ETSI ES 201 873-6 V4.8.1 (2016-07)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 6: TTCN‑3 Control Interface (TCI)

**ETSI Standard**

### 7.2.2 Abstract TTCN‑3 data types and values

#### 7.2.2.0 Definition and scope of use

The clause 7.2.2 defines the set of abstract data types that build up the TTCN‑3 type, value and template representation. Functionality of each data type is defined by an accompanying set of operations. Operations on or using this abstract data type return either a value of this abstract type or a basic type like boolean.

All operations have been defined using the Interface Description Language (IDL). Concrete language mappings for the operations on the abstract data types are given in clauses 8, 9, 10, 11 and 12. In certain languages, the application of an operation on an abstract data type is represented by passing (either by‑value or by-reference, depending on the mapping) the concrete value as a parameter to the operation. Other languages might choose other referencing method to the concrete value, e.g. by considering the value as an object on which a method corresponding to the operation is invoked. To indicate the inability to perform a certain task or to indicate the absence of an optional parameter in the following, the distinct value null is used. It can be considered as being a reserved value indicating a special value. The language mappings will define a concrete representation of this distinct value null.

The abstract TTCN‑3 type, value and template representation consists of the following parts:

* an abstract data type Type that represents all TTCN‑3 types in a TTCN‑3 module;
* different abstract data types that represent TTCN‑3 values, i.e. TTCN‑3 values of a given TTCN‑3 type. This can be either values of TTCN‑3 predefined types or of TTCN‑3 user‑defined types;
* different abstract data types that represent matching mechanisms that can occur in TTCN-3 templates;
* other abstract data types that represent complex value properties such a permutations or length restrictions.

For accessing, evaluating, and coding the TTCN‑3 data the test system uses the abstract data type Type and the different abstract value data types. Therefore, these abstract data types define the abstraction level between the TTCN‑3 Executable (TE) and the remaining test system using the TCI interfaces.

#### 7.2.2.1 Abstract TTCN‑3 data types

According to the present document TTCN‑3 types, either predefined or user‑defined, will be represented at the TCI interfaces using the abstract data type Type.

For the abstract data type Type a set of operations is defined to:

* reference predefined and user‑defined TTCN‑3 data types; and
* create and maintain TTCN‑3 values and templates.

The following operations are defined for the abstract data type Type:

TciModuleIdType getDefiningModule() Returns the module identifier of the module in which type is defined. Returns the distinct value null if type is a TTCN‑3 base type, e.g. boolean, integer, etc.).

TString getName() Returns the name of the type as defined in the TTCN‑3 module. If the type is a nested type without explicit name, the TE has to create an additional unique identifier for this type which is consistently used in TRI/TCI.

NOTE 1: The creation of identifiers for nested types is tool dependent.

NOTE 2: The naming for a nested type without explicit name can follow the rules defined in clauses 6.2.1.1 and 6.2.3.2 of ETSI ES 201 873‑1 [1], e.g. TypeIdOrExpression.ElementId and TypeId[-], respectively.

TciTypeClassType getTypeClass() Returns the type class of the respective type. A value of TciTypeClassType can have one of the following constants: ADDRESS, ANYTYPE, ARRAY, BITSTRING, BOOLEAN, CHARSTRING, COMPONENT, ENUMERATED, FLOAT, HEXSTRING, INTEGER, OCTETSTRING, RECORD, RECORD\_OF, SET, SET\_OF, UNION, UNIVERSAL\_CHARSTRING, VERDICT, DEFAULT, PORT, TIMER.

Value newInstance() Returns a freshly created value of the given type. This initial value of the created value is undefined.

NOTE 3: Newly created instances of empty record types are considered to be initialized.

TString getTypeEncoding() Returns the type encode attribute as defined in the TTCN‑3 module. If no encode attribute is defined the distinct value null is returned. If the type has more than one encode attributes associated with it, all encode attributes are concatenated to a single string using NL (new line, char(0, 0, 0, 13) ), CR (carriage return char(0, 0, 0, 10) ) or their combination as a separator.

TString getTypeEncodingVariant() This operation returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute is defined, the distinct value null is returned. If the type has more than one variant attributes associated with it, all variant attributes are concatenated to a single string using NL (new line, char(0, 0, 0, 13) ), CR (carriage return char(0, 0, 0, 10) ) or their combination as a separator. In case the type supports multiple encodings, the function returns variants for all encodings. Each variant is prefixed with the associated encode attribute and followed by a full stop character in this case.

