ETSI ES 201 873-8 V4.8.1 (2021-06)

Methods for Testing and Specification (MTS);

The Testing and Test Control Notation version 3;

Part 8: The IDL to TTCN-3 Mapping

**ETSI Standard**

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 - APE 7112B

Association à but non lucratif enregistrée à la

Sous-Préfecture de Grasse (06) N° w061004871

***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>

***Notice of disclaimer & limitation of liability***

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or

other professional standard and applicable regulations.

No recommendation as to products and services or vendors is made or should be implied.

No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.

In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

***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 2021.

All rights reserved.

Contents

Intellectual Property Rights 5

Foreword 5

Modal verbs terminology 5

1 Scope 6

2 References 6

2.1 Normative references 6

2.2 Informative references 6

3 Definition of terms, symbols and abbreviations 7

3.1 Terms 7

3.2 Symbols 7

3.3 Abbreviations 7

4 General considerations 8

4.1 Introduction 8

4.2 Approach 9

4.3 Conformance and compatibility 9

5 Lexical Conventions 9

5.0 General 9

5.1 Comments 9

5.2 Identifiers 9

5.3 Keywords 9

5.4 Literals 9

6 Pre-processing 10

7 Importing from IDL specifications 10

7.0 General 10

7.1 Importing module declaration 10

7.2 Importing interface declaration 11

7.3 Importing value declaration 12

7.4 Importing constant declaration 12

8 Importing type declaration 13

8.0 General 13

8.1 IDL basic types 13

8.1.0 General approach 13

8.1.1 Integer and floating-point types 13

8.1.2 Char and wide char type 13

8.1.3 Boolean type 14

8.1.4 Octet type 14

8.1.5 Any type 14

8.2 Constructed types 14

8.2.0 General approach 14

8.2.1 Struct 14

8.2.2 Discriminated unions 15

8.2.3 Enumerations 16

8.3 Template types 16

8.3.0 General approach 16

8.3.1 Sequence 16

8.3.2 String and wstring 16

8.3.3 Fixed types 17

8.4 Complex declarator 17

8.4.0 General approach 17

8.4.1 Arrays 17

8.4.2 Native types 17

9 Importing exception declaration 17

10 Importing operation declaration 19

11 Importing attribute declaration 20

12 Names and scoping 20

Annex A (informative): Examples 22

A.1 The example 22

A.2 IDL specification 22

A.3 Derived TTCN-3 specification 23

Annex B (informative): Mapping lists 28

B.1 IDL keyword and concept mapping list 28

B.2 Comparison of IDL, ASN.1, TTCN-2 and TTCN-3 data types 29

Annex C (informative): Bibliography 30

History 31

# Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

**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.

# Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Methods for Testing and Specification (MTS).

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](https://portal.etsi.org/Services/editHelp!/Howtostart/ETSIDraftingRules.aspx) (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 JavaBeansTM

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).

**Behavior Types**

**Continuous signals**

**OO**

**Features**

**Advanced Parameteri-**

**zation**

**Advanced Matching**

…

TTCN-3 Packages

**TTCN-3 Core**

**Language**

TTCN-3 User

Other types & values

**JSON types & values**

*The shaded boxes are not defined in the present document*

**Performance**

**and Real Time Testing**

**Configuration and deploy-ment support**

**XML types**

**IDL types**

**ASN.1 types and values**

Figure : 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 : 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 n**ame,

**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 |

IDL EXAMPLE:

**const** **long** number = 017; // 017 == 0xF == 15

**const** **long** size = ( ( number << 3 ) % 0x1F ) & 0123;

TTCN EXAMPLE:

**const** **long** number := "17"O;

**const** **long** size := ( ( number << 3 ) **mod** '1F'H ) **and4b** '0123'O;

# 8 Importing type declaration

## 8.0 General

Type declaration mapping will be shown in the following clauses.

A construct for naming data types and defining new types by using the keyword **typedef** is provided by IDL. This can be done under TTCN-3 via the keyword **type**, too.

To enhance readability and to provide a clear distinction, mapped IDL data types get the prefix IDL and the extension attribute "**variant**" as done in TTCN-3 for type **IDLfixed** (see clause E.2.3.0 in ETSI ES 201 873-1 [1]).

## 8.1 IDL basic types

### 8.1.0 General approach

IDL basic data types are mapped to predefined or useful types in TTCN-3.

### 8.1.1 Integer and floating-point types

Integer and floating-point types are mapped onto the corresponding useful types **short**, **unsignedshort**, **long**, **unsignedlong**, **longlong**, **unsignedlonglong**, **IEEE754float**, **IEEE754double**, and **IEEE754extdouble**.

IDL EXAMPLE:

**const** **long** size = ( ( number << 3 ) % 0x1F ) & 0123;

**const** **float** decimal = 15.7;

TTCN EXAMPLE:

**const** **long** size := ( ( number << 3 ) **mod** '1F'H ) **and4b** '0123'O;

**const** **IEEE754float** decimal := 15.7;

### 8.1.2 Char and wide char type

The IDL **char** and **wide char** type represent a single and wide character. They are mapped to the self defined type **iso8859char** and type **uchar**.

