ETSI ES 203 790 V1.2.1 (2020-05)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

TTCN-3 Language Extensions: Object-Oriented Features

**ETSI Standard**

Reference

RES/MTS-203790-OOFv1.2.1

Keywords

language, 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 prevailing version of an ETSI deliverable is the one made publicly available in PDF format at [www.etsi.org/deliver](http://www.etsi.org/deliver).

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at <https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx>

If you find errors in the present document, please send your comment to one of the following services:
<https://portal.etsi.org/People/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 2020.

All rights reserved.

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

####

#### 5.1.1.0 General

***Syntactical Structure***

[**public** | **private**]
**type** [**external**] **class** [**@final** |**@abstract**]
*Identifier* [**extends** *ClassType*]
[*runsOnSpec*] [*systemSpec*] [*mtcSpec*]
"{" {*ClassMember*} "}"
[**finally** *StatementBlock*]

***Semantic Description***

A class is a type where the values are called objects. A class can declare fields (variables, constants, templates, timers, classes), methods and properties as its members. Each member name inside the class shall be unique, there is no overloading. The private and protected fields and methods are only accessible by the methods of the class, while the public members of the class can be accessed also from behaviour not defined in the class. The private members of the class can be accessed directly only by members of the class itself. All members which are neither private nor public are protected and can also be accessed by members of subclasses.

All fields may be declared without initializer, even const and template fields.

A class can extend another class. The extended class is called the superclass, while the extending class is called the subclass. The resulting type of a class definition is the set of object instances of the class itself and all instances of its direct or indirect subclasses. A subclass is a subtype of its direct and indirect superclasses and its object instances are type compatible with them. If a class does not explicitly extend another class type, it implicitly extends the root class type **object**. Thus, all classes are directly or indirectly extensions of the **object** class.

A class can have optional runs on, mtc and system clauses. This restricts the type of component context that can create objects of that class and all methods of this class. If a class does not have one of these clauses, it inherits it from its superclass, if the superclass has one. If the superclass has or inherits a runs on, mtc or system clause, the subclass may declare each of these clauses with a more specific component type than the one inherited. The function members of classes shall not have runs on, system or mtc classes but inherit them from their surrounding class or its superclasses.

***Restrictions***

1. Templates are not allowed for class types.
2. Passing of object references to the create operation of a component type or a function started on another component is not allowed.
3. No subtyping definition is allowed for class types via the normal subtype definition.
4. No local/global constants or module parameters of class type or containing class type fields or elements are allowed.
5. Class type cannot be the contained value of an anytype value.
6. The functions of a class shall not have a runs on, mtc or system clause.
7. The runs on type of a class shall be runs on compatible with the runs on type of the behaviour creating a class.
8. The runs on type of a class shall be runs on compatible with the runs on type of the superclass.
9. The mtc and system type of a class shall be mtc and system compatible with the mtc and system types of the superclass, respectively.

#### 5.1.1.2 Abstract classes

A class can be declared as @abstract. In that case, it is allowed that it also declares abstract member functions, abstract properties or properties with abstract getters or setters who shall be defined by all non-abstract subclasses. An abstract method function has no function body but can be called in all concrete instances of subclasses of the abstract class declaring it. Other members of the abstract class or its subclasses may use the abstract functions as if it was concrete where at runtime the concrete overriding definition will be used.

Abstract getters and setters have no body but the properties containing them can be referenced in all concrete instance of subclasses of the abstract class declaring them. Other members of the abstract class or its subclasses may reference abstract properties as if they were concrete. At runtime the concrete overriding definition will always be used.

NOTE 1: Abstract classes are only useful as superclasses of concrete classes.

***Restrictions***

1. Abstract classes cannot be explicitly instantiated.
2. If a class that is not declared abstract extends an abstract class, all methods, property getters and setters that have no implementation in the superclass shall be implemented in this class.

NOTE 2: Variables of an abstract class type can only contain references to instances of non-abstract subclasses.

#### 5.1.1.5 Constructors

***Syntactic Structure***

**create** "(" { *FormalParameter* , }\* ")"

**[ external** "(" { *FormalParameter* , }\* ")" ]
[":" *ClassType* "(" { *ActualParameter* , }+ ")" ]
[ *StatementBlock* ]

***Semantic Description***

A class may define a constructor called create.

If no constructor is defined inside a class body, an implicit default constructor is provided where the formal parameters of the constructor are the parameters of the (implicit or explicit) constructor of the direct superclass and one additional formal in parameter for each declared **var** field or automatic property of the class itself unless they are declared with the **@internal** modifierand also all **const** or **template** fields with no initializer in their order of declaration with the same type as in the declaration.

