ETSI ES 203 022 V1.2.1 (2018-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 extension: Advanced Matching

**ETSI Standard**

Reference

RES/MTS-203022ed121

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 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.

# 6 TCI Extensions for the Package

## 6.5 Extensions to clause 8 of ETSI ES 201 873-6, JavaTM language mapping

**Clause 8.3.2.17 TciMatchingTypeType**

This clause is to be extended.

// TCI IDL TciMatchingTypeType

package org.etsi.ttcn.tci;

public interface TciMatchingType {

public final static int TEMPLATE\_LIST = 0 ;

public final static int COMPLEMENTED\_LIST = 1 ;

public final static int ANY\_VALUE = 2 ;

public final static int ANY\_VALUE\_OR\_NONE = 3 ;

public final static int VALUE\_RANGE = 4 ;

public final static int SUBSET = 5 ;

public final static int SUPERSET = 6 ;

public final static int ANY\_ELEMENT = 7 ;

public final static int ANY\_ELEMENTS\_OR\_NONE = 8 ;

public final static int PATTERN = 9 ;

public final static int MATCH\_DECODED\_CONTENT = 10 ;

public final static int OMIT\_TEMPLATE = 11 ;

public final static int DYNAMIC\_MATCHING = 12 ;

public final static int CONJUNCTION = 13 ;

public final static int IMPLICATION = 14 ;

public final static int EXCLUSION = 15 ;

public final static int DISJUNCTION = 16 ;

public final static int REPETITION = 17 ;

}

**Clause 8.3.4.1 Value**

This clause is to be extended.

**Value** is mapped to the following interface:

// TCI IDL Value

package org.etsi.ttcn.tci;

public interface Value {

public Type getType();

public boolean notPresent();

public String getValueEncoding();

public String getValueEncodingVariant();

public String[] getEncodeAttributes();

public String[] getVariantAttributes(String encoding);

public boolean isMatchingSymbol();

public String valueToString ();

public boolean isLazy();

public boolean isFuzzy();

public boolean isEvaluated();

public LengthRestriction getLengthRestriction ();

public LengthRestriction newLengthRestriction ();

public void setLengthRestriction (LengthRestriction restriction);

public boolean isIfPresentEnabled ();

public void setIfPresentEnabled (boolean enabled);

public RangeBoundary getLowerTypeBoundary();

public RangeBoundary getUpperTypeBoundary();

public LengthRestriction getTypeLengthRestriction();

public MatchingMechanism getTypeMatchingMechanism();

public boolean isOptional();

public Mutation getMutation();

}

**Methods:**

* getMutation Returns a mutation annotation if defined for the value.

**Clause 8.3.5.6 DynamicMatch**

This clause is to be added.

**DynamicMatch** is mapped to the following interface:

// TCI IDL DynamicMatch

package org.etsi.ttcn.tci;

public interface DynamicMatch {

public Boolean isFunctionBased ();

public TciBehaviourId getMatchingFunction ();

public void setMatchingFunction (TciBehaviourId functionId);

}

**Methods:**

* isFunctionBased Returns true if the mechanism uses the short-hand notation **@dynamic** *FunctionRef* and false otherwise.
* getMatchingFunction Returns the qualified name of the associated function.
* setMatchingFunction Sets the function associated with the matching mechanism.

**Clause 8.3.5.7 TwoStepMatch**

This clause is to be added.

**TwoStepMatch** is mapped to the following interface:

// TCI IDL TwoStepMatch

package org.etsi.ttcn.tci;

public interface TwoStepMatch {

public Value getPrimaryTemplate ();

public void setPrimaryTemplate (Value primaryTemplate);

public Value getSecondaryTemplate ();

public void setSecondaryTemplate (Value secondaryTemplate);

}

**Methods:**

* getPrimaryTemplate Returns the primary template.
* setPrimaryTemplate Sets the primary template.
* getSecondaryTemplate Returns the secondary template.
* setSecondaryTemplate Sets the secondary template.

**Clause 8.3.5.8 Repetition**

This clause is to be added.

**Repetition** is mapped to the following interface:

// TCI IDL Repetition

package org.etsi.ttcn.tci;

public interface Repetition {

public Value getRepeatedTemplate ();

public void setRepeatedTemplate (Value primaryTemplate);

public LengthRestriction getRepetitionCount ();

public void setRepetitionCount (LengthRestriction repetitionCount);

}

**Methods:**

* getRepeatedTemplate Returns the repeated template.
* setRepeatedTemplate Sets the repeated template.
* getRepetitionCount Returns the repetition count.
* setRepetitionCount Sets the repetition count.

**Clause 8.3.2.21 Mutation**

This clause is to be added.

