Final draft ETSI ES 201 873-8 V4.8.1 (2021-03)

Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping

Reference
RES/MTS-201873-8T3ed481

Keywords
IDL, testing, TTCN

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

The present document can be downloaded from:
http://www.etsi.org/standards-search

Copyright Notification

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This final draft ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 8 of a multi-part deliverable. Full details of the entire series can be found in part 1 [1].

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document defines the mapping rules for CORBA IDL (as defined in clause 3 in [4]) to TTCN-3 (as defined in ETSI ES 201 873-1 [1]) to enable testing of CORBA-based systems. The principles of mapping CORBA IDL to TTCN-3 can be also used for the mapping of interface specification languages of other object-/component-based technologies.

The specification of other mappings is outside the scope of the present document.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".

[2] Recommendation ITU-T T.50: "International Reference Alphabet (IRA) (Formerly International Alphabet No. 5 or IA5) - Information technology - 7-bit coded character set for information interchange".

[3] ISO/IEC 10646:2017: "Information technology -- Universal Coded Character Set (UCS)".

[4] CORBA® 3.0: "The Common Object Request Broker: Architecture and Specification".

NOTE: Available at http://www.omg.org/spec/CORBA/.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".

[i.2] Void.

[i.3] Void.

[i.4] Void.

[i.5] ETSI ES 202 781: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support".

[i.6] ETSI ES 202 782: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing".

[i.7] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization".

[i.8] ETSI ES 202 785: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behaviour Types".

[i.9] ETSI ES 202 786: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Support of interfaces with continuous signals".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASN.1 Abstract Syntax Notation One
CCM CORBA Component Model

NOTE: By OMG®.

CORBA Common Object Request Broker Architecture

NOTE: By OMG®.

DCE Distributed Computing Environment

NOTE: By OSF.

EJB Enterprise JavaBeans™

NOTE: By Sun®.

IDL Interface Definition Language
NET XML-based component technology

NOTE: By Microsoft®.

OMG Object Management Group
OSF Open Software Foundation
SUT System Under Test
TTCN Testing and Test Control Notation
XML eXtended Markup Language

4 General considerations

4.1 Introduction

Object-based technologies (such as CORBA, DCOM, DCE) and component-based technologies (such as CCM, EJB, Microsoft®, NET) use interface specifications to describe the structure of an object-/component-based system and its operations and capabilities to interact with the environment. These interface specifications support interoperability and reusability of objects/components.

The techniques used for interface specifications are often called Interface Definition Language (IDL), for example CORBA IDL, Microsoft® IDL or DCE IDL. These languages are comparable in their abilities to define system interfaces, operations at system interfaces and system structures to various extends. They differ in details of the object/component model.

When considering the testing of object-/component-based systems with TTCN-3, one is faced with the problem of accessing the systems to be tested via the system interfaces as described in an IDL specification. In particular, for TTCN-3 based test systems a direct import of IDL specifications into the test specifications for the use of e.g. system's interface, operation and exception definitions is prevalent to any manual transformation into TTCN-3.

The present document discusses the mapping of CORBA IDL specifications into TTCN-3. This mapping rules out the principles not only for CORBA IDL, but also for other interface specification languages. The mapping can be adapted to the details of other interface specification languages.

The Interface Definition Language (IDL) (clause 3 in [4]) is a base of the whole Common Object Request Broker Architecture (CORBA) [4] and an important point in developing distributed systems with CORBA. It allows the reuse and interoperability of objects in a system. A mapping between IDL and a programming language is defined in the CORBA standard. IDL is very similar to C++ containing pre-processor directives (include, comments, etc.), grammar as well as constant, type and operation declarations. There are no programming language features like, e.g. `if-statements`.

The core language of TTCN-3 is defined in ETSI ES 201 873-1 [1] and provides a full text-based syntax, static semantics and operational semantics. The IDL mapping provides a definition for the use of the core language with IDL (figure 1).