The constructor is invoked on a type reference to the class and the result of this invocation is a new instance object of the constructor's specific class. If a class is extending another class with a constructor with at least one parameter without default, that constructor shall be invoked by adding a super-constructor clause to the constructor declaration. The super-constructor clause consist of a reference to the class being extended and an actual parameter list. An implicit constructor will automatically pass the required actual parameters to the constructor of its superclass.

In the constructor, it is allowed to refer to the object being constructed as this to reference the fields of the object to be created in case that the names of the formal parameters clash with the names of those fields. They are explicitly allowed to have the same names as class members.

When an object is created via the invocation of a constructor, the fields of each class body in the class hierarchy that have initializers are initialized before the execution of that class body’s constructor body. The fields of a superclass that have initializers are initialized before the fields of the subclass. Also, the constructor of the superclass is executed before the constructor body of the subclass. Thus, it is ensured that all initialization of the superclass hierarchy as well as local fields with initializers is finished before the execution of a constructor body.

Since the members of a class body can appear in any order and forward references are allowed between them, a field with an initializer which is referenced by the initializer of another field, is initialized first.

As the underlying external constructor of external classes might need additional parameters, these can be provided via the additional external formal parameter list. If no internal constructor needs to be defined, the constructor may be defined without external formal parameter list and no body. In that case, the formal parameter list defines the formal parameters passed to the external constructor.

***Restrictions***

1. All formal parameters of the constructor shall be **in** parameters.
2. The constructor body shall not assign anything to variables that are not local to the constructor body or accessible fields of the class the constructor belongs to.
3. The constructor body shall not use blocking operations.
4. The initialization of a member field shall not invoke any member function in the object being initialized.
5. The constructor body shall not invoke any member function in the object being initialized.
6. A member constant or template shall be initialized exactly once, either by its initialization part or by at most one constructor body.
7. Direct or indirect cyclic initialization is not allowed. That is the initializer of a field shall not use the same field directly or indirectly.
8. The initializer of a field shall not use a field that does not have an initializer.

EXAMPLE 1:

**type** **class** MyClass {

 **var** **integer** a;

 **const** **float** b;

 const float c := 7;

 template float myTemplate := ?;

 // implicit constructor:

 // only using variable fields and non-variable fields with no initializer

 //**create**(**integer** a, **float** b) { // no parameter for c and myTemplate

 // **this**.a := a;

 // **this**.b := b

 //}

}

**type** **class** MyClass2 extends MyClass {

 **template** **integer** t;

 // explicit constructor

 **create**(**template** **integer** t) : MyClass(2, 0.5) {

 this.t := t;

 }

}

type class MyClass3 extends MyClass {

 var float f;

 // implicit constructor:

 // create(integer a, float b, float f) : MyClass(a, b) {

 // this.f := f;

 // }

}

EXAMPLE 2:

For each initialization statement it is marked with its initialization order in the comment.

**type** **class** MySuperClass {

 **var** **integer** a := 5; // 1

 **const** **float** b;

 **create**(**integer** a, **float** b) {

 **this**.a := a; // 3

 **this**.b := b; // 4

 }

}

**type** **class** MySubClass **extends** MySuperClass {

 **var** **template** **integer** t := ?; // 2

 **create**(**template** **integer** t) : MySuperClass(2, 0.5) {

 **this**.t := t; // 5

 }

}

#### 5.1.1.13 Properties

***Syntactic Structure***

@**property** [ *TemplateModifier* ] { ( **@abstract** | **@final** | **@deterministic** | **@internal )**} *Type Identifier*

[ ":=" *TemplateBody* ]

[ **"{"**

 [ { ( **@abstract** | **@final** | **@deterministic** ) }

**@get** [

 ("=>" *TemplateBody* [";"]) |

 ("{" *StatementBlock* "}" )

 ]

 ]

 [ { ( **@abstract** | **@final** | **@deterministic ) }**

**@set** [

 ("=>" *Assignment* [";"]) |

 ("{" *StatementBlock* "}" )

 ]

 ]

"}" ]

***Semantic Description***

A class property is a class member which is referenced like a record field for reading and writing with the dotted notation, but implemented via getter and setter functions that are provided in the definition of the property (allowing value checking/normalization/conversion when setting a value and on-the-fly computation when getting the value).

Properties are in many regards similar to member functions, they can be declared with modifiers and it is allowed to override them in subclasses. When overriding a property, it is allowed to add a getter or setter even if it was not present in the parent class. Such a getter or setter is available only in the overriding class and its subclasses. When overriding a property, if a getter or setter is present in the parent class but not in the overriding class, then the getter or setter from the parent class is inherited by the overriding class.

Modifiers can occur either on the property level or in a getter and setter declaration. Modifiers declared on a property level are valid for both the getter and setter.