IDL EXAMPLE:

**const** **char** letter = 'ABCD';

**const** **wchar** wideLetter = L'ABCD';

TTCN EXAMPLE:

**type universal charstring** uchar **length**(1)**;**

**type** uchar **iso8859char** (**char** ( 0,0,0,0 ) .. **char** ( 0,0,0,255)) **with** { **variant** "8 bit" };

**const** iso8859charletter:= **char** ( 65, 66, 67, 68 );

**const** ucharwideLetter:= **char** ( 65, 66, 67, 68 );

### 8.1.3 Boolean type

The IDL **boolean** type is equivalent to the TTCN-3 **boolean** type.

IDL EXAMPLE:

const boolean isValid = TRUE;

TTCN EXAMPLE:

**const boolean** isValid = **true;**

### 8.1.4 Octet type

**Octet** cannot be mapped onto an integer type because it has the special feature that it will not change its internal ordering if transferred between different system architectures. To represent it **octet** is mapped to **octetstring**.

IDL EXAMPLE:

**const** **octet** data = 0x55;

TTCN EXAMPLE:

**const** **octetstring** data = '55'H

### 8.1.5 Any type

The IDL **any** type is mapped onto **anytype** in TTCN-3 which was especially introduced for this mapping.

IDL EXAMPLE:

**typedef** **any** AllTypes;

TTCN EXAMPLE:

**type** **anytype** AllTypes;

## 8.2 Constructed types

### 8.2.0 General approach

IDL provides the three constructed types **struct,** **union,** and **enum**. Recursive construction of types is only permitted with the **sequence** template.

### 8.2.1 Struct

**struct** is used to collect ordered data in one place where it is mapped onto **record** in TTCN-3.

IDL EXAMPLE:

**typedef struct** NC {

**string** id;

**string** kind;

} NameComponent;

TTCN EXAMPLE:

**type** **record** NameComponent {

**iso8859string** id,

**iso8859string** kind

}

### 8.2.2 Discriminated unions

In IDL, unions are discriminated to determine the actual type. Therefore, a **record** type is used, which contains two members. The first one stores the discriminator information using an enumeration type. The second member is a TTCN‑3 **union** type which members are defined according to the specified IDL union members.

In addition, two types are defined to express the link between discriminator's type and union's type: a type to reflect the discriminating type of a union and an enumeration to distinguish the discriminated cases. Using the information provided by these type definitions, the marshalling/unmarshalling for discriminated unions is possible in an unambiguous manner: to encode or decode a union value, the value of the kind field to resolve the corresponding chosen option and calculate then the real value for the discriminator by resolving this value in the discriminator enumeration shall be used.

IDL EXAMPLE 1:

**union** MyUnion **switch**( **long** ) {

**case** 0 : **boolean** b;

**case** 1 : **char** c;

**case** 2 : **octet** o;

**case** 3 : **short** s; };

TTCN EXAMPLE 1:

**type long** MyUnion\_\_Switch;

**type** **union** MyUnionType {

**boolean** b,

**iso8859string** c,

**octetstring** o,

**short** s }

**type** **enumerated** MyUnionEnumType {

boolean\_b, iso8859string\_c, octetstring\_o, short\_s

}

**type** **record** MyUnion {

MyUnionEnumType kind,

MyUnionType value

}

IDL EXAMPLE 2:

**Enum** MyDiscr {

BOOLEAN\_DISCR,

CHAR\_DISCR,

OCTET\_DISCR,

SEQ\_DISCR,

SHORT\_DISCR

};

**union** MyUnion **switch**( MyDiscr) {

**case** BOOLEAN\_DISCR : **boolean** b;

**case** SHORT\_DISCR : **short** s;

};

TTCN EXAMPLE 2:

**type** **enumerated** MyDiscr {

BOOLEAN\_DISCR, CHAR\_DISCR, OCTET\_DISCR, SEQ\_DISCR, SHORT\_DISCR

}

**type** MyDiscrMyUnion**\_\_Switch**;

**type enumerated** MyUnion**\_\_CasesType** {

case\_BOOLEAN\_DISCR,

case\_SHORT\_DISCR

}

**type** **union** MyUnionType {

**boolean** b,

**short** s

}

**type** **enumerated** MyUnionEnumType {

boolean\_b,

short\_s

}

**type** **record** MyUnion {

MyUnionEnumType kind\_,

MyUnionType value\_

}

### 8.2.3 Enumerations

Enumerations are equally defined in IDL and TTCN-3.

IDL EXAMPLE:

**enum** NotFoundReason {

missing\_node,

not\_context,

not\_object };

TTCN EXAMPLE:

**type** **enumerated** NotFoundReason {

missing\_node,

not\_context,

not\_object }

## 8.3 Template types

### 8.3.0 General approach

IDL supports the template types **sequence**, **string**, **wide string** and **fixed** type.

### 8.3.1 Sequence

IDL **sequence** is mapped to **record of** in TTCN-3 to maintain order and to allow unbounded sequences.