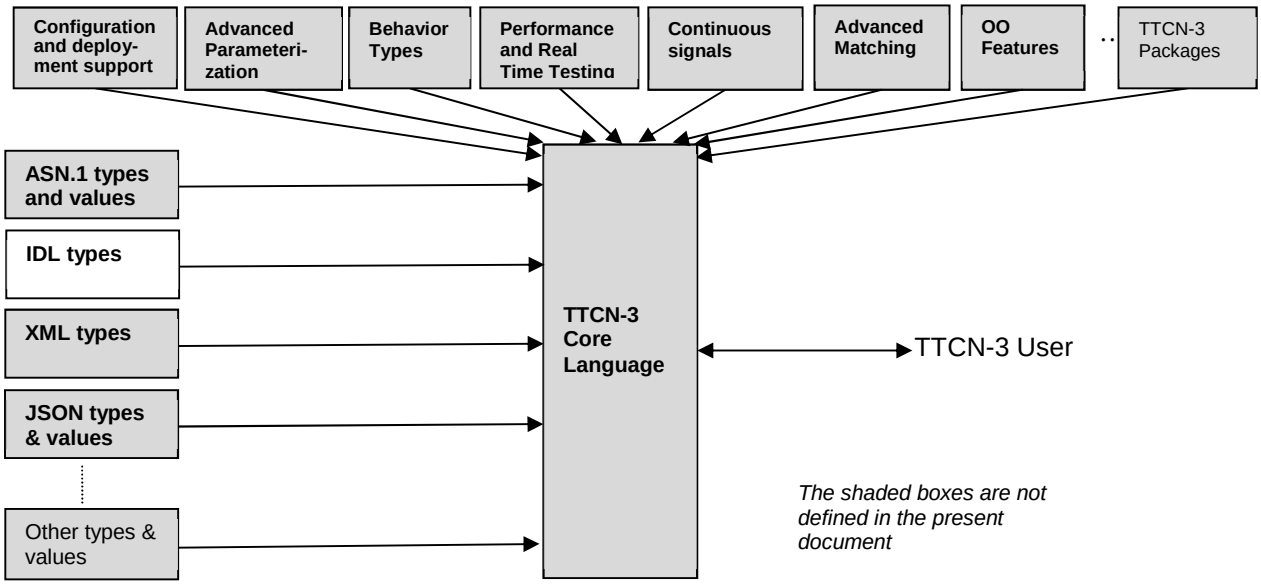

Figure 1: User's view of the core language and its packages

It makes no difference for the mapping if requested or provided interfaces are required by the test system and SUT. Hence, TTCN can be used on client and server side without modifications to the mapping rules.

The present document is structured similar to the IDL specification document to provide easy access to the mapping of each IDL element.

4.2 Approach

Two different approaches can be identified: the use of either implicit or explicit mapping. The implicit mapping makes use of the import mechanism of TTCN-3, denoted by the keywords language and import. It facilitates the immediate use of data specified in other languages. Therefore, the definition of a specific data interface for each of these languages is required. Currently, ASN.1 data can be used besides the native TTCN-3 types (see ETSI ES 201 873-7 [i.1]).

The present document follows the approach of explicit mapping, i.e. IDL data are translated into appropriate TTCN-3 data. And only those TTCN-3 data are further used in the test specification.

4.3 Conformance and compatibility

For an implementation claiming to support the IDL to TTCN-3 mapping, all features specified in the present document shall be implemented consistently with the requirements given in the present document and in ETSI ES 201 873-1 [1].

5 Lexical Conventions

5.0 General

The lexical conventions of IDL define the comments, identifiers, keywords and literals conventions which are described in the following clauses.

5.1 Comments

Comment definitions in TTCN-3 and IDL are the same and therefore, no conversion of comments is necessary.

5.2 Identifiers

IDL identifier rules define a subset of the TTCN-3 rules in which no conversion is necessary.

5.3 Keywords

When IDL is used with TTCN-3 the keywords of TTCN-3 shall not be used as identifiers in an IDL module.

5.4 Literals

The definition of literals differs slightly between IDL and TTCN-3 why some modifications have to be made. Table B.1 gives the mapping for each literal type.

Table 1: Literal mapping

Literal	IDL	TTCN
Integer	no "0" as first digit	no "0" as first digit
Octet	"0" as first digit	'FF96'O
Hex	"0X" or "0x" as first digits	'AB01D'H
Floating	1222.44E5 (Base 10)	1222.44E5 (Base 10)
Char	'A'	"A"
Wide char	L"A"	"A"
Boolean	TRUE, FALSE	true, false
String	"text"	"text"
Wide string	L"text"	"text"
Fixed point	33.33D	(see useful type IDLfixed)

IDL uses the ISO Latin-1 character set for **string** and **wide string** literals and TTCN-3 uses Recommendation ITU-T T.50 [2] for **string** literals and ISO/IEC 10646 [3] for **wide string** literals.

6 Pre-processing

Pre-processor statements are not matched to TTCN-3 because the IDL specification shall be used after pre-processing it.

7 Importing from IDL specifications

7.0 General

The import of module, interface, value and constant declaration are described in this clause. The type and exception declaration as well as the bodies of interfaces are described later.

All imported IDL declarations are in TTCN-3 **public** by default (see clause 8.2.5 of ETSI ES 201 873-1 [1]).

7.1 Importing module declaration

IDL modules are mapped to TTCN-3 modules. Nested IDL modules shall be flattened accordingly to TTCN-3 modules.

As one IDL module can contain many nested IDL modules where several nested modules can have equal names in different scopes, these names can clash. Hence, module names identifiers are to be used which are composed of the identifiers of the upper level IDL modules (from hierarchical point of view) and the nested IDL module name, separated one from each other by two underscores.