While most properties are declared with a property body containing either a getter or setter function, it is allowed to declare a property without a body. Such a property is called an automatic property. Automatic properties are similar to member variables, they are always generated with an associatied anonymous member variable and a getter that returns this variable and a setter that assigns a value to this variable. While they are similar to member variables, there are two important differences: automatic properties can be public and it is possible to override them in subclasses. If a class contains an implicit default constructor, this constructor contains a formal parameter for each anonymous member variable associated with an automatic property with the name of the automatic property as the name of the formal parameter.

All getters and setters shall have a body unless they are declared abstract. A body of a getter can be specified in two distinct forms: either in a simple form that contains a single *TemplateBody* or in an extended form that consist of a statement block. In case of the simple form, referencing the property on the right hand side of an assignment will return the *TemplateBody* referenced in the getter declaration. When the extended form is referenced on the right hand side of an assignment, the statement block of the getter function will be executed and the value from the **return** clause of this block will be returned.

A body of a setter can be specified in two distinct forms: either in a simple form that contains a single assignment or in an extended form that consist of a statement block. The setter is executed when the property is referenced on the left hand side of an assignment. Both forms of the setter may reference a special variable **value**. This variable works as an **in** formal parameter of the setter function. It is of the same type as the property itself and when the setter is invoked, the value from the left hand side of the assignment where the property was referenced is passed into it as an actual parameter according to the rules specified in the section 5.4.2 of [1]. The **value** variable of the setter function has the same template modifiers as the property itself.

Properties that contain a setter (including automatic properties that have an implicit setter) may be optionally declared with an initial value. The initial value follows the identifier of the property and is preceded by an assign symbol. The initial value is automatically passed to the setter when an instance of the defining class is created. This automatic invocation takes place after execution of a constructor of the parent class and before execution of the constructor of the defining class. Properties are automatically initialized in the declaration order.

***Restrictions***

1. With the exception of a special case of automatic properties, a property shall always have a getter or a setter or both of them. An empty property body is not allowed.
2. The *TemplateBody* in the simplified form of the getter function and in the return clause of the common form of the getter function shall be compatible with the property type according to the rules specified in the section 6.3 of [1]. If the property has no *TemplateModifier*, the *TemplateBody* shall contain a value. Otherwise, it might contain a template that fulfils the restrictions set by the *TemplateModifier* that are specified in the section 15.8 of [1].
3. When passing a value to the value variable of the setter function, the rules specified in the section 5.4.2 of [1] shall apply.
4. If a property or its getter or setter contain the **@deterministic** modifier, rules for derministic functions specified in the section 16.1.0 of [1] shall apply to the body of the concerned getter or setter.
5. An error shall be produced when a property that has no getter is referenced on the right hand side of an assignment.
6. An error shall be produced when a property that has no setter is referenced on the left hand side of an assignement.
7. An error shall be produced if execution of the extended form of a getter is terminated by reaching the end of the statement block without executing a **return** statement or a statement that terminates component execution (such as **stop** or **testcase.stop**).
8. The initial value of a property shall be compatible with the property type. If the property has no *TemplateModifier*, the initial value shall resolve into a value. Otherwise, it might resolve into a template that fulfills the restrictions set by the *TemplateModifier* of the property. These restrictions are specified in the section 15.8 of [1].
9. An error shall be produced if an index or dot notation is applied to a property referenced on the left hand side of an assignment.

EXAMPLE:

**type class** Rectangle {

 **private var integer** heightVal;

 **public @property integer** width; // automatic property width

 **public @property integer** height { // property with a getter and setter

 **@get** => heightVal; // simple form of a getter

 **@set** { // extended form of a setter

 if (**value** > 0) { // simple data integrity check

 heightVal := **value**;

 }

 }

 }

 **public @property integer** perimeter {

 **@get** { // extended form of a getter

 **return** 2 \* (width + height);

 }

 }

}

…

**var** Rectangle v\_rect := Rectangle.**create**(heightVal := 10, width := 9); // instantiation using an

 // implicit constructor

v\_rect.width := 16; // change the width to 16

**log** (v\_rect.perimeter); // prints 144 to the log

v\_rect.perimeter := 100; // causes an error as the referenced property has no setter

