SLOVENSKI STANDARD
SIST ES 202 781 V1.1.1:2013

01-marec-2013

Metode za preskušanje in specificiranje (MTS) - 3. različica preskušanja in zapisa krmilnih preskusov - Razširitev nabora jezikov TTCN-3: Podpora konfiguriranju in uvajanju

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

Ta slovenski standard je istoveten z: **ES 202 781 Version 1.1.1**

ICS:

35.060	Jeziki, ki se uporabljajo v informacijski tehniki in tehnologiji	Languages used in information technology

SIST ES 202 781 V1.1.1:2013 **en**

ETSI ES 202 781 V1.1.1 (2010-08)

ETSI Standard

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

Reference
DES/MTS-00112 T3EXT_CONFDEP

Keywords
conformance, testing, TTCN

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

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

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
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document 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 (http://webapp.etsi.org/IPR/home.asp).

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.

Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

The present document relates to the multi-part standard covering the Testing and Test Control Notation version 3, as identified below:

ES 201 873-1 [1]: "TTCN-3 Core Language";

ES 201 873-2 [i.1]: "TTCN-3 Tabular presentation Format (TFT)";

ES 201 873-3 [i.2]: "TTCN-3 Graphical presentation Format (GFT)";

ES 201 873-4 [2]: "TTCN-3 Operational Semantics";

ES 201 873-5 [3]: "TTCN-3 Runtime Interface (TRI)";

ES 201 873-6 [4]: "TTCN-3 Control Interface (TCI)";

ES 201 873-7 [i.3]: "Using ASN.1 with TTCN-3";

ES 201 873-8 [i.4]: "The IDL to TTCN-3 Mapping";

ES 201 873-9 [i.5]: "Using XML schema with TTCN-3";

ES 201 873-10 [i.6]: "TTCN-3 Documentation Comment Specification".

1 Scope

The present document defines the Configuration and Deployment Supportpackage 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 CORBA based platforms, 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

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 reference document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://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.

2.1 Normative references

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

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 201 873-2: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular presentation Format (TFT)".

[i.2] ETSI ES 201 873-3: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT)".

[i.3] 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.4] 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.5] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML with TTCN-3".

[i.6] 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 Definitions and abbreviations

3.1 Definitions

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

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in ES 201 873-1 [1], ES 201 873-4 [2], ES 201 873-5 [3], 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 and in ES 201 873-1 [1] and ES 201 873-4 [2].

The package presented in the present document is compatible to:

- ES 201 873-1 [1] version 4.2.1;
- ES 201 873-2 [i.1] version 3.2.1;
- ES 201 873-3 [i.2] version 3.2.1;
- ES 201 873-4 [2] version 4.2.1;
- ES 201 873-5 [3] version 4.2.1;
- ES 201 873-6 [4] version 4.2.1;
- ES 201 873-7 [i.3] version 4.2.1;
- ES 201 873-8 [i.4] version 4.2.1;

- ES 201 873-9 [i.5] version 4.2.1;
- ES 201 873-10 [i.6] version 4.2.1.

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.

5 Package Concepts for the Core Language

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 reininitialized 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 deactiviated, all local verdicts and the global verdict are set to **none**.

5.1 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.

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.
```

5.2 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, intial 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.

Restrictions

a) The rules for formal parameter lists for the configuration function shall be followed as defined in clause 5.4 of 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 the following restrictions shall hold:

- Only static test components, static connections and static mappings shall be created or established.
- Once created or established static test components, static connections and static mappings shall not be destroyed.
- It is not allowed to create and establish non-static test components, connections and mappings.
- 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
    }
```


5.3 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
        ...
    }
}
```

5.4 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.
```

5.5 Creation of static test components

The creation of static test components shall be indicated by the additional keyword **static** in the **create** operation. The extension of the **create** operation in clause 21.2.1 of ES 201 873-4 [2] required for the creation of static test components is described in the following sections.

Syntactical Structure

ComponentType "." **create** ["(" *Expression* ")"] [**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 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

a) 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.

b) 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;
```


5.6 Establishment of static connections and static mappings

The establishment of static connections and static mappings shall be indicated by the additional keyword **static** in **connect** and the **map** operations. The extension of the **connect** and **map** operation in clause 21.1.1 of ES 201 873-4 [2] required for the establishment of static connections and mapping is described in the following sections.

Syntactical Structure

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

a) 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.

b) 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 PCO1 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:PCO1) static;
 :
```

5.7 Test case definitions for static test configuration

Test cases that are executed on a static test configuration have to 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 ES 201 873-4 [2] required for the execution of a test case on a static test configuration is described in the following sections.

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 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:PCO1, system:PCO1)             // static mapping ot MTC
    map (PeerComponent:PCO2, system:PCO2);  // static mapping of Peer Component

    return              // successful start of test configuration
}

testcase MyTestCase () execute on aConfiguration {

    default := activate(UnexpectedReceptions()); // activate a default
```