as normal connections and mappings. Further details on this can be found in clause 21.1.1 of ETSI ES 201 873-4 [2].

NOTE: Static connections and mappings can only be established directly or indirectly by a creator function of a configuration type. This may be checkable at runtime and therefore the keyword **static** may not be required, but for having an explicit specification of static test configurations and for keeping the feature of static test configurations extendible, the keyword **static** has been introduced.

Restrictions

- The **connect** and **map** operation in combination with the keyword **static** shall only be used in configuration functions and in functions that may be directly or indirectly called by a configuration function.
- Static connections and static mappings shall only be established to connect ports of static test components and to map ports of a static component to the ports of the test system interface of a configuration type.

EXAMPLES:

```
// The following code fragment may be part of a creator function of a configuration type.
// It is assumed that the ports Port1, Port2, Port3 and PC01 are properly defined and declared
// in the corresponding port type and component type definitions
:
var MyComponentType MyNewPTC;
MyNewPTC := MyComponentType.create static;
:
connect(MyNewPTC:Port1, mtc:Port3) static;
map(MyNewPTC:Port2, system:PC01) static;
:
```

5.1.7 Test case definitions for static test configuration

Test cases that are executed on a static test configuration have to be defined in a special manner. Such test cases shall reference the configuration function that starts a static configuration on which the test case can be executed. The type of the MTC and the type of the test system interface are referenced in the configuration function and shall therefore not be specified in the test case header. The extension of the test case definition in clause 16.3 of ETSI ES 201 873-4 [2] required for the execution of a test case on a static test configuration is described in this clause.

Syntactical Structure

```
testcase TestcaseIdentifier
  "(" [ { ( FormalValuePar | FormalTemplatePar ) [ "," ] } ] ")"
  ( runs on ComponentType [ system ComponentType ] | execute on ConfigurationType )
  StatementBlock
```

Semantic Description

A test case definition that includes an **execute on** clause will be executed on previously created static test configuration of the given configuration type. The type of the MTC and the type of the test system interface is defined in the referenced configuration type. A test case definition that includes an **execute on** clause shall not have a **runs on** or a **system** clause.

Apart from the execute on clause, the definition of test cases to be executed on a static test configuration follows the same rules as described in clause 16.3 of ETSI ES 201 873-4 [2].

Restrictions

- a) A test case definition that includes an **execute on** clause shall not have a **runs on** or a **system** clause.

EXAMPLES:

```

configuration aConfiguration () runs on MyMTCtype system MySystemType {
    PeerComponent := MyPTCType.create static; // creation of a static PTC
                                           // PeerComponent is a component variable

    connect(mtc:syncPort, PeerComponent:syncPort); // static connection

    map (mtc:PC01, system:PC01) // static mapping of MTC
    map (PeerComponent:PC02, system:PC02); // static mapping of Peer Component

    return // successful start of test configuration
}

testcase MyTestCase () execute on aConfiguration {
    default := activate(UnexpectedReceptions()); // activate a default

    PeerComponent.start (PTCbehaviour()); // starting PTC behaviour
    SyncPort.send (Ready); // synchronization with PTC
    SyncPort.receive(Ready); // PTC ready

    PC01.send (stimulus); // test starts
    ... // test behaviour
}

```

iTech STANDARD
PREVIEW
(standards.iteh.ai)
ETSI ES 202 781 V1.9.1 (2022-02)
<https://standards.iteh.ai/catalog/standards/sist/ca1d2984-7755-4fce-9a6a-bb72fbd0a03a/etsi-es-202-781-v1-9-1-2022-02>

5.1.8 Executing test cases on static test configurations

This clause only describes the syntax extensions of the **execute** statement to allow the execution of test cases with an **execute on** clause on static test configurations and the semantics for executing such test cases. The semantics of the **execute** statement for test cases without **execute on** clause remains unchanged.

Syntactical Structure

```

execute "(" TestcaseIdentifier "(" [ { TemplateInstance [ "," ] } ] ")"
        [ "," (TimerValue | "-")
        [ "," (HostId | "-")
        [ "," ConfigurationRef ] ] ] ")"

```

Semantic Description

A test case definition that includes an **execute on** clause shall be executed on previously started static test configuration of a given configuration function. The reference of the previously started static test configuration shall be referenced in the **execute** statement.

Trying to execute a test case on a non-existing or unfitting static test configuration shall cause a run time error. Unfitting test configuration means that the referenced static test configuration has not been created by the configuration function referenced in the test case header.

If the execution of a test case on a static test configuration causes an **error** verdict, all following usages of this static test configuration in **execute** statements shall cause a runtime error.