IDL EXAMPLE 1:

**typedef** **sequence**<NameComponent> Name;

TTCN EXAMPLE 1:

**type** **record** **of** NameComponent Name;

IDL sequences with a specified maximum size are mapped to **record of** with limited number of elements to maintain order and restrict the maximum number of elements.

IDL EXAMPLE 2:

**typedef** **sequence**<NameComponent, maximum\_size> Name;

TTCN EXAMPLE 2:

**type** **record** **length** (0, maximum\_size-1) **of** NameComponent Name;

### 8.3.2 String and wstring

**string** and **wstring** types are sequences of **char** and **wchar**. Therefore, **string** and **wstring** are mapped to the useful type **iso8859string** and **universal** **charstring**.

IDL EXAMPLE:

**const** **string** name = "My String";

**const** **wstring** wideName = L"My String";

TTCN EXAMPLE:

**const** **iso8859string** name := "My String";

**const** **universal** **charstring** wideName := "My String";

### 8.3.3 Fixed types

The **fixed** type represents a fixed-point decimal number. It is mapped to the corresponding useful type **IDLfixed** in TTCN-3 (see clause E.2.3.0 in ETSI ES 201 873-1 [1]).

IDL EXAMPLE:

**typedef** **fixed**<12,7> myFix;

TTCN EXAMPLE:

**template IDLfixed myFixTemplate := { 12, 7, ? };** // e.g. in module definition part

**var** **IDLfixed** myFix := { 12, 7, "12345.1234567" }; // e.g. in module control part

## 8.4 Complex declarator

### 8.4.0 General approach

The last kind of type declarators are the complex **array** and **native** types.

### 8.4.1 Arrays

IDL **array** is equal to the TTCN-3 **array** type.

IDL EXAMPLE:

**typedef** **long** NumberList[100];

TTCN EXAMPLE:

**type** **long** NumberList[100];

### 8.4.2 Native types

Native types are used to allow implementation of dependent types. TTCN-3 provides the type **address** to address entities inside a SUT. Hence, **address** can be used for mapping of type **native** and concrete implementation is left to the user.

IDL EXAMPLE:

**typedef native** MyNativeVariable;

TTCN EXAMPLE:

**type** MyNativeVariable address;

# 9 Importing exception declaration

In IDL, exceptions are used in conjunction with operations to handle exceptional conditions during an operation call. Thus, a special struct-like **exception** type is provided which has to be associated with each operation that can trigger this exception. TTCN-3 also supports the use of exceptions with procedure calls by binding it to signature definitions. However, it provides no special **exception** type. Hence, exceptions are defined by using type **record**.

A definition of an **exception** is shown in the following example. The use of exception binding in signature definitions and exception catching is shown in the context of operation declaration.

IDL EXAMPLE:

**exception** NotFoundException {

NotFoundReason why;

Name rest\_of\_name; };

TTCN EXAMPLE:

// definition of an exception type

**type** **record** NotFoundException {

NotFoundReason why,

Name rest\_of\_name }

// definition of a template for the

// defined exception type

**template** NotFoundException

NotFoundExceptionTemplate ( NotFoundReason reason, Name name ) := {

why := reason,

rest\_of\_name := name }

In addition to user defined exceptions, there are CORBA **system exceptions** defined in chapter 4 in [4]. In order to make them available for use in TTCN-3, the following definitions are to be used:

// CORBA system exceptions

type record UNKNOWN{} // the unknown type record

type record BAD\_PARAM{} // an invalid parameter was passed

type record NO\_MEMORY{} // dynamic memory allocation failure

type record IMP\_LIMIT{} // violated implementation limit

type record COMM\_FAILURE{} // communication failure

type record INV\_OBJREF{} // invalid object reference

type record NO\_PERMISSION{} // no permission for attempted op.

type record INTERNAL{} // ORB internal error

type record MARSHAL{} // error marshaling param/result

type record INITIALIZE{} // ORB initialization failure

type record NO\_IMPLEMENT{} // operation implementation unavailable

type record BAD\_TYPECODE{} // bad typecode

type record BAD\_OPERATION{} // invalid operation

type record NO\_RESOURCES{} // insufficient resources for req.