**Mutation** is mapped to the following interface:

// TCI IDL Mutation

package org.etsi.ttcn.tci;

public interface Mutation {

public boolean isMessageNeeded ();

public TriMessage mutate (TriMessage subMessage);

}

**Methods:**

* isMessageNeeded Returns true if encoded sub-message is required for mutation, false otherwise.
* mutate Returns mutated version of the sub-message.

## 6.6 Extensions to clause 9 of ETSI ES 201 873-6, ANSI C language mapping

**Clause 9.2 Value interfaces**

Table 5 is to be extended.

| TCI IDL Interface | ANSI C representation | Notes and comments |
| --- | --- | --- |
| : |  |  |
| Value | | |
| Mutation getMutation () | Boolean tciIsMutationDefined (Value inst) |  |
| TBoolean  Mutation.isMessageNeeded () | Boolean tciIsMessageNeededForMutation (Value inst) |  |
| TriMessage Mutation.mutate (TriMessage subMessage) | TriMessage tciMutate (Value inst, TriMessage \* subMessage) |  |
| DynamicMatch | | |
| TBoolean isFunctionBased () | Boolean tciIsMatchFunctionBased (Value inst) |  |
| QualifiedName getMatchingFunction() | QualifiedName \* tciGetMatchingFunction (Value inst) |  |
| Void setMatchingFunction (QualifiedName functionId) | void tciSetMatchingFunction(Value inst, QualifiedName functionId) |  |
| **TwoStepMatch** | | |
| Value getPrimaryTemplate() | Value getPrimaryTamplate(Value inst) |  |
| void setPrimaryTemplate(Value template) | void setPrimaryTemplate(Value inst, Value template) |  |
| Value getSecondaryTemplate() | Value getSecondaryTamplate(Value inst) |  |
| void setSecondaryTemplate(Value template) | void setSecondaryTemplate(Value inst, Value template) |  |
| Repetition | | |
| Value getRepeatedTemplate() | Value getRepeatedTemplate(Value inst) |  |
| void setRepeatedTemplate(Value template) | void setRepeatedTemplate(Value inst, Value template) |  |
| LengthRestriction getRepeatedTemplate() | TciLengthRestriction getRepetitionCount(Value inst) |  |
| Void setRepetitionCount (LengthRestriction repetitionCount) | void setRepetitionCount(Value inst, TciLengthRestriction repetitionCount) |  |

## 6.7 Extensions to clause 10 of ETSI ES 201 873-6, C++ language mapping

**Clause 10.5.2.16 TciMatchingTypeType**

This clause is to be extended.

typedef enum

{

TCI\_TEMPLATE\_LIST = 0,

TCI\_COMPLEMENTED\_LIST = 1,

TCI\_ANY\_VALUE = 2,

TCI\_ANY\_VALUE\_OR\_NONE = 3,

TCI\_VALUE\_RANGE = 4,

TCI\_SUBSET = 5,

TCI\_SUPERSET = 6,

TCI\_ANY\_ELEMENT = 7,

TCI\_ANY\_ELEMENTS\_OR\_NONE = 8,

TCI\_PATTERN = 9,

TCI\_MATCH\_DECODED\_CONTENT = 10,

TCI\_OMIT\_TEMPLATE = 11,

TCI\_DYNAMIC\_MATCHING = 12,

TCI\_CONJUNCTION = 13,

TCI\_IMPLICATION = 14,

TCI\_EXCLUSTION = 15,

TCI\_DISJUNCTION = 16,

TCI\_REPETITION = 17

} TciMatchingType;

**Clause 10.5.2.20 Mutation**

This clause is to be added.

Specifies a mutation annotation. It is mapped to the following pure virtual class:

class Mutation {

public:

virtual ~Mutation ();

virtual Tboolean isMessageNeeded () const =0;

virtual TriMessage mutate (const TriMessage \* p\_message) const =0;

virtual Tboolean operator== (const RangeBoundary &p\_boundary) const =0;

virtual Mutation \* clone () const =0;

virtual Tboolean operator< (const RangeBoundary &p\_boundary) const =0;

}

**Methods:**

~Mutation

Destructor

isMessageNeeded

Returns true if encoded sub-message is required for mutation, false otherwise

mutate

Returns mutated version of the sub-message

operator==

Returns true if both objects are equal

clone ()

Returns a copy of the Mutation

operator<

Operator < overload

**Clause 10.5.3.2 Value**

This clause is to be extended.

