ETSI ES 202 781 V1.6.1 (2018-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 Language Extensions:

Configuration and Deployment Support

**ETSI Standard**

Reference

RES/MTS-202781ConfDepled161

Keywords

protocol, 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 only prevailing document is the print of the Portable Document Format (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 <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>

***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 2018.

All rights reserved.

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

### 5.2.4 Translation state

In addition to port state dimensions defined ETSI ES 201 873-1 [1], all ports working in translation mode have an additional port state dimension called translation state. The translation state always contains the result of the last executed translation function performed by the port.

There are five possible translation states:

* **unset** is the default state before invoking a translation error. If a translation function ends with this state, an error is generated;
* **not translated** means that the translation function has not been successful;
* **fragmented** indicates the translation function did not finish translation, because the input data did not contain a complete message (i.e. more fragments are needed to finish translation);
* **translated** means that the translation function successfully performed translation and there are no non‑translated data left;
* **partially translated** is used when the translation function successfully performed translation, but there are additional data which has not been translated yet (i.e. the input data contained more than one message).
* **discarded** is used when the translation function finished successfully, by discarding the message.

Translation state is set implicitly to *unset* whenever a translation function is called to translate a sent or received message. The translation state can be changed by a **setstate** operation.

***Syntactical Structure***

**port.setstate**"("SingleExpression { "," ( FreeText | TemplateInstance ) } ")"

***Semantic Description***

The **setstate** operation can be used only inside a function that is called during a translation procedure to translate a sent or received a message. It changes the translation state of the related port.

The optional parameters allow to provide information that explains the reasons for setting a port translation state. This information is composed to a string and might be used for logging purposes.

***Restrictions***

1. The value passed to the **setstate** operation in the first parameter shall be of the **integer** type and shall have one of the following values:

* 0 (meaning *translated*)
* 1 (meaning *not translated*)
* 2 (meaning *fragmented*)
* 3 (meaning *partially translated*)
* 4 (meaning *discarded*)

NOTE 1:Numeric parameter values 0, 1 and 2 are the same as results of the predefined **decvalue** function.

NOTE 2: Clause B.2.1 of the present document includes the type definition translation state and the constant definitions TRANSLATED, NOT\_TRANSLATED, FRAGMENTED, PARTIALLY\_TRANSLATED, DISCARDED.

1. Calling the **setstate** operation with an **integer** not listed in a) in the first parameter shall lead to an error.
2. Calling the **setstate** operation outside of a translation function or in a translation function translating an address shall cause a runtime error.
3. For *FreeText* and *TemplateInstance*, the same rules and restrictions apply as for the parameters of the log statement. See clause 19.11 of ETSI ES 201 873-1 [1] for more details.

NOTE 3:The *unset* state cannot be set by the setstate operation, it is reserved for TE internal use only.

### 5.2.5 Sending

When a message is to be sent over a port, working in translation mode, the following shall apply:

* If no *OutFunction* is specified for the given *InnerOutType,* it is simply sent over the port transparently.
* If an *OutFunction* is specified for the *InnerOutType*, the translation procedure first sets the translation state to *Unset*. Then the *OutFunction* is automatically invoked to translate the *InnerOutType* to the *OuterOutType.* When the function execution is finished, then depending on the current translation state one of the following actions is taken:
* The *unset* state shall cause an error (i.e. if there is no **setstate** operation is invoked in the translation function).
* If the state is *not translated*, the translation procedure tries to translate the message using the next *OutFunction* specified for the given *InnerOutType*. *OutFunction*-s are tried according to their textual order in the port type definition. If there is no such a function, an error is generated.
* If the state is *fragmented*, the translation procedure ends but no data is sent to the connected or mapped port (the port will wait for the next fragment to complete translation). The **to** clause of the following send operation shall be the same as the **to** clause of the current send operation or missing if the current send operation does not contain any to clause.
* If the state is *translated*, the translation procedure sends the translated message (retrieved from the out parameter of the *OutFunction*) to the port it is mapped or connected to.
* If the state is *partially translated*, the sent message of the*InnerOutType* contains several messages (or message fragments) of the*OuterOutType.* In this case, the translation procedure sends the translated message to the mapped or connected port. The translation function is then called again, with the same **in** parameter value, to enable sending of the remaining messages.
* If the state is *discarded*, the translation procedure ends, with no data sent to the connected or mapped port (the message was intentionally discarded).

NOTE: In the *fragmented* case the non-translated part of *InnerOutType* has to be explicitly assigned to port variables.

### 5.2.6 Receiving

Unlike a port working in standard mode, ports working in translation mode maintain two different queues. The outer queue is used to keep not translated messages that are either enqueued or sent to the port working in translation mode. The inner message queue contains already translated messages. Receiving operations access this inner queue. In case of successful receiving (see clause 22.2.2 of ETSI ES 201 873-1 [1]), the successfully received message is removed from the inner queue. Messages stored in the outer queue can be removed from it only by the translation procedure as described below.

The TTCN‑3 Executable (TE, see ETSI ES 201 873-6 [4]) shall control the translation process and the normal decoding algorithm (see note 1) in co-operation, as specified below. But yet, the normal decoding algorithm itself is not changed.

**decode (TRI message, decoding hypothesis: B**)

TE

**Port in translation mode**

**System**

**adaptor**

TRI message

**p.receive(A:?)**

Co**dec**

**decoded value**

**Outer queue**

**Inner** **queue**

***InFunction***  
 (**in B**, **out A**)

Figure 2: Illustration of the interworking of decoding and translation procedure during receiving

NOTE 1: In this clause the "normal decoding algorithm" refers to the process that the TE invokes decoding the received bitstring as specified in clauses 7.3.2 and C.5.4 of ETSI ES 201 873-6 [4].