TStringseq getEncodingAttributes() Returns all enconde attributes of the type as defined in the TTCN‑3 module. If no encode attribute is defined the distinct value null is returned.

TStringseq getVariantAttributes(in TString encoding)
This operation all variant attributes of the type as defined in the TTCN-3 module. If no encode attribute is defined, the distinct value null is returned. The parameter is used to specify encoding the variant attributes are related to. It is required when the type has multiple encodings associated with it. If the type uses a single encoding , the parameter can be set to the special value null. The function returns the distinct value null, if the parameter specifies a non-existent encoding or if it contains null and the type contains multiple encodings.

TStringseq getTypeExtension() Returns the type extension attribute as defined in the TTCN‑3 module.

Value parseValue(in TString val) This operation creates a new value of the given type from a string provided in the parameter. The input string shall use valid TTCN‑3 syntax for values or templates of this type. The only references allowed in the input string are type references. If the input string contains an error, the distinct value null is returned. The operation is an optional part of the TCI and tool vendors are not required to support value parsing. If not supported, parseValue will always return the distinct value null.

NOTE 4: The parseValue operation can be used for defining matching symbols to enable the representation of templates in TCI.

MatchingMechanism newTemplate(TciMatchingType matchingType)
Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created (see clause 7.2.2.3.1 for more details). If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism.

RangeBoundary getLowerTypeBoundary()
Returns the lower range boundary of the type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.

RangeBoundary getUpperTypeBoundary()
Returns the lower range boundary of the type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.

LengthRestriction getTypeLengthRestriction()
Returns the length restriction of the type if it has a length restriction attached to it. Otherwise, the distinct value null is returned.

MatchingMechanism getTypeMatchingMechanism()
Returns the matching mechanism of the type if it is restricted by a matching mechanism. Otherwise, the distinct value null is returned.

#### 7.2.2.2 Abstract TTCN‑3 values

##### 7.2.2.2.0 Basic rules

According to the present document, TTCN‑3 values are represented at the TCI interfaces via numerous abstract data types.

Figure 4 presents the hierarchy between the abstract data types for TTCN‑3 values (short: abstract values).

Type

Value

MatchingMechanism

IntegerValue

FloatValue

BooleanValue

CharstringValue

UniversalCharstringValue

BitstringValue

OctetstringValue

HexstringValue

RecordOfValue

RecordValue

UnionValue

EnumeratedValue

VerdictValue

AddressValue

MatchingList

ValueRange

CharacterPattern

MatchDecodedContent

Figure 4: Hierarchy of abstract values

As shown in figure 4, all TTCN‑3 abstract values share the same base abstract data type Value. All operations defined on this common base data type are implicitly defined also for the abstract value types derived from it.

In addition, Value can be used to represent matching mechanisms, which are used instead or inside values e.g. in template parameters or for template variables. Two additional operations: isMatchingSymbol (returns true for matching symbols) and valueToString (for printing value content in the same way as the log operation; can be used for displaying value content) are defined. These operations are not mandatory - it is up to a tool vendor to support them or not.

Values using @lazy and @fuzzy modifiers are represented by the Value data type too. However, it is not possible to use the Value data type to perform evaluation of these values; evaluation can be performed by the TE only. If a @lazy or @fuzzy value has been assigned, but it doesn't contain result of the evaluation, any data access operations shall result in an error.

##### 7.2.2.2.1 The abstract data type Value

The following operations are defined on the base abstract data type Value. The concrete representations of these operations are defined in the respective language mapping sections:

Type getType() Returns the type of the specified value.

TBoolean notPresent() Returns true if the specified value is omit, false otherwise.

TString getValueEncoding() Returns the value encode attribute as defined in the TTCN-3 module, if any. If no encoding attribute is defined, the distinct value null is returned. If the value has more than one encode attributes associated with it, all encode attributes are concatenated to a single string using NL (new line, char(0, 0, 0, 13) ), CR (carriage return char(0, 0, 0, 10) ) or their combination as a separator.