type record NO\_RESPONSE{} // response to req. not yet available

type record PERSIST\_STORE{} // persistent storage failure

type record BAD\_INV\_ORDER{} // routine invocations out of order

type record TRANSIENT{} // transient failure - reissue request

type record FREE\_MEM{} // cannot free memory

type record INV\_IDENT{} // invalid identifier syntax

type record INV\_FLAG{} // invalid flag was specified

type record INTF\_REPOS{} // error accessing interface repository

type record BAD\_CONTEXT{} // error processing context object

type record OBJ\_ADAPTER{} // failure detected by object adapter

type record DATA\_CONVERSION{} // data conversion error

type record OBJECT\_NOT\_EXIST{} // non-existent object, delete reference

type record TRANSACTION\_REQUIRED{} // transaction required

type record TRANSACTION\_ROLLEDBACK{}// transaction rolled back

type record INVALID\_TRANSACTION{} // invalid transaction

type record INV\_POLICY{} // invalid policy

type record CODESET\_INCOMPATIBLE{} // incompatible code set

type record REBIND{} // rebind needed

type record TIMEOUT{} // operation timed out

type record TRANSACTION\_UNAVAILABLE{} // no transaction

type record TRANSACTION\_MODE{} // invalid transaction mode

type record BAD\_QOS{} // bad quality of service

type record INVALID\_ACTIVITY{} // bad quality of service

type record ACTIVITY\_COMPLETED{} // bad quality of service

type record ACTIVITY\_REQUIRED{} // bad quality of service

type union SYSTEM\_EXCEPTION {

UNKNOWN unknown,

BAD\_PARAM bAD\_PARAM,

NO\_MEMORY nO\_MEMORY,

IMP\_LIMIT iMP\_LIMIT,

COMM\_FAILURE cOMM\_FAILURE,

INV\_OBJREF iNV\_OBJREF,

NO\_PERMISSION nO\_PERMISSION,

INTERNAL iNTERNAL,

MARSHAL mARSHAL,

INITIALIZE iNITIALIZE,

NO\_IMPLEMENT nO\_IMPLEMENT,

BAD\_TYPECODE bAD\_TYPECODE,

BAD\_OPERATION bAD\_OPERATION,

NO\_RESOURCES nO\_RESOURCES,

NO\_RESPONSE nO\_RESPONSE,

PERSIST\_STORE pERSIST\_STORE,

BAD\_INV\_ORDER bAD\_INV\_ORDER,

TRANSIENT tRANSIENT,

FREE\_MEM fREE\_MEM,

INV\_IDENT iNV\_IDENT,

INV\_FLAG iNV\_FLAG,

INTF\_REPOS iNTF\_REPOS,

BAD\_CONTEXT bAD\_CONTEXT,

OBJ\_ADAPTER oBJ\_ADAPTER,

DATA\_CONVERSION dATA\_CONVERSION,

OBJECT\_NOT\_EXIST oBJECT\_NOT\_EXIST,

TRANSACTION\_REQUIRED tRANSACTION\_REQUIRED,

TRANSACTION\_ROLLEDBACK tRANSACTION\_ROLLEDBACK,

INVALID\_TRANSACTION iNVALID\_TRANSACTION,

INV\_POLICY iNV\_POLICY,

CODESET\_INCOMPATIBLE cODESET\_INCOMPATIBLE,

REBIND rEBIND,

TIMEOUT tIMEOUT,

TRANSACTION\_UNAVAILABLE tRANSACTION\_UNAVAILABLE,

TRANSACTION\_MODE tRANSACTION\_MODE,

BAD\_QOS bAD\_QOS,

INVALID\_ACTIVITY iNVALID\_ACTIVITY,

ACTIVITY\_COMPLETED aCTIVITY\_COMPLETED,

ACTIVITY\_REQUIRED aCTIVITY\_REQUIRED

}

# 10 Importing operation declaration

Apart from attributes, operations are the main part of interface definitions in IDL and are used, for instance, in the CORBA scheme as procedures which can be called by clients. Procedure calls in general are supported by TTCN-3 by means of synchronous communication operations which are used in combination with ports.

IDL supports an optional **oneway** attribute for operations which implies best-effort invocation semantics without a guarantee of delivery but with a most-once invocation semantics. Message or procedure-based ports can be used for **oneway** procedures because both would be a valid mapping based upon IDL. However, the use of procedure-based ports for **oneway** procedures is recommended because the IDL specification does not guarantee that **oneway** calls are non-blocking or asynchronous. Furthermore, CORBA implements **oneway** procedures by synchronous communication, too. Use of non-blocking or blocking procedures for **oneway** operations is left to the user. Mapped **oneway** operations acquire an additional **variant** attribute (see example).

The parameter attributes **in**, **inout** and **out** describe the transmission direction of parameters and can be mapped directly to the communication parameter attributes in TTCN-3 because they have the exact same semantics.

A **raise** expression specifies all user-defined exceptions which can be thrown by an operation. In addition, all CORBA system exceptions as defined in clause 9 can be raised. The raise expression can be mapped directly to TTCN-3 because it can be indicated by the procedure signature definition by specifying the list of exceptions.

A **context** expression provides access to local properties of the called operation. These properties consist of a name and a string value. The **context** expression can be matched by redefining the operation with the context parameters included in the operation parameters (see clause 4.6 in [4]). The additional parameter shall be of type **array** containing a type **record** for each context parameter. The **record** itself contains two variables of type **string** for the context name and value.

IDL EXAMPLE:

// NotFoundException is defined clause "Exception declaration"

**string** remoteProc1( **in** long Par11, **out** long Par12, **inout** string name1 )

**raises**( NotFoundException )

**context**( "MyContext1" );

// oneway procedure: no return value and no inout or out allowed!!!