A value of TciValue represents TTCN-3 values for a given type. It is mapped to the following pure virtual class:

class TciValue {

public:

virtual ~TciValue ();

virtual const TciType & getType () const =0;

virtual const Tstring & getValueEncoding () const =0;

virtual const Tstring & getValueEncodingVariant () const =0;

virtual const std::vector<Tstring\*> & getEncodeAttributes () const =0;

virtual const std::vector<Tstring\*> & getVariantAttributes (const Tstring \* encoding) const =0;

virtual Tboolean notPresent () const =0;

virtual Tboolean isMatchingSymbol () const =0;

virtual const Tstring & valueToString () const =0;

virtual Tboolean isLazy () const =0;

virtual Tboolean isFuzzy () const =0;

virtual Tboolean isEvaluated () const =0;

virtual LengthRestriction \* getLengthRestriction () const = 0;

virtual LengthRestriction \* newLengthRestriction () const = 0;

virtual void setLengthRestriction (const LengthRestriction \* p\_restriction) =0;

virtual Tboolean isIfPresentEnabled () const =0;

virtual void setIfPresentEnabled (Tboolean p\_enabled) =0;

virtual Tboolean isOptional () const =0;

virtual RangeBoundary \* getLowerTypeBoundary() const = 0;

virtual RangeBoundary \* getUpperTypeBoundary() const = 0;

virtual LengthRestriction \* getTypeLengthRestriction() const = 0;

virtual MatchingMechanism \* getTypeMatchingMechanism() const = 0;

virtual Mutation \* getMutation() const = 0;

virtual Tboolean operator== (const TciValue &p\_val) const =0;

virtual TciValue \* clone () const =0;

virtual Tboolean operator< (const TciValue &p\_val) const =0;

}

**Methods:**

* getMutation Returns a mutation annotation if defined for the value.

**Clause 10.5.3.23 DynamicMatch**

This clause is to be added.

TTCN-3 dynamic matching mechanism support. It is mapped to the following pure virtual class:

class DynamicMatch : public virtual MatchingMechanism {

public:

virtual ~DynamicMatch ();

virtual Tboolean isFunctionBased () const =0;

virtual const TciBehaviourId \* getMatchingFunction () const =0;

virtual void setMatchingFunction (const TciBehaviourId & functionId) =0;

virtual Tboolean operator== (const DynamicMatch &p\_dynamicMatch) const =0;

virtual DynamicMatch \* clone () const =0;

virtual Tboolean operator< (const DynamicMatch &p\_content) const =0;

}

**Methods:**

~DynamicMatch

Destructor

isFunctionBased

Returns true if the mechanism uses the short-hand notation **@dynamic** *FunctionRef* and false otherwise

getMatchingFunction

Returns the qualified name of the associated function

setMatchingFunction

Sets the function associated with the matching mechanism

operator==

Returns true if both objects are equal

clone

Return a copy of the matching mechanism

operator<

Operator < overload

**Clause 10.5.3.24 TwoStepMatch**

This clause is to be added.

TTCN-3 implication and exclusion matching mechanism support. It is mapped to the following pure virtual class:

class TwoStepMatch : public virtual MatchingMechanism {

public:

virtual ~TwoStepMatch ();

virtual Value & getPrimaryTemplate () const =0;

virtual void setPrimaryTemplate (const Value & template) =0;

virtual Value & getSecondaryTemplate () const =0;

virtual void setSecondaryTemplate (const Value & template) =0;

virtual Tboolean operator== (const TwoStepMatch &p\_twoStepMatch) const =0;

virtual TwoStepMatch \* clone () const =0;

virtual Tboolean operator< (const TwoStepMatch &p\_content) const =0;

}

**Methods:**

~TwoStepMatch

Destructor

getPrimaryTemplate

Returns the primary template

setPrimaryTemplate

Sets the primary template

getSecondaryTemplate

Returns the secondary template

setSecondaryTemplate

Sets the secondary template

operator==

Returns true if both objects are equal

clone

Return a copy of the matching mechanism

operator<

Operator < overload

**Clause 10.5.3.24 Repetition**

This clause is to be added.

TTCN-3 repetition matching mechanism support. It is mapped to the following pure virtual class:

class Repetition : public virtual MatchingMechanism {

public:

virtual ~Repetition ();

virtual Value & getRepeatedemplate () const =0;

virtual void setRepeatedTemplate (const Value & template) =0;

virtual LengthRestriction & getRepetitionCount () const =0;

virtual void setRepetitionCount (const LengthRestriction & repetitionCount) =0;

virtual Tboolean operator== (const Repetition &p\_repetition) const =0;

virtual Repetition \* clone () const =0;

virtual Tboolean operator< (const Repetition &p\_content) const =0;

}

**Methods:**

~Repetition

Destructor

getRepeatedTemplate

Returns the repeated template

setRepeatedTemplate

Sets the repeated template

getRepetitionCount

Returns repetition count

setRepetitionCount

Sets repetition count

operator==

Returns true if both objects are equal

clone

Return a copy of the matching mechanism

operator<

Operator < overload

## 6.8 Extensions to clause 12 of ETSI ES 201 873-6, C# language mapping

**Clause 12.4.2.16 TciMatchingTypeType**

This clause is to be extended.