TString getValueEncodingVariant() Returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute is defined the distinct value null is returned. If the value has more than one variant attributes associated with it, all variant attributes are concatenated to a single string using NL (new line, char(0, 0, 0, 13) ), CR (carriage return char(0, 0, 0, 10) ) or their combination as a separator. In case the value supports multiple encodings, the function returns variants for all encodings. Each variant will be prefixed with the associated encode attribute and followed by a full stop character in this case.

TStringseq getEncodingAttributes() Returns all enconde attributes of the value as defined in the TTCN‑3 module. If no encode attribute is defined the distinct value null is returned.

TStringseq getVariantAttributes(in TString encoding)
This operation all variant attributes of the value as defined in the TTCN-3 module. If no variant attribute is defined the distinct value null is returned. The parameter is used to specify encoding the variant attributes are related to. It is required when the value has multiple encodings associated with it. If the type uses a single encoding , the parameter can be set to the special value null. The function returns the distinct value null, if the parameter specifies a non-existent encoding or if it contains null and the value contains multiple encodings.

TBoolean isMatchingSymbol() Returns true if the instance is of the MatchingMechanism abstract data type (or any other abstract data type derived from the MatchingMechanism data type) and false in all other cases.

NOTE: This method can be used for detecting the exact abstract data type of the instance. If the method returns false, it is safe to assume that the instance is one of the abstract value data types defined in clause 7.2.2.2. If the method returns true, the instance is one of the matching mechanism defined in clause 7.2.2.3.

TString valueToString() Returns the same string as produced by the log operation (specified in clause 19.11 of ETSI ES 201 873‑1 [1]) with the specified value as its parameter.

TBoolean isFuzzy () Returns true if the specified value has the @fuzzy modifier, false otherwise.

TBoolean isLazy () Returns true if the specified value has the @lazy modifier, false otherwise.

TBoolean isEvaluated () Returns true if the value has been evaluated and its data content is available, false otherwise. In case of uninitialized values, false is always returned. The method is typically used for @lazy values, and it returns false for values that have been assigned, but not evaluated yet and true if the value contains the evaluation result. The method returns false for @fuzzy values, as the result of evaluation is never stored by the TE. For all other values, the method returns true.

LengthRestriction getLengthRestriction()
Returns a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) in case it is attached to the value or the distinct value null if no such matching attribute is present.

void setLengthRestriction(LengthRestriction restriction)
Adds a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) to the value or modifies an existing one. The distinct value null can be used as a parameter to disable an existing length restriction.

TBoolean isIfPresentEnabled() Returns true if the ifpresent indicator (specified in clause B.1.4.2 of ETSI ES 201 873‑1 [1]) is attached to the value and false otherwise.

void setIfPresentEnabled(TBoolean enabled)
Sets the whether the ifpresent indicator (specified in clause B.1.4.2 of ETSI ES 201 873‑1 [1]) is attached to the value or not.

RangeBoundary getLowerTypeBoundary()
Returns the lower range boundary of the value's type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.
This is only applicable for values with types of typeclass INTEGER and FLOAT.

RangeBoundary getUpperTypeBoundary()
Returns the lower range boundary of the value's type if it has a range-restriction attached to it. Otherwise, the distinct value null is returned.
This is only applicable for values with types of typeclass INTEGER or FLOAT.

LengthRestriction getTypeLengthRestriction()
Returns a length restriction matching attribute (specified in clause B.1.4.1 of ETSI ES 201 873‑1 [1]) in case it is attached to the value's type or the distinct value null if no such matching attribute is present.
This is only applicable for values with types of typeclass CHARSTRING, UNIVERSAL\_CHARSTRING, BITSTRING, HEXSTRING, OCTETSTRING, RECORD\_OF, SET\_OF or ARRAY.

MatchingMechanism getTypeMatchingMechanism()
Returns the matching mechanism (see clause 7.2.2.3.1) of the value's type if it is restricted by a subtype specification attribute (specified in clauses 6.1.2 and 6.2.13 of ETSI ES 201 873‑1 [1]). Otherwise, the distinct value null is returned.

When working with length restriction data using the getLengthRestriction and setLengthRestriction, methods, no assumption shall be made on how a the data are stored in a value. An internal implementation might choose to use a reference to the data or copy the data. It is safe to assume that the data are copied. Therefore, it should be assumed that subsequent modifications of the length restriction data will not be considered in the value object.