v\_rect.height := -100; // does not change the heightVal variable

## 7.5 Extensions to clause 7.3.4.1 of ETSI ES 201 873-6 Abstract TCI-TL provided

**Clause 7.3.4.1.122 tliObjCreateEnter**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjCreateEnter(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TciParameterListType tciPars) |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| obj | The object being created. |
| tciPars | The parameters of the constructor. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the entering of a constructor of an object. This event occurs after the constructor has been entered. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.123 tliObjCreateLeave**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjCreateLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TciParameterListType tciPars) |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| obj | The created object instance. |
| tciPars | The parameters of the constructor. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the leaving of an object constructor. This event occurs after the constructor has been left. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.124 tliObjFinallyEnter**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjFinallyEnter(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj) |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| obj | The object instance being destroyed. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the entering of a destructor of an object. This event occurs after the destructor has been entered. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.125 tliObjFinallyLeave**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjCreateLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TciParameterListType tciPars) |
| In Parameters | am | An additional message. |
| ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| c | The component which produces this event. |
| obj | The object being destroyed. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the leaving of an object destructor. This event occurs after the destructor has been left. Accessing any members, properties and methods of a destroyed object with exception of methods used for comparison shall cause an error. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.126 tliObjMethodEnter**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjMethodEnter(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString methodName, in TciParameterListType tciPars) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| C | The component which produces this event. |
| obj | The affected object instance. |
| methodName | The name of the called method. |
| tciPars | The parameters of the called method. |
| Return Value | void |
| Constraint | Shall be called by TE to log the entering of an object method. This event occurs after the method has been entered. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.127 tliObjMethodLeave**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjMethodLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString methodName, in TciParameterListType tciPars, in Value returnValue) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| C | The component which produces this event. |
| obj | The affected object instance. |
| methodName | The name of the called method. |
| tciPars | The parameters of the called method. |
| returnValue | The return value of the called method. |
| Return Value | void |
| Constraint | Shall be called by TE to log the leaving of an object method. This event occurs after the method has been left. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.132 tliObjVar**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjVar(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString name, in Value value) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| Src | The source file of the test specification. |
| Line | The line number where the request is performed. |
| C | The component which produces this event. |
| Obj | The affected object instance. |
| name | The name of the member variable. |
| value | The new value of the member variable. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the modification of the value of a field of an object. This event occurs after the field value has been changed. In case of @lazy fields, it is called also after performing evaluation as the evaluation result is automatically assigned to the field. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.133 tliObjGetEnter**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjGetEnter(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString propertyName) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| C | The component which produces this event. |
| obj | The affected object instance. |
| propertyName | The name of the referenced property. |
| Return Value | void |
| Constraint | Shall be called by TE to log the entering of an object getter. This event occurs after the getter has been entered. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.134 tliObjGetLeave**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjGetLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString propertyName, in Value value) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| Src | The source file of the test specification. |
| Line | The line number where the request is performed. |
| C | The component which produces this event. |
| Obj | The affected object instance. |
| propertyName | The name of the referenced property. |
| returnValue | The value return by the getter. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the leaving of an object getter. This event occurs after the getter has been left. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.135 tliObjSetEnter**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjSetEnter(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString propertyName, in Value value) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| C | The component which produces this event. |
| obj | The affected object instance. |
| propertyName | The name of the referenced property. |
| value | The value passed to the setter. |
| Return Value | Void |
| Constraint | Shall be called by TE to log the entering of an object setter. This event occurs after the setter has been entered. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

**Clause 7.3.4.1.136 tliObjSetLeave**

This clause is to be added.

|  |  |
| --- | --- |
| Signature | void tliObjGetLeave(in TString am, in TInteger ts, in TString src,  in TInteger line, in TriComponentIdType c,  in ObjectInstance obj, in TString propertyName) |
| In Parameters | Am | An additional message. |
| Ts | The time when the event is produced. |
| src | The source file of the test specification. |
| line | The line number where the request is performed. |
| C | The component which produces this event. |
| obj | The affected object instance. |
| propertyName | The name of the referenced property. |
| Return Value | void |
| Constraint | Shall be called by TE to log the leaving of an object setter. This event occurs after the setter has been left. |
| Effect | The TL presents all the information provided in the parameters of this operation to the user, how this is done is not within the scope of the present document. |

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

**Clause 8.3.2.4 TciTypeClassType**

This clause is to be extended.

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

// TCI IDL TciTypeClassType

package org.etsi.ttcn.tci;

public interface TciTypeClass {

 public final static int ADDRESS = 0 ;

 public final static int ANYTYPE = 1 ;

 public final static int BITSTRING = 2 ;

 public final static int BOOLEAN = 3 ;

 public final static int CHARSTRING = 5 ;

 public final static int COMPONENT = 6 ;

 public final static int ENUMERATED = 7 ;

 public final static int FLOAT = 8 ;

 public final static int HEXSTRING = 9 ;

 public final static int INTEGER = 10 ;

 public final static int OCTETSTRING = 12 ;

 public final static int RECORD = 13 ;

 public final static int RECORD\_OF = 14 ;

 public final static int ARRAY = 15 ;

 public final static int SET = 16 ;

 public final static int SET\_OF = 17 ;

 public final static int UNION = 18 ;

 public final static int UNIVERSAL\_CHARSTRING = 20 ;

 public final static int VERDICT = 21 ;

 public final static int DEFAULT = 22 ;

 public final static int PORT = 23 ;

 public final static int TIMER = 24 ;

 public final static int CLASS = 25 ;

}

**Clause 8.3.6.7 Abstract class mapping**

This clause is to be added.