According to the IDL scoping rules nested modules have access to the scope of upper level modules. As there are no nested modules in TTCN-3, TTCN-3 modules have to import upper level modules. For avoiding name clashes, a prefix for the imported definitions composed of the identifier of the module from which it is imported shall be used. The prefix and the identifier are separated by a dot (.) as defined in TTCN-3.

IDL EXAMPLE:

```
module identifier1 {
    typedef long mylong1;

   module identifier2 {
      typedef string mystring2;
      typedef mylong1 mylong2;

      module identifier3 {
        typedef mylong1 long_from_module_1;
        typedef mystring2 string_from_module_2;
        typedef mylong2 long_from_module_1_2;
      };
   };
};
```

TTCN EXAMPLE:

```
module identifier1 {
    type long mylong1;
}

module identifier1__identifier2 {
    import from identifier1 all;
    type iso8859string mystring2;
    type identifier1.mylong1 mylong2;
}

module identifier1__identifier2__identifier3 {
    import from identifier1 all;
    import from identifier1__identifier2 all;

    type identifier1.mylong1 long_from_module_1;
    type identifier1__identifier2.mystring2 string_from_module_2;
    type identifier1__identifier2.mylong2 long_from_module_1_2;
};
```

7.2 Importing interface declaration

Interfaces are flattened and all interface definitions are stored in one group. In contrast to interfaces in IDL, groups in TTCN-3 do not create a scope. Therefore, prefixes for all identifiers of type definitions inside of the interface shall be used, which are a combination of the interface name and two underscores as the prefix.

Import of single interface definitions from other modules via the importing group statement is possible. This can be used if inheritance is used in the IDL specification.

For each interface, a procedure-based port type is defined for the test specification. It is associated with signatures translated from attributes and operations of the interface.

An IDL attribute is mapped to two signatures: one for the setting of a value and one for getting it. These signatures have names composed of the prefix (interface name and two underscores), attribute name and the word "Set" (except for "readonly") or "Get" correspondingly.

Since an interface can be used in operation parameters to pass object references, an **address** type is also declared in the data part - the concrete implementation is left to the user. Components are used as collection of interfaces or objects.

IDL EXAMPLE:

```
interface identifier {
    attribute long attributeId ;
    void operationname ( in string param_value ) raises ( ExceptionType ) ;
... other body definitions ...
};
```

TTCN EXAMPLE:

```
group identifierInterface {
    signature identifier__attributeIdGet () return long
    exception ( ... /* and all system exceptions defined in clause 9 */ );
    signature identifier__attributeIdSet (in long identifier__attributeId)
    exception ( ... /* and all system exceptions defined in clause 9 */ );

    signature identifier__operationname ( in iso8859string identifier__param_value )
    exception ( ExceptionType, ... /* and all system exceptions defined in clause 9 */ ) ;

    ...other body definitions ...

    type port identifier procedure { ... }
    type charstring identifierObject;   /* a possible definition for the address type */
    type identifierObject address;
}
```

Interface inheritance is executed by rolling out all inherited elements. Thus, they have to be handled as defined in the interface itself. Multiple inheritance elements have to be inherited only once! As normally an inherited IDL interface uses types defined in the module, usually it is essential to import the complete mapped TTCN-3 module. All inherited elements have to be rolled out directly in the TTCN-3 group for the interface, even if the inheritance is multiple.

Forward references of interfaces are provided by forward referencing the according port of the interface. Local interfaces are treated as normal interfaces. However it is recommend not to use forward references and to move a TTCN-3 definition of the interface (group) to a place where a forward definition is used first time.

7.3 Importing value declaration

In contrast to type **interface**, the IDL type **value** has local operations that are not used outside the object, and are therefore not relevant from the functional testing point of view. However, since the public attributes of **value** instances are used to communicate object states, the IDL **value** type is mapped to the **record** type in TTCN-3.

The example below shows how to map **valuetype** and was used from clause 5.2.5 in [4].

IDL EXAMPLE:

```
valuetype EmployeeRecord {
   // note this is not a CORBA::Object
   // state definition
   private string name;
   private string email;
   private string SSN;

   // initializer
   factory init(
    in string name, in string SSN );
};
```

TTCN EXAMPLE:

```
type record EmployeeRecord {
   iso8859string name,
   iso8859string email,
   iso8859string SSN
}
```


7.4 Importing constant declaration

Constant declarations can be transformed by use of literal (see table B.1) and operator mapping for floating-point and integer values (see table 2).

Table 2: Operators for constant expressions

Operator	IDL	TTCN
Unary floating-point		
Positive	+	+
Negative	-	-
Binary floating-point		
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Division	/	/
Unary integer		
Positive	+	+
Negative	-	-
Bit-complement	~	not4b
Binary integer		
Addition	+	+
Subtraction	-	-
Multiplication	*	*
Division	/	/
Modulo	%	mod
Shift left	<<	<<
Shift right	>>	>>
Bitwise and	&	and4b
Bitwise or	|	or4b
Bitwise xor	^	xor4b