# 8 JavaTM language mapping

#### 8.3.3.1 Type

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

// TCI IDL Type

package org.etsi.ttcn.tci;

public interface Type {

 public TciModuleId getDefiningModule ();

 public String getName ();

 public int getTypeClass ();

 public Value newInstance ();

 public String getTypeEncoding ();

 public String getTypeEncodingVariant();

 public String[] getEncodeAttributes();

 public String[] getVariantAttributes(String encoding);

 public String[] getTypeExtension();

 public Value parseValue (String val);

 public MatchingMechanism newTemplate (int matchingType);

}

**Methods:**

* getDefiningModule Returns the module identifier of the module the type has been defined in. If the type represents a TTCN‑3 base type the distinct value null will be returned.
* getName Returns name of the type as defined in the TTCN‑3 module.
* getTypeClass Returns the type class of the respective type. A value of TciTypeClassType can have on of the following constants: ADDRESS, ANYTYPE, BITSTRING, BOOLEAN, CHARSTRING, COMPONENT, ENUMERATED, FLOAT, HEXSTRING, INTEGER, OCTETSTRING, RECORD, RECORD\_OF, ARRAY, SET, SET\_OF, UNION, UNIVERSAL\_CHARSTRING, VERDICT, DEFAULT, PORT, TIMER.
* newInstance Returns a freshly created value of the given type. This initial value of the created value is undefined.
* getTypeEncoding Returns the type encoding attribute as defined in the TTCN‑3 module.
* getTypeEncodingVariant This operation returns the value encoding variant attribute as defined in the TTCN-3 module, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* getEncodeAttributes Returns all encode attributes of the type.
* getVariantAttributes Returns all variant attributes of the type for the specified encoding.
* getTypeExtension Returns the type extension attribute as defined in the TTCN‑3 module.
* parseValue Parses the value provided in the parameter and in case of successful parsing returns a Value object representing the parsed value. In case of an error or if value parsing is not supported by the tool, the method returns null.
* newTemplate Returns a freshly created matching mechanism of this type. The matchingType parameter determines what kind of matching mechanism will be created and it shall be one of the following constants: TEMPLATE\_LIST, COMPLEMENTED\_LIST, ANY\_VALUE, ANY\_VALUE\_OR\_NONE, VALUE\_RANGE, SUBSET, SUPERSET, ANY\_ELEMENT, ANY\_ELEMENTS\_OR\_NONE, PATTERN, DECODED\_MATCH. If the created matching mechanism contains additional data properties, these properties are uninitialized in the created matching mechanism.

### 8.3.4 Abstract value mapping

#### 8.3.4.0 General principles

TTCN‑3 values can be retrieved from the TE and constructed using the Value interface. The value mapping interface is constructed hierarchically with Value as the basic interface. Specialized interfaces for different types of values have been defined.

#### 8.3.4.1 Value

**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 LengthRestriction getTypeLengthRestriction();

**Methods:**

* getType Returns the type of the specified value.
* notPresent Returns true if the specified value is omit, false otherwise.
* getValueEncoding This operation returns the value encoding attribute as defined in the TTCN‑3module, if any. If no encoding attribute has been defined the distinct valuenull will be returned.
* getValueEncodingVariant This operation returns the value encoding variant attribute as defined in TTCN‑3, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* getEncodeAttributes Returns all encode attributes of the value.
* getVariantAttributes Returns all variant attributes of the value for the specified encoding.
* isMatchingSymbol Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise.
* valueToString Returns the same string as produced by the log operation with the specified value as its parameter.
* isLazy Returns true if the specified value is @lazy, false otherwise.
* isFuzzy Returns true if the specified value is @fuzzy, false otherwise.
* isEvaluated Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details).
* getLengthRestriction Returns a length restriction matching attribute in case it is attached to the value or the distinct value null if no such matching attribute is present.
* newLengthRestriction Creates a new instance of the LengthRestriction interface.
* setLengthRestriction Adds a length restriction matching to the value or modifies an existing one. The distinct value null can be used as a parameter to disable an existing length restriction.
* isIfPresentEnabled Returns true if the ifpresent is attached to the value and false otherwise.
* setIfPresentEnabled Sets the whether the ifpresent indicator is attached to the value or not.
* getLowerTypeBoundary Return the lower boundary of the value's type restriction or null.
* getUpperTypeBoundary Return the upper boundary of the value's type restriction or null.
* getTypeLengthRestriction Return the value's type length restriction or null.
* getTypeMatchingMechanism Return the value's type restriction.