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

// TCI IDL Type

package org.etsi.ttcn.tci;

public interface Class extends Type {

 public ObjectInstance create (TriComponentId c, TciParameterList tciPars);

 public Class[] getSuperclasses ();

 public String[] getFieldNames ();

 public String[] getMethodNames ();

 public TciParameterTypeList getConstructorParmeters ();

 public TciParameterTypeList getMethodParameters (String methodName);

 public Type getFieldType (String name);

 public Type getMethodReturnType (String methodName);

}

**Methods:**

* create Calls the constructor to create a new instance of this class using the supplied parameters for the specified component.
* getSuperclasses Returns the list of superclasses of this class.
* getFieldNames Returns the names of all public fields defined in the class.
* getMethodNames Returns the names of all public methods of the class.
* getConstructorParmeters Returns the formal parameters of the class constructor.
* getMethodParameters Returns the formal parameters of the specified public method.
* getFieldType Returns the type of the specified public field.
* getMethodReturnType Returns the return type of specified public method or the distinct value null if no return type is declared.

**Clause 8.3.6.8 ClassSeq**

This clause is to be added.

**ClassSeq** abstract data type mapped to an array of TciClass.

**Clause 8.3.4.16 ObjectInstance**

This clause is to be added.

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

// TCI IDL DynamicMatch

package org.etsi.ttcn.tci;

public interface ObjectInstance extends Value {

 public TriComponentId getOwner ();

 public TString getId ();

 public void setObject (ObjectInstance source);

 public Value callMethod (String methodName, TciParameterList tciPars);

}

**Methods:**

* getOwner Returns the component that owns the object instance.
* getId Returns the unique identifier of the object instance.
* setObject The operation sets the referenced object to the same reference as the given object.
* getField Gets the value of the referenced public field.
* callMethod Calls a method of the object instance.

**Clause 8.5.4.1 TCI-TL provided**

The TciTLProvided interface is to be extended:

package org.etsi.ttcn.tci;

public interface TciTLProvided {

 …

 public void tliObjCreateEnter(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, TciParameterList tciPars);

 public void tliObjCreateLeave(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, TciParameterList tciPars);

 public void tliObjFinallyEnter(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj);

 public void tliObjFinallyLeave(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj);

 public void tliObjMethodEnter(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String methodName, TciParameterList tciPars);

 public void tliObjMethodLeave(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String methodName, TciParameterList tciPars, Value returnValue);

 public void tliObjVar(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String name, Value value);

 public void tliObjGetEnter(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String propertyName);

 public void tliObjGetLeave(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String propertyName, Value returnValue);

 public void tliObjSetEnter(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String propertyName, Value value);

 public void tliObjSetLeave(String am, int ts, String src, int line, TriComponentId c,

 ObjectInstance obj, String propertyName);

}

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

**Clause 9.2 Data**

The table 5 is to be extended.

| TCI IDL Interface | ANSI C representation | Notes and comments |
| --- | --- | --- |
| : |  |  |
| Class |
| Value create(TriComponentIdType c, TciParameterListType tciPars) | Value tciObjCreate(Type cls, TriComponentId c, TciParameterListType tciPars) |  |
| ClassSeq getSuperclasses () | Type\* tciGetSuperclasses (Type cls) | Returns null pointer or a null‑pointer terminated array |
| TStringSeq getFieldNames () | String\* tciGetClassFieldNames (Type cls) | Returns null pointer or a null‑pointer terminated array |
| TStringSeq getMethodNames () | String\* tciGetClassMethodNames (Type cls) |
| TciParameterTypeListType getConstructorParmeters () | TciParameterTypeListType\* tciGetClassConstructorParameters (Type cls) |  |
| TciParameterTypeListType getMethodParameters (TString methodName) | TciParameterTypeListType\* tciGetClassMethodParameters (Type cls, String methodName) |  |
| Type getMemberType (TString name) | Type tciGetClassFieldType(Type cls, String name) |  |
| Type getMethodReturnType (TString methodName) | Type tciGetClassMethodReturnType (Type cls, String methodName) |  |
| **ObjectInstance** |
| TriComponentIdType getOwner () | TriComponentId tciGetObjOwner (Value obj) |  |
| TString getId () | char \* tciGetObjUniqueId (Value obj) |  |
| void setObject (ObjectInstance source) | void tciSetObject (Value obj, Value source) |  |
| Value getField (TString fieldName) | Value tciGetObjField (Value obj, String fieldName) |  |
| Value callMethod(TString methodName, TciParameterListType tciPars) | Value tciCallObjMethod(Value obj, String methodName, TciParameterListType tciPars) |  |