The translation procedure for receiving operations is invoked by the snapshot mechanism. This procedure iterates through all **in** clauses (*InnerInType* -s) defined in the port type definition. The **in** clauses are iterated according to their textual order. During this iteration, the following shall apply:

* If no *InFunction* is specified for the given *InnerInType*, the translation procedure checks, if the top item of the outer queue is of *InnerInType* (i.e. invokes the normal decoding algorithm, and the check is successful if the decoding is successful). If the result of the check is positive, the message is moved from the outer queue into the inner queue (i.e. the port will relay the message from the outer port to the inner port transparently) and iteration ends.
* Otherwise (if the *InFunction* is present for the *InnerInType*), then the translation procedure checks if the top item of the outer queue is of the *OuterInType,* by invoking the normal decoding algorithm, as described above. If the check is successful, the translation procedure automatically executes the *InFunction*: first sets the translation state to *Unset* and passes the message of the *OuterInType* to it, in the first parameter. When the function execution is finished, the translation procedure checks the translation state of the port:
* The *unset* state shall cause an error (i.e. if there is no **setstate** operation is invoked in the translation function).
* If the state is *not* *translated*, the iteration shall continue with the next *InFunction* for the same *OuterInType*. If there is no more such *InFunction*, the translation procedure shall continue with the next *OuterInType*. If there is no more *OuterInType* -s for the given *InnerInType*, the iteration process shall continue with the next *InnerInType*. The order is determined by the textual order in the port type definition.
* If the state is *fragmented*, the top item of the outer queue is removed and the iteration shall be restarted to process the next message in the outer queue. The next message shall have the same address as the current one (including a missing address). If there is no such message, the iteration shall continue with the next *InnerInType*.
* If the state is *translated*, the top item of the outer queue is removed and the translated message (retrieved from the out parameter of the *InFunction*) is inserted into the inner queue. This ends the whole iteration.
* If the state is *partially translated*, the received message of the *OuterInType* contains several messages (or message fragments) of the *InnerInType.* In this case, the translated message (retrieved from the out parameter of the *InFunction*) is inserted into the inner queue. Unlike in the *translated* case, the top message is not removed from the outer queue. Instead, it is kept in its decoded form in the queue to enable translation of the remaining messages embedded in the outer message in subsequent receive calls.
* If the state is *discarded*, the top item of the outer queue is removed. No new message is inserted into the inner queue. The iteration shall be restarted to process the next message in the outer queue.

NOTE 2: In the *fragmented* case the non-translated part of *OuterInType*has to be explicitly assigned to port variables.

* If the iteration has processed all **in** clauses without any success (no transparently relayed message was successfully moved from the outer to inner queue and all *InFunction* calls ended with the *not translated* state), the iteration process returns.
* In case the iteration produces a successful result, the translation procedure might restart the iteration in order to translate the remaining messages in the outer queue (if there are any), or it might for performance consideration postpone this translation to the moment when the next snapshot is taken. For the same performance reasons, the snapshot mechanism is not required to start the translation procedure in case the inner queue already contains some messages.

### 5.2.7 Address

When an address type associated with a mapped port working in the translation mode contains a **to** or **from** clause and one of the *OuterAddrType*-s is the same as the address type of the mapped TSI port, the translation procedure is applied to all addresses used by sending or receiving calls of the port.

In case of sending a message, the translation procedure automatically invokes the *AddrOutFunction* passing the address value defined in the **to** clause to it, in its first parameter. In case of receiving a message, the translation procedure automatically invokes the *AddrInFunction* passing the received address value to it, in its first parameter. When the function execution is over, the translation procedure retrieves the translated address from the **out** parameter of the translation function and the control is returned to the calling sending or receiving procedure to finish the operation using the translated address value.

NOTE:Unlike translation functions used for translating sent or received messages, the translation functions for addresses do not use translation states.

EXAMPLE:

**type port** TransportPort

{

...

**address** TransportAddress;

}

**type port** DataPort **map to** TransportPort

{

...

**address** DataAddress **to** TransportAddress **with** toTransportAddress()

**from** TransportAddress **with** fromTransportAddress;

}

**function** toTransportAddress(DataAddress p\_addr, **out** TransportAddress p\_translated) { ...}

**function** fromTransportAddress(TransportAddress p\_addr, **out** DataAddress p\_translated) { ... }

### 5.2.8 Clear, start, stop and halt operation

The **clear** and **start** operations clean messages both from inner and outer message queues. In addition to that, all port variables are reset in the following way: if a variable declaration contains an assignment, the assignment operation will be performed as a part of the clear or start operation restoring the initial value of the variable. Otherwise (if the variable declaration does not contain an assignment part), the value of the variable will be uninitialized after the clear or start operation.

The **halt** operation affects the outer queue only. The translation procedure can still insert translated messages into the inner queue of a halted port, provided that there are available messages in the outer queue.

Since the **stop** port operation requires all communication operations to cease before the port is stopped, all unfinished translation operations shall be completely performed before the working of the port is suspended.

# B.2 Useful TTCN‑3 types

## B.2.1 Status values for port states

Type and constants defined in this clause support the secure usage of the **setstate** port operation defined in clause 5.10.4 of ETSI ES 201 873‑1 [1].

The type definition for this type is:

**type integer** translationState(0..4);

Useful constant definitions for working with object states are:

**const** translationState TRANSLATED := 0;

**const** translationState NOT\_TRANSLATED := 1;

**const** translationState FRAGMENTED := 2;

**const** translationState PARTIALLY\_TRANSLATED :=3;

**const** translationState DISCARDED := 4;

# 