public enum TciMatchingType {   
 TemplateList = 0,  
 ComplementedList = 1,  
 AnyValue = 2,  
 AnyValueOrNone = 3,  
 ValueRange = 4,  
 Subset = 5,  
 Superset = 6,  
 AnyElement = 7,  
 AnyElementsOrNone = 8,  
 Pattern = 9,  
 MatchDecodedContent = 10,  
 OmitTemplate = 11,  
 DynamicMatch = 12,   
 Conjunction = 13,   
 Implication = 14,   
 Exclusion = 15,   
 Disjunction = 16,  
 Repetition = 17  
}

**Clause 12.4.2.20 Mutation**

This clause is to be added.

**Mutation** is mapped to the following interface:

public interface ITciMutation

{

bool IsMessageNeeded { get; }

ITriMessage Mutate (ITriMessage subMessage);

}

**Methods:**

* IsMessageNeeded Returns true if encoded sub-message is required for mutation, false otherwise.
* Mutate Returns mutated version of the sub-message.

**Clause 12.4.4.1 Mutation**

This clause is to be extended.

The IDL type **Value** is mapped to the following interface:

public interface ITciValue {  
 ITciType Type { get; }  
 bool NotPresent { get; }  
 string ValueEncoding { get; }  
 string ValueEncodingVariant { get; }  
 string[] EncodeAttributes { get; }  
 string[] GetVariantAttributes(string encoding);  
 bool IsMatchingSymbol { get; }  
 string ValueToString();  
 bool IsLazy { get; }  
 bool IsFuzzy { get; }  
 bool IsEvaluated { get; }

ITciLengthRestriction LengthRestriction { get; set; }

ITciLengthRestriction NewLengthRestriction ();

bool IsIfPresentEnabled { get; set; }

ITciRangeBoundary LowerTypeBoundary { get; }

ITciRangeBoundary UpperTypeBoundary { get; }

ITciLengthRestriction TypeLengthRestriction { get; }

ITciMatchingMechanism TypeMatchingMechanism { get; }  
 bool IsOptional { get; }

Mutation { get; }

}

**Members:**

* Mutation  
  Returns a mutation annotation if defined for the value.

**Clause 12.4.5.6 DynamicMatch**

This clause is to be added.

The IDL type **DynamicMatch** is mapped to the following interface:

public interface ITciDynamicMatch : IMatchingMechanism

{

bool IsFunctionBased { get; }

ITciBehaviourId MatchingFunction { get; set; }

}

**Methods:**

* IsFunctionBased Returns true if the mechanism uses short-hand notation **@dynamic** *FunctionRef* and false otherwise.
* MatchingFunction Gets or sets the function associated with the matching mechanism.

**Clause 12.4.5.7 TwoStepMatch**

This clause is to be added.

The IDL type **TwoStepMatch** is mapped to the following interface:

public interface ITciTwoStepMatch : IMatchingMechanism

{

ITciValue PrimaryTemplate { get; set; }

ITciValue SecondaryTemplate { get; set; }

}

**Methods:**

* PrimaryTemplate Gets or sets the primary template.
* SecondaryTemplate Gets or sets the secondary template.

**Clause 12.4.5.8 Repetition**

This clause is to be added.

The IDL type **Repetition** is mapped to the following interface:

public interface ITciRepetition : IMatchingMechanism

{

ITciValue RepeatedTemplate { get; set; }

ITciLengthRestriction RepetitionCount { get; set; }

}

**Methods:**

* RepeatedTemplate Gets or sets the repeated template.
* RepetitionCount Gets or sets the repetition count

## 6.9 Extensions to clause 7.2.2.2.1 of ETSI ES 201 873-6, The abstract data type Value

The definition of the abstract data type Value is to be extended by adding the following operation:

Mutation getMutation() If a mutation annotation is attached to the value, the operation returns properties of the mutation. Otherwise, the distinct value null is returned.

## 6.10 Extensions to clause 7.2.2.4 of ETSI ES 201 873-6, Data types for complex TTCN-3 properties

**Clause 7.2.2.4.4 Mutation**

This clause is to be added.

The abstract data type Mutation is used to describe properties of a mutation annotation.

The following operations are defined on the base abstract data type Mutation:

TBoolean isMessageNeeded () Returns true if an encoded sub-message is required for evaluation of the mutation. Otherwise, false is returned.

TriMessage mutate (TriMessage subMessage)  
Returns the mutated version of the value. In case an encoded sub-message is required for evaluation of the mutation, the encoded sub-message shall be passed in the parameter. If the encoded sub-message is not required for evaluation, it is allowed to set the parameter to the distinct value null. The operation is used by the encoder.