**Clause 9.4.4.1 TCI-TL provided**

The clause is to be extended.

void tliObjCreateEnter

 (String am, int ts, String src, int line, TriComponentId c, Value obj,

TciParameterListType tciPars);

void tliObjCreateLeave

 (String am, int ts, String src, int line, TriComponentId c, Value obj,

TciParameterListType tciPars);

void tliObjFinallyEnter

 (String am, int ts, String src, int line, TriComponentId c, Value obj);

void tliObjFinallyLeave

 (String am, int ts, String src, int line, TriComponentId c, Value obj);

void tliObjMethodEnter

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String methodName,

TciParameterListType tciPars);

void tliObjMethodLeave

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String methodName,

TciParameterListType tciPars, Value returnValue);

void tliObjVar

(String am, int ts, String src, int line, TriComponentId c, Value obj, String name,

Value value);

void tliObjGetEnter

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String propertyName);

void tliObjGetLeave

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String propertyName,

 Value returnValue);

void tliObjSetEnter

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String propertyName,

 Value value);

void tliObjSetLeave

 (String am, int ts, String src, int line, TriComponentId c, Value obj, String propertyName);

**Clause 9.5 Data**

The definition of the TciTypeClassType in the table 7 is to be modified.

| TCI IDL ADT | ANSI C representation (Type definition) | Notes and comments |
| --- | --- | --- |
| : |  |  |
| TciTypeClassType | typedef enum{TCI\_ADDRESS\_TYPE = 0,TCI\_ANYTYPE\_TYPE = 1,TCI\_BITSTRING\_TYPE = 2,TCI\_BOOLEAN\_TYPE = 3,TCI\_CHARSTRING\_TYPE = 5,TCI\_COMPONENT\_TYPE = 6,TCI\_ENUMERATED\_TYPE = 7,TCI\_FLOAT\_TYPE = 8,TCI\_HEXSTRING\_TYPE = 9,TCI\_INTEGER\_TYPE = 10, TCI\_OCTETSTRING\_TYPE = 12,TCI\_RECORD\_TYPE = 13,TCI\_RECORD\_OF\_TYPE = 14, TCI\_ARRAY\_TYPE = 15,TCI\_SET\_TYPE = 16,TCI\_SET\_OF\_TYPE = 17,TCI\_UNION\_TYPE = 18,TCI\_UNIVERSAL\_CHARSTRING\_TYPE = 20,TCI\_VERDICT\_TYPE = 21TCI\_DEFAULT\_TYPE = 22,TCI\_PORT\_TYPE = 23,TCI\_TIMER\_TYPE = 24,TCI\_CLASS\_TYPE = 25} TciTypeClassType; |  |
| : |  |  |

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

**Clause 10.5.2.14 TciTypeClass**

This clause is to be extended.

typedef enum

{

 TCI\_ADDRESS = 0,

 TCI\_ANYTYPE = 1,

 TCI\_BITSTRING = 2,

 TCI\_BOOLEAN = 3,

 TCI\_CHARSTRING = 5,

 TCI\_COMPONENT = 6,

 TCI\_ENUMERATED = 7,

 TCI\_FLOAT = 8,

 TCI\_HEXSTRING = 9,

 TCI\_INTEGER = 10,

 TCI\_OCTETSTRING = 12,

 TCI\_RECORD = 13,

 TCI\_RECORD\_OF = 14,

 TCI\_ARRAY = 15,

 TCI\_SET = 16,

 TCI\_SET\_OF = 17,

 TCI\_UNION = 18,

 TCI\_UNIVERSAL\_CHARSTRING = 20,

 TCI\_VERDICT = 21

 TCI\_DEFAULT = 22,

 TCI\_PORT = 23,

 TCI\_TIMER = 24

 TCI\_CLASS = 25

} TciTypeClass;

**Clause 10.5.3.23 Class**

This clause is to be added.