# 9 ANSI C language mapping

## 9.1 Introduction

This clause defines the TCI ANSI‑C [8] language mapping for the TCI data specified in clause 7.2 and for the TCI operations specified in clause 7.3.

## 9.2 Value interfaces

Table

| TCI IDL Interface | ANSI C representation | Notes and comments |
| --- | --- | --- |
| Type |
| TciModuleIdType getDefiningModule() | TciModuleIdTypetciGetDefiningModule(Type inst) |  |
| Tstring getName() | String tciGetName(Type inst) | String type reused from IDL (OMG recommendation). |
| TciTypeClassType getTypeClass() | TciTypeClassType tciGetTypeClass (Type inst) |  |
| Value newInstance() | Value tciNewInstance(Type inst) |  |
| TString getTypeEncoding() | String tciGetTypeEncoding(Type inst) |  |
| TStringseq getTypeExtension() | String\* tciGetTypeExtension(Type inst) |  |
| TString getTypeEncodingVariant() | String tciGetTypeEncodingVariant(Type inst) | Returns null pointer or a null‑pointer terminated array  |
| TString getEncodeAttributes () | String\* tciGetTypeEncodeAttributes(Type inst) | Returns null pointer or a null‑pointer terminated array |
| TString getVariantAttributes (TString encoding) | String\* tciGetTypeVariantAttributes(Type inst, String encoding) | Returns null pointer or a null‑pointer terminated array |
| Value parseValue(TString val) | Value tciParseValue(Type inst, String val) |  |
| Value |
| TString getValueEncoding() | String tciGetValueEncoding(Value inst) |  |
| TString getValueEncodingVariant() | String tciGetValueEncodingVariant(Value inst) |  |
| TString getEncodeAttributes () | String\* tciGetValueEncodeAttributes(Type inst) | Returns null pointer or a null‑pointer terminated array |
| TString getVariantAttributes (TString encoding) | String\* tciGetValueVariantAttributes(Type inst, String encoding) | Returns null pointer or a null‑pointer terminated array |
| Type getType() | Type tciGetType(Value inst) |  |
| Tboolean notPresent() | Boolean tciNotPresent(Value inst) | Boolean type reused from IDL (OMG recommendation). |
|  | void tciSetNull(Value inst) | For optional parameters of operations, see clause 9.7. |
|  | Boolean tciIsNull(Value inst) | For optional parameters of operations, see clause 9.7. Boolean type reused from IDL (OMG recommendation). |
| Tboolean isMatchingSymbol() | Boolean tciIsMatchingSymbol(Value inst) |  |
| TString valueToString() | String tciValueToString(Value inst) |  |
| Tboolean isLazy () | Boolean tciIsLazy(Value inst) |  |
| Tboolean isFuzzy () | Boolean tciIsFuzzy(Value inst) |  |
| Tboolean isEvaluated() | Boolean tciIsEvaluated(Value inst) |  |
|  | Boolean tciHasLengthRestriction(Value inst) | Returns true if length restriction is present, false otherwise. |
| LengthRestriction getLengthRestriction() | TciLengthRestriction tciGetLengthRestriction(Value inst) |  |
|  | void tciRemoveLengthRestriction(Value inst) | Removes length restriction. |
| void setLengthRestriction (LengthRestriction restriction) | void tciSetLengthRestriction(Value inst, TciLengthRestriction restriction) |  |
| TBoolean isIfPresentEnabled() | Boolean tciIsIfPresentEnabled(Value inst) |  |
| void setIfPresentEnabled(TBoolean enabled) | void tciSetIfPresentEnabled(Value inst, Boolean enabled) |  |
| LengthRestriction getTypeLengthRestriction() | TciLengthRestrictiontciGetTypeLengthRestriction(Value inst) | Get length restriction from type of value |
| RangeBoundary getLowerTypeBoundary() | TciRangeBoundarytciGetLowerTypeBoundary(Value inst) | Get lower boundary of type of value |
| RangeBoundary geUpperTypeBoundary() | TciRangeBoundarytciGetUpperTypeBoundary(Value inst) | Get upper boundary of type of value |
| MatchingMechanism getTypeMatchingMechanism() | Value tciGetTypeMatchingMechanism(Value inst) | Get the restriction of type of value |