**oneway** **void** remoteProc2( **in** long Par21, **in** long Par22, **in** string name2 );

TTCN EXAMPLE:

// only operation definition

**type** **record** IDLContextElement {

**iso8859string** name,

**iso8859string** value\_

}

**type** **record** **of** IDLContextElement IDLContext;

**signature** RemoteProcSignature1(

**in** **long** Par11, **out** **long** Par12,

**inout** **charstring** name1, **in** IDLContext context )

**return** **iso8859string**

**exception**( // user-defined exception

NotFoundException,

SYSTEM\_EXCEPTION

);

**signature** RemoteProcSignature2(

**in** **long** Par21, **in** **long** Par22,

**in** **iso8859string** name2 )

exception ( SYSTEM\_EXCEPTION )

**with** { **variant** "IDL:oneway FORMAL/01-12-01 v.2.6" };

**type** **port** RemoteProcPort **procedure** {

**out** RemoteProcSignature1;

**out** RemoteProcSignature2

}

**type** **component** CorbaSystem {

**port** RemoteProcPort PCO

}

# 11 Importing attribute declaration

An **attribute** is like a set- and get-operation pair to access a value. If an attribute is marked as **readonly**, only the get‑operation is used. Therefore, attribute mapping can be done by the operation mapping.

# 12 Names and scoping

The name definition scheme of IDL does not collide with the name definition in TTCN-3. Scoping is more restrictive in IDL than in TTCN-3, where the IDL scoping rules have to be mapped appropriately to allow seamless mapping. IDL uses nested scopes for modules, interfaces, structures, unions, operations and exceptions and identifiers are scoped in types, constants, enumeration values, exceptions, interfaces, attributes and operations. The hierarchical scopes in TTCN‑3 are **module**, control part of module, **function**, **testcase** and statement blocks within control part of **module**, **function** and **testcase**.

Furthermore, TTCN-3 supports no overloading of identifiers so that no identifier name can be used more than once in a scope hierarchy. However, IDL allows redefinition of self defined types if defined inside a **module**, **interface** or **valuetype**. Hence, identifiers have to be mapped by using their path name including all **interface** and **valuetype** names as designated in IDL and TTCN-3. The use of module names is not necessary because they are reflected by the TTCN‑3 module structure. An underscore is used as a separator and existing underscores are doubled.

Several new identifiers are generated during transformation of IDL types by adding to the original IDL type identifier suffixes like: "Type", "Enum", "Object", "Interface", etc. This approach and the use of TTCN-3 keywords in IDL modules can cause a name clashes, which are to be resolved by a suffix "\_":

NOTE: ETSI ES 201 873‑1 [1] clause A.1.5 table A.2 defines the keywords of the core language. However, TTCN‑3 language extensions (see [i.5] to [i.9], but other extensions may also be published after the publication of the present document) may define additional keywords and rules for handling those keywords in TTCN-3 modules requiring the given extension.

IDL EXAMPLE:

**interface** identifier {

... body definitions ...

};

//an example of the identifier, which can cause a name clash

**typedef** **long** identifierObject;

TTCN EXAMPLE:

**group** identifierInterface {

... body definitions ...

**type port** identifier **procedure** { ... }

//the suffix '\_' is used only where necessary

//to resolve the name clash

**type** **charstring** identifierObject\_;

**type** identifierObject\_ **address**;

}

**type** **long** identifierObject;

To indicate the special treatment of TTCN-3 statements derived from IDL, TTCN-3 provides a new mechanism to attach attributes to language elements. The use of attributes makes code more readable and requires no special naming scheme. Therefore, the **variant** attribute can be used to indicate the derivation of types from IDL and the special treatment for encoding by the test system. This is used in TTCN-3 for the **IDLfixed** useful type:

**type record** **IDLfixed** {  
 **unsignedshort** digits,  
 **short** scale,  
 **charstring** value\_  
 }  
 **with** { **variant** "IDL:fixed FORMAL/01-12-01 v.2.6" };

Names of new types which are specially defined for the IDL mapping and their use in conjunction with IDL shall always begin with the word IDL to provide better distinction.

Annex A (informative):  
Examples

# A.1 The example

The following example shows how a mapping would look like if a complete IDL and TTCN-3 specification, including a test case, is used. It is only intended to give an impression of how the different elements have to be mapped and used in TTCN-3.

Some parts are used from the CORBA standard like the Naming Service with slight modifications to cover more IDL elements.

# A.2 IDL specification

**module** ttcnExample

{

// \*\*\*\*\*\*\*\*\*\*\*

// Basic Types

// \*\*\*\*\*\*\*\*\*\*\*

**const** **long** number = 017; // 017 == 0xF == 15

**const** **long** size = ( ( number << 3 ) % 0x1F ) & 0123;

**const** **float** decimal = 15.7;

**const** **char** letter = 'A';

**const** **wchar** wideLetter = L'A';

**const** **boolean** isValid = **TRUE**;

**const** **octet** anOctet = 0x55; // limited to 8 bit

**const** **string** myName = "my name";

**const** **wstring** wideMyName = L"my name";

**typedef** **string** MyString;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructed Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**typedef** **struct** NC {

MyString id;

MyString kind;