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

class TciClass : public virtual TciType {

public:

 virtual ~TciClass ();

 virtual ObjectInstance \* create(const TriComponentId & c, TciParameterList & tciPars) =0;

 virtual const std::vector<TciClass\*> & getSuperclasses () const =0;

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

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

 virtual const TciParameterTypeList & getConstructorParmeters () const =0;

 virtual const TciParameterTypeList & getMethodParameters (Tstring methodName) const =0;

 virtual const TciType & getMemberType (const Tstring & name) const =0;

 virtual const TciType & getMethodReturnValue (const Tstring & name) const =0;

 virtual Tboolean operator== (const TciClass &p\_class) const =0;

 virtual TciClass \* clone () const =0;

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

}

**Methods:**

~TciClass

 Destructor

create

 Calls the constructor to create a new instance of this class using the supplied parameters for the specified component

getSuperclasses

 Returns the superclasses of the current class

getFieldNames

 Returns the names of all public fields defined in the class

getMethodNames

 Returns the names of all public methods of the class

getConstructorParmeters

 Returns formal parameters of the class constructor

getMethodParameters

 Returns formal parameters of the specified public method

getFieldType

 Returns the type of the specified public field

getMethodReturnValue

 Returns the return type of specified public method or the distinct value null if no return value is defined

operator==

 Returns true if both objects are equal

clone

 Return a copy of the matching mechanism

operator<

 Operator < overload

**Clause 10.5.3.24 ObjectInstance**

This clause is to be added.

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

class ObjectInstance : public virtual TciValue {

public:

 virtual ~ObjectInstance ();

 virtual const TriComponentId & getOwner () const =0;

 virtual const TString getId () const =0;

 virtual void setObject (ObjectInstance & val) =0;

 virtual TciValue \* getField (const Tstring & fieldName) =0;

 virtual Value callMethod(const TString & methodName, TciParameterList & tciPars) =0;

 virtual Tboolean operator== (const ObjectInstance &p\_obj) const =0;

 virtual ObjectInstance \* clone () const =0;

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

}

**Methods:**

~ObjectInstance

 Destructor

getOwner

 Returns the component that owns the object instance

getId

 Returns the unique identifier of the object instance

setObject

 The operation sets the referenced object

getField

 Returns the value of the referenced public field

callMethod

 Calls a method of the object instance

operator==

 Returns true if both objects are equal

clone

 Return a copy of the matching mechanism

operator<

 Operator < overload

**Clause 10.5.3.25 ClassSeq**

This clause is to be added.

The ClassSeq abstract data type is mapped to std::vector<TciClass\*> .

**Clause 10.6.4.1 TciTlProvided**

This clause is to be extended.

//Called by TE to log the entering of a constructor

virtual void tliObjCreateEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const TciParameterList \*tciPars)=0;

//Called by TE to log the leaving of a constructor

virtual void tliObjCreateLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const TciParameterList \*tciPars)=0;

//Called by TE to log the entering of a destructor

virtual void tliObjFinallyEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj)=0;

//Called by TE to log the leaving of a destructor

virtual void tliObjFinallyLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj)=0;

//Called by TE to log the entering of an object method

virtual void tliObjMethodEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &methodName, const TciParameterList \*tciPars)=0;

//Called by TE to log the leaving of an object method

virtual void tliObjMethodLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &methodName, const TciParameterList \*tciPars, const TciValue \*returnValue)=0;

//Called by TE to log the modification of a member variable of an object

virtual void tliObjVar (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &name, const TciValue \*value)=0;

//Called by TE to log the entering of an object getter

virtual void tliObjGetEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &propertyName)=0;

//Called by TE to log the leaving of an object getter

virtual void tliObjGetLeave (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &propertyName, const TciValue \*returnValue)=0;

//Called by TE to log the entering of an object setter

virtual void tliObjSetEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &propertyName, const TciValue \*value)=0;

//Called by TE to log the leaving of an object setter

virtual void tliObjSetEnter (const Tstring &am, const timeval ts, const Tstring &src, const Tinteger line, const TriComponentId \*c, const ObjectInstance \*obj, const Tstring &propertyName)=0;

## 7.9 Extensions to clause 11 of ETSI ES 201 873-6 W3C XML mapping

**Clause 11.3.3.30 ObjectInstance**

ObjectInstance type is mapped to the complex type specified below. The content of the XML elements based on the ObjectInstance type shall be equal to the string produced by the valueToString operation (described in clause 7.2.2.2.1 of ETSI ES 201 873-6 [4]):

 <xsd:complexType name="ObjectInstance">

 <xsd:group ref="Values:BaseValue"/>

 <xsd:attributeGroup ref="Values:ValueAtts"/>

 </xsd:complexType>

**Items:**

* BaseValue Object instance content described in clause 11.3.3.1 of ETSI ES 2001-873-6
* ValueAtts Value attributes described in clause 11.3.3.1 of ETSI ES 2001-873-6

**Clause 11.4.2.1 TCI-TL provided**

**This clause is to be extended.**

 <xsd:complexType name="tliObjCreateEnter">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjCreateLeave">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

 <xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjFinallyEnter">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjFinallyLeave">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjMethodEnter">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="methodName" type="SimpleTypes:TString" />