# 10 C++ language mapping

### 10.5.3 Abstract TTCN-3 data types and values

#### 10.5.3.1 TciType

A value of TciType represents one of the TTCN-3 types in a TTCN-3 module. It is mapped to the following pure virtual class:

class TciType {

public:

 virtual ~TciType ();

 virtual const TciModuleId & getDefiningModule () const =0;

 virtual const Tstring & getName () const =0;

 virtual const TciTypeClass & getTypeClass () const =0;

 virtual const Tstring & getTypeEncoding () const =0;

 virtual const Tstring & getTypeEncodingVariant () const =0;

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

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

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

 virtual TciValue \* newInstance () const =0;

 virtual TciValue \* parseValue (const Tstring & val) const =0;

 virtual RangeBoundary \* getLowerTypeBoundary() const = 0;

 virtual RangeBoundary \* getUpperTypeBoundary() const = 0;

 virtual LengthRestriction \* getTypeLengthRestriction() const = 0;

 virtual MatchingMechanism \* getTypeMatchingMechanism() const = 0;

 virtual Tboolean operator== (const TciType &typ) const =0;

 virtual TciType \* clone () const =0;

 virtual Tboolean operator< (const TciType &typ) const =0;

}

**Methods:**

~TciType

 Destructor

getDefiningModule

 Return the defining module as defined in the TTCN-3 module

getName

 Return type name as defined in the TTCN-3 module

getTypeClass

 Return this type class

getTypeEncoding

 Return type encoding as defined in the TTCN-3 module

getTypeEncodingVariant

 Return encoding variant as defined in the TTCN-3 module

getEncodeAttributes

 Returns all encode attributes of the type as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector.

getVariantAttributes

 Returns all variant attributes of the type as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector.

getTypeExtension

 Return type extension as defined in the TTCN-3 module

newInstance

 Return a new Value instance of this type

parseValue

 Returns a new TciValue instance in case of successful parsing or null pointer in case of parsing error or if value parsing is not supported by the tool

getLowerTypeBoundary

 Return the lower boundary of the value's type restriction or null.

getUpperTypeBoundary

 Return the upper boundary of the value's type restriction or null.

getTypeLengthRestriction

 Return the value's type length restriction or null.

getTypeMatchingMechanism

 Return the value's type restriction.

operator==

 Return true if the types are equal

clone

 Return a copy of the TciType

operator<

 Operator < overload

#### 10.5.3.2 TciValue

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

 virtual TciValue \* clone () const =0;

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

}

**Methods:**

~TciValue

 Destructor

getType

 Returns the type of the specified value

getValueEncoding

 Returns the value encoding attribute as defined in the TTCN-3 module

getValueEncodingVariant

 Returns the value encoding variant attribute as defined in the TTCN-3 module

getEncodeAttributes

 Returns all encode attributes of the value as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector.

getVariantAttributes

 Returns all variant attributes of the value as defined in the TTCN-3 module. The distinct value null is mapped to an empty vector.

notPresent

 Returns true if the specified value is omit

isMatchingSymbol

 Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise

valueToString

 Returns the same string as produced by the log operation

isLazy

 Returns true if the specified value is @lazy, false otherwise

isFuzzy

 Returns true if the specified value is @fuzzy, false otherwise

isEvaluated

 Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details)

getLengthRestriction

 Returns a length restriction matching attribute or null if no restriction is present

newLengthRestriction

 Creates a new instance of the LengthRestriction class

setLengthRestriction

 Adds a length restriction matching to the value or modifies an existing one. Null pointer can be used to remove an existing length restriction

isIfPresentEnabled

 Returns true if the ifpresent matching attribute is attached to the value and false
otherwise

setIfPresentEnabled

 Sets the whether the ifpresent matching attribute is attached to the value or not

operator==

 Returns true if both objects are equal

clone

 Return a copy of the TciValue

operator<

 Operator < overload

# 12 C# mapping

### 12.4.3 Abstract type mapping

#### 12.4.3.0 Mapping rules

The TTCN-3 data types are modelled in C# using the abstract type mapping as defined in this clause. The ITciType interface defines only operations used to retrieve in TTCN-3 defined types. No TTCN-3 types can be constructed using the ITciType interface. Types are modelled using the single interface ITciType, that provides methods to identify types and to retrieve values of a given type.