} NameComponent;

**union** MyUnion **switch**( **long** ) {

**case** 0 : **boolean** b;

**case** 1 : **char** c;

**case** 2 : **octet** o;

**case** 3 : **short** s;

};

**enum** NotFoundReason { missing\_node,

not\_context,

not\_object };

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Template Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*

**typedef** **sequence** <NameComponent> Name;

**typedef** **sequence** <NameComponent> Key;

**typedef** **fixed**<12,7> Fix;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Complex Declarator

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**typedef** **long** NumberList[100];

**native** MyNativeVariable;

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Valuetype Definition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**valuetype** StringValue **string**;

**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);

};

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Interface Definition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**interface** NamingContext {

**attribute** **string** object\_type;

**readonly** **attribute** Key external\_form\_id;

**exception** NotFoundException {

NotFoundReason why;

Name rest\_of\_name;

};

MyString bind( **in** Name n, **inout** Object obj, **out** Object myObj )

**raises**( NotFoundException ) **context** ( "Hostname" );

**oneway** **void** rebind( **in** Name n, **in** Object obj );

}; // end of interface NamingContext

}; // end of module ttcnExample

# A.3 Derived TTCN-3 specification

**module** ttcnExample {

**import from** IDLaux **all;**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Mapping of the IDL Specification

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Mapping of Basic Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**const long number := oct2int('17'O) ;**

**const long size := oct2int(int2oct(oct2int(int2oct(number,4)<<3) mod hex2int('1F'H),4) and4b '0123'O);**

**const IEEE754float decimal := 15.7;**

**type universal charstring** uchar **length**(1)**;**

**type uchar iso8859char (char ( 0,0,0,0 ) .. char ( 0,0,0,255))**

**with { variant "8 bit" };**

**const iso8859char letter := "A";**

**const uchar wideLetter := "A";**

**const boolean isValid := true;**

**const octetstring anOctet := hex2oct('55'H);**

**const iso8859string myName := "my name";**

**const universal charstring wideMyName := "my name";**

**type iso8859string MyString;**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Constructed Types

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\*\*\*\*\*

// Struct

// \*\*\*\*\*\*

**type record NameComponent {**

**MyString id,**

**MyString kind**

**};**

// \*\*\*\*\*

// Union

// \*\*\*\*\*

**type union MyUnion {**

**boolean b,**

**iso8859char c,**

**octetstring o,**

**short s**

**};**

// \*\*\*\*\*\*\*\*\*\*\*

// Enumeration

// \*\*\*\*\*\*\*\*\*\*\*

**type enumerated NotFoundReason {**

**missing\_node,**

**not\_context,**

**not\_object**

**}**

// \*\*\*\*\*\*\*\*

// Sequence

// \*\*\*\*\*\*\*\*

**type record of NameComponent Name;**

**type record of NameComponent Key;**

//\*\*\*\*\*\*

// Fixed

// \*\*\*\*\*

// see also using of fixed in testcase below

**template IDLfixed fixTemplate := { 12, 7, ? };**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Complex Declarator

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**type long numberList[100];**

// see using of native in testcase below

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Valuetype Definition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**type iso8859string StringValue;**

**type record EmployeeRecord {**

**iso8859string name,**

**iso8859string email,**

**iso8859string SSN**

**};**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Interface Definition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**type record IDLContextElement {**