 <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjMethodLeave">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="methodName" type="SimpleTypes:TString" />

 <xsd:element name="tciPars" type="Types:TciParameterListType" minOccurs="0"/>

 <xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjVar">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="name" type="SimpleTypes:TString" />

 <xsd:element name="val" type="Values:Value" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 xsd:complexType name="tliObjGetEnter">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="propertyName" type="SimpleTypes:TString" />

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjGetLeave">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="propertyName" type="SimpleTypes:TString" />

 <xsd:element name="returnValue" type="Values:Value" minOccurs="0"/>

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 <xsd:complexType name="tliObjSetEnter">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="propertyName" type="SimpleTypes:TString" />

 <xsd:element name="value" type="Values:Value" />

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

 xsd:complexType name="tliObjSetLeave">

 <xsd:complexContent mixed="true">

 <xsd:extension base="Events:Event">

 <xsd:sequence>

 <xsd:element name="obj" type="Values:ObjectInstance" />

 <xsd:element name="propertyName" type="SimpleTypes:TString" />

 </xsd:sequence>

 </xsd:extension>

 </xsd:complexContent>

 </xsd:complexType>

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

**Clause 12.4.2.4 TciTypeClassType**

This clause is to be extended.

**TciTypeClassType** is mapped to the following enumeration:

public enum TciTypeClass {
 Address = 0,
 Anytype = 1,
 Bitstring = 2,
 BooleanType = 3,
 Charstring = 5,
 Component = 6,
 Enumerated = 7,
 Float = 8,
 Hexstring = 9,
 IntegerType = 10,
 Octetstring = 12,
 Record = 13,
 RecordOf = 14,
 Array = 15
 Set = 16,
 SetOf = 17,
 Union = 18,
 UniversalCharstring = 20,
 Verdict = 21,
 Default = 22,
 Port = 23,
 Timer = 24,
 Class = 25
}

**Clause 12.4.7 Abstract class mapping**

This clause is to be added.

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

// TCI IDL Type

package org.etsi.ttcn.tci;

public interface ITciClass : ITciType {

 ITciObjectInstance Create (ITriComponentId c, ITciParameterList tciPars);

 ITciClass[] Superclasses { get; }

 String[] FieldNames { get };

 String[] MethodNames { get; }

 ITciParameterTypeList ConstructorParmeters { get; }

 ITciParameterTypeList GetMethodParameters (String methodName);

 ITciType GetFieldType (String name);

 ITciType GetMethodReturnType (String methodName);

}

**Methods:**

* Create Calls the constructor to create a new instance of this class using the supplied parameters for the specified component.
* Superclasses Returns the list of superclass of the current.
* FieldNames Returns the names of all public fields defined in the class.
* MethodNames Returns the names of all public methods of the class.
* ConstructorParmeters Returns formal parameters of the class constructor.
* GetMethodParameters Returns formal parameters of the specified public method.
* GetFieldType Returns the type of the specified public field.
* GetMethodReturnType Returns the return type of specified public method or the distinct value null if no return value is defined.

**Clause 12.4.8 ClassSeq mapping**

This clause is to be added.

The **ClassSeq** abstract data type is mapped to **ITciClass[]**.

**Clause 12.4.4.16 ObjectInstance**

This clause is to be added.

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

public interface ITciObjectInstance : ITciValue {

 ITciComponentId Owner { get; }

 String Id { get; }

 void SetObject (ITciObjectInstance source);

 ITciValue GetField (String fieldName);

 ITciValue CallMethod (String methodName, ITciParameterList tciPars);

}

**Methods:**

* Owner Returns the component that owns the object instance.
* Id Returns the unique identifier of the object instance.
* SetObject The operation sets the referenced object.
* GetField Returns the value of the referenced public field.
* CallMethod Calls a method of the object instance.

**Clause 12.5.4.1 TCI-TL provided**

The ITciTLProvided interface is to be extended:

public interface ITciTLProvided {

…

 void TliObjCreateEnter(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, ITciParameterList tciPars);
 void TliObjCreateLeave(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, ITciParameterList tciPars);
 void TliObjFinallyEnter(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj);
 void TliObjFinallyLeave(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj);
 void TliObjMethodEnter(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string methodName, ITciParameterList tciPars);
 void TliObjMethodLeave(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string methodName, ITciParameterList tciPars,
 ITciValue returnValue);
 void TliObjVar (string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string name, ITciValue value);

 void TliObjGetEnter(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string propertyName);
 void TliObjGetLeave(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string propertyName, ITciValue returnValue);
 void TliObjSetEnter(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string propertyName, ITciValue value);
 void TliObjSetLeave(string am, System.DateTime ts, string src, int line,
 ITriComponentId c, ITciObjectInstance obj, string propertyName);
}