#### 12.4.3.1 Type

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

public interface ITciType {
 ITciModuleId DefiningModule { get; }
 string Name { get; }
 TciTypeClass TypeClass { get; }
 ITciValue NewInstance();
 string TypeEncoding { get; }
 string TypeEncodingVariant { get; }
 string[] EncodeAttributes { get; }
 string[] GetVariantAttributes(string encoding);
 string[] TypeExtension { get; }
 ItciValue ParseValue (string val);
}

**Members:**

* DefiningModule
Returns the module identifier of the module the type has been defined in. If the type represents a TTCN-3 base type the distinct value null will be returned.
* Name
Returns name of the type as defined in the TTCN-3 module.
* TypeClass
Returns the type class of the respective type.
* NewInstance
Returns a freshly created value of the given type. This initial value of the created value is undefined.
* TypeEncoding
Returns the type encoding attribute as defined in the TTCN-3 module, if any. If no encoding attribute has been defined, the distinct value null will be returned.
* TypeEncodingVariant
This property returns the type encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined, the distinct value null will be returned.
* EncodeAttributes
Returns all encode attributes of the type as defined in the TTCN-3 module. If no encode attribute has been defined, the distinct value null is returned.
* GetVariantAttributes
Returns all variant attributes of the type as defined in the TTCN-3 module. If no encoding attribute has been defined, the distinct value null is returned.
* TypeExtension
Returns the type extension attributes as defined in the TTCN-3 module. If no extension attributes have been defined, the distinct value null will be returned.
* ParseValue
Parses the value provided in the parameter and in case of successful parsing returns a Value object representing the parsed value. In case of an error or if value parsing is not supported by the tool, the method returns null.

### 12.4.4 Abstract value mapping

#### 12.4.4.0 Mapping rules

TTCN-3 values can be retrieved from the TE and constructed using the ITciValue interface. The value mapping interface is constructed hierarchically with ITciValue as the basic interface. Specialized interfaces for different types of values have been defined.

#### 12.4.4.1 Value

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 ();

 public bool IsIfPresentEnabled { get; set; }

 ITciRangeBoundary LowerTypeBoundary { get; }

 ITciRangeBoundary UpperTypeBoundary { get; }

 ITciLengthRestriction TypeLengthRestriction { get; }

 ITciMatchingMechanism TypeMatchingMechanism { get; }
}

**Members:**

* Type
Returns the type of the specified value.
* NotPresent
Returns true if the specified value is omit, false otherwise.
* ValueEncoding
This property returns the value encoding attribute as defined in TTCN-3, if any. If no encoding attribute has been defined the distinct value null will be returned.
* ValueEncodingVariant
This property returns the value encoding variant attribute as defined in TTCN-3, if any. If no encoding variant attribute has been defined the distinct value null will be returned.
* EncodeAttributes
Returns all encode attributes of the value as defined in the TTCN-3 module. If no encode attribute has been defined, the distinct value null is returned.
* GetVariantAttributes
Returns all variant attributes of the value as defined in the TTCN-3 module. If no encoding attribute has been defined, the distinct value null is returned.
* NotPresent
Returns true if the specified value is a matching symbol (see clause 7.2.2.2.1 for more details), false otherwise.
* ValueToString
Returns the same string as produced by the log operation with the specified value as its parameter.
* IsLazy
Returns true if the specified value is @lazy, false otherwise.
* IsFuzzy
Returns true if the specified value is @fuzzy, false otherwise.
* IsEvaluated
Returns true if the specified value contains an evaluation result, false otherwise (see clause 7.2.2.2.1 for more details).
* LengthRestriction
Gets or sets a length restriction matching attribute attached to the value.
* NewLengthRestriction
Creates a new instance of the LengthRestriction interface.
* IfPresentEnabled
Gets or sets whether the the ifpresent matching attribute is attached to the value or not.
* LowerTypeBoundary
Gets the value's type's lower range boundary or null.
* UpperTypeBoundary
Gets the value's type's upper range boundary or null.
* TypeLengthRestriction
Gets the value's type's length restriction or null.
* TypeMatchingMechanism
Gets the values's type restriction matching mechanism.