**iso8859string name,**

**iso8859string value\_**

**}**

**type record of IDLContextElement IDLContext;**

**group NamingContextInterface {**

**type** **charstring** **NamingContextObject**;

**type NamingContextObject address;**

// attribute object\_type

**signature NamingContext\_\_object\_typeGet () return iso8859string**

**exception ( SYSTEM\_EXCEPTION );**

**signature NamingContext\_\_object\_typeSet ( in iso8859string NamingContext\_\_object\_type )**

**exception ( SYSTEM\_EXCEPTION );**

**template NamingContext\_\_object\_typeSet ObjectTypeSetSignatureTemplate := {**

**object\_type := "my object type"**

**}**

//

// attribute external\_from\_id

//

**signature NamingContext\_\_external\_form\_idGet() return Key**

**exception ( SYSTEM\_EXCEPTION );**

// exception notFoundException

**type record NamingContext\_\_NotFoundException {**

**NotFoundReason why,**

**Name rest\_of\_name**

**}**

**template NamingContext\_\_NotFoundException**

**NamingContext\_\_NotFoundExceptionTemplate ( NotFoundReason reason, Name name ) := {**

**why := reason,**

**rest\_of\_name := name**

**}**

//

// bind procedure

//

**signature NamingContext\_\_BindSignature**

**( in Name n, inout address obj, inout address myObj,**

**in IDLContext context ) return MyString**

**exception( NamingContext\_\_NotFoundException,**

**SYSTEM\_EXCEPTION );**

**template NamingContext\_\_BindSignature**

**NamingContext\_\_BindTemplate ( charstring object, IDLContext con ) := {**

**n := { {"name", ""} },**

**obj := object,**

**myObj := ?,**

**context := con**

**}**

//

// rebind procedure

//

**signature NamingContext\_\_RebindSignature( in Name n, in address obj )**

**exception ( SYSTEM\_EXCEPTION )**

**with { variant "IDL:oneway FORMAL/01-12-01 v.2.6" };**

**template NamingContext\_\_RebindSignature**

**NamingContext\_\_RebindTemplate ( address object ) := {**

**n := { {"name", ""} },**

**obj := object**

**}**

**type port NamingContext procedure {**

**out NamingContext\_\_object\_typeGet;**

**out NamingContext\_\_object\_typeSet;**

**out NamingContext\_\_external\_form\_idGet;**

**out NamingContext\_\_BindSignature;**

**}**

**}**

// component is necessary for test case

**type component CorbaSystemInterface {**

**port NamingContext PCO;**

**}**

// somewhere has main test component MyMTC to be defined

**type component MyMTC {**

**port NamingContext NamingContextPCO;**

**}**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Testcase Definition

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**testcase MyNamingServiceTestCase() runs on MyMTC system CorbaSystemInterface {**

// examples to show how above definitions can be used inside a

// testcase definition

**var CorbaSystemInterface myCorbaSystem := CorbaSystemInterface.create;**

**connect( self:NamingContextPCO, myCorbaSystem:PCO );**

**myCorbaSystem.start;**

//

// Fixed Type

//

**var IDLfixed fix := { 12, 7, "12345.1234567" };**

//

// Native

//

**var address MyNativeVariable;**

//

// Procedure Calls

//

**var MyString myResult1;**

**var Key myResult2;**

**var MyString myResult3;**

**var address object, myObject, resultObject, resultMyObject;**

**var IDLContextElement contextElement := {**

**name := "Hostname",**

**value\_ := "disen"**

**}**

**var IDLContext contextParameter := { contextElement };**

//

// procedure get object\_type

//

**NamingContextPCO.call( ObjectTypeGetSignature )**

**{**

**[] NamingContextPCO.getreply( ObjectTypeGetSignature value \* )**

**-> value myResult1 {}**

**}**

//

// procedure set object\_type

//

**NamingContextPCO.call( ObjectTypeSetSignatureTemplate );**

//

// procedure get external\_from\_id

//

**NamingContextPCO.call( ExternalFormIdGetSignature )**

**{**

**[] NamingContextPCO.getreply( ExternalFormIdGetSignature value \* )**

**-> value MyResult2 {}**

**}**

//

// procedure bind (with template)

//

**NamingContextPCO.call( BindTemplate( object, contextParameter ) )**

**{**

**[] NamingContextPCO.getreply( BindTemplate( \* ) value \* )**

**-> value myResult3**

**param( resultObject, resultMYObject ) sender mySender {}**

**[] NamingContextPCO.catch( BindSignature,**

**NamingContext\_\_NotFoundExceptionTemplate )**

**{**

**setverdict( fail );**

**stop;**

**}**

**}**

//

// procedure bind (without template)

//

**NamingContextPCO.call(**

**BindSignature:{ myName, object, myObject, contextParameter } )**

**{**

**[] NamingContextPCO.getreply( BindSignature:{ -, \*, myObject }**

**value \* ) -> value myResult3 param( resultObject, resultMYObject ) sender mySender {}**

**}**

//

// procedure rebind

//

**NamingContextPCO.call( RebindSignature:{ myName, object} ); // or use a template**

//

// raising an exception

//

// this would be used to raise an exception inside of procedure bind

// if defined by TTCN-3 (if used on server side).

**var NamingContext\_\_NotFoundException myNotFoundException := {**

**why := missing\_node,**

**rest\_of\_name := "noname"**

**}**

**NamingContextPCO.raise( BindSignature, myNotFoundException );**

**}** // end of testcase MyNamingServiceTestCase

**}**

Annex B (informative):  
Mapping lists

# B.1 IDL keyword and concept mapping list

Table B.1 lists the mapping of keywords and concepts of IDL to TTCN-3 keywords or concepts. Literal and operator mapping can be seen in tables B.1 and 2.

Table B.1: Conceptual list of IDL mapping

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IDL | TTCN-3 |  | IDL | TTCN-3 |
| FALSE | false |  | module | module |
| Object | address |  | native | address |
| TRUE | true |  | octet | octetstring |
| abstract | has to be rolled out |  | oneway | operation with variant attribute |
| any | anytype |  | operation | signature for procedure |
| array | array |  | out | out |
| attribute | get (and set) operation |  | raises | exception |
| boolean | boolean |  | readonly | only a get-operation for the attribute |
| char | iso8859char (self defined type) |  | sequence | record of |
| const | const |  | short | short |
| context | additional procedure parameter of type record |  | string | iso8859string |
| enum | enumerated |  | struct | record |
| exception | record |  | typedef | type |
| fixed | IDLfixed |  | union | record, enumerated, union |
| float | IEEE754float |  | unsigned long | unsignedlong |
| double | IEEE754double |  | unsigned long long | unsignedlonglong |
| long double | IEEE754extdouble |  | unsigned short | unsignedshort |
| in | in |  | valuetype | record |
| inout | inout |  | wchar | universal charstring |
| interface | group, port |  | wstring | universal charstring |
| local | --- |  |  |  |
| long | long |  |  |  |
| long long | longlong |  |  |  |

# B.2 Comparison of IDL, ASN.1, TTCN-2 and TTCN-3 data types

Table B.2

|  |  |  |  |
| --- | --- | --- | --- |
| IDL | ASN.1 | TTCN-2 | TTCN-3 |
| Object | ObjectInstance (X.500 Distinguished name) | IA5String | address |
| any | SEQUENCE {typecode, anyValue} | CHOICE | anytype |
| array | SEQUENCE OF (with sizeConstraint subtype) | SEQUENCE SIZE(n) OF | array |
| boolean | BOOLEAN | BOOLEAN | boolean |
| char | GraphicString | GraphicString or IA5String(SIZE(1)) | iso8859char (self defined type) |
| enum | ENUMERATED | ENUMERATED | enumerated |
| exception | SPECIFIC ERRORS | SEQUENCE | record |
| fixed | See note | See note | IDLfixed |
| float | REAL | See note | IEEE754float |
| double | REAL | See note | IEEE754double |
| long double | REAL | See note | IEEE754extdouble |
| long | INTEGER | INTEGER | long |
| long long | INTEGER | INTEGER | longlong |
| native | See note | See note | address |
| octet | OCTET STRING | OCTET STRING (SIZE(1)) | octetstring |
| sequence | SEQUENCE OF (with optional sizeConstraint subtype for IDL bounds) | SEQUENCE OF | record of |
| short | INTEGER | INTEGER | short |
| string | GraphicString | GraphicString | iso8859string |
| struct | SEQUENCE | SEQUENCE | record |
| union, switch, case | CHOICE (with ASN.1 TAGS) | SEQUENCE | record, enumerated, union |
| unsigned long | INTEGER | INTEGER | unsignedlong |
| unsigned long long | INTEGER | INTEGER | unsignedlonglong |
| unsigned short | INTEGER | INTEGER | unsignedshort |
| valuetype | See note | See note | record |
| wchar | See note | GraphicString or BMPString(SIZE(1)) | universal charstring |
| wstring | See note | GraphicString | universal charstring |
| NOTE: Mapping of this type was not considered. | | | |

Annex C (informative):  
Bibliography

M. Ebner, A. Yin, and M. Li (2002): "Definition and Utilization of OMG IDL to TTCN-3 Mappings". In testing of communicating systems XIV - Application to Internet Technologies and Services, ed. I. Schieferdecker, H. König and  
A. Wolisz. IFIP, Kluwer Academic Publishers, pp. 443-458. ISBN 0-7923-7695-1.

M. Ebner (2001): "A Mapping of OMG IDL to TTCN-3". SIIM Technical Report SIIM-TR-A- 01-11, Institute for Telematics, Medical University of Lübeck, Germany. Schriftenreihe der Institute für Informatik/Mathematik.

M. Ebner (2001): "Mapping CORBA IDL to TTCN-3 based on IDL to TTCN-2 mappings". In Proceedings of the 11th GI/ITG Technical Meeting on Formal Description Techniques for Distributed Systems, Bruchsal, Germany,  
21-22, June 2001. International University in Germany.

A. Yin (2001): "Testing Operation-Based Interfaces Exemplified for CORBA with ADL and TTCN-3". Diplomarbeit, Telecommunication Network Group, Faculty of Electrical Engineering and Computer Science, Technical University Berlin, Germany.

A.Yin, I. Schieferdecker and M. Li (2001): "Mapping of IDL to TTCN-3". Technical Report, Fraunhofer Institute for Open Communication Systems (FOKUS), Germany.

ISO/IEC 9646-3: "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular Combined Notation (TTCN)".

ISO/IEC 646: "Information technology - ISO 7-bit coded character set for information interchange".

IEEE 754: "IEEE Standard for Floating-Point Arithmetic".

ISO/IEC 8859-1: "Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No.1".

ETSI ES 202 789: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Extended TRI".

ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Using XML schema with TTCN-3".

ETSI ES 201 873-11: "MTS The Testing and Test Control Notation version 3; Part 11: Using JSON with TTCN-3".

ETSI ES 203 022: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language extension: Advanced Matching".

ETSI ES 203 790: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Object-Oriented Features".

# History

|  |  |  |
| --- | --- | --- |
| **Document history** | | |
| V3.2.1 | February 2007 | Publication |
| V3.3.1 | April 2008 | Publication |
| V4.2.1 | July 2010 | Publication |
| V4.3.1 | June 2011 | Publication |
| V4.4.1 | April 2012 | Publication |
| V4.5.1 | April 2013 | Publication |
| V4.6.1 | June 2015 | Publication |
| V4.7.1 | May 2017 | Publication |
| V4.8.1 | March 2021 | Membership Approval Procedure MV 20210530: 2021-03-31 to 2021-05-31 |
| V4.8.1 | June 2021 | Publication |