INTERNATIONAL STANDARD

ISO/IEC 19516

First edition
2020-02

Information technology — Object management group — Interface definition language (IDL) 4.2

Reference number
ISO/IEC 19516:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by the Object Management Group (OMG) (as the OMG specification for Interface Definition Language (IDL), v4.2) and drafted in accordance with its editorial rules. It was adopted, under the JTC 1 PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

This document is related to:

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1995, Information Technology — Open Distributed Processing — Reference Model: Foundations
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1995, Information Technology — Open Distributed Processing — Reference Model: Architecture
- ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1997, Information Technology — Open Distributed Processing — Interface Definition Language

Apart from this Foreword, the text of this document is identical with that for the OMG specification for Interface Definition Language (IDL), v4.2.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for this standardization and ITU-T Recommendations X.901-904 | ISO/IEC 10746, the Reference Model of Open Distributed Processing (RM-ODP) provides such a framework. It defines an architecture within which support of distribution, interoperability, and portability can be integrated.

RM-ODP Part 2 (ISO RM-ODP Part 2 (ISO/IEC 10746-2) defines the foundational concepts and modeling framework for describing distributed systems. The scopes and objectives of the RM-ODP Part 2 and the UML, while related, are not the same and, in a number of cases, the RM-ODP Part 2 and the UML specification use the same term for concepts which are related but not identical (e.g., interface). Nevertheless, a specification using the Part 2 modeling concepts can be expressed using UML with appropriate extensions (using stereotypes, tags, and constraints).

RM-ODP Part 3 (ISO/IEC 10746-3) specifies a generic architecture of open distributed systems, expressed using the foundational concepts and framework defined in Part 2. Given the relation between UML as a modeling language and Part 3 of the RM-ODP standard, it is easy to show that UML is suitable as a notation for the individual viewpoint specifications defined by the RM-ODP.

This International Standard defines a method for automating the counting of Function Points that is generally consistent with the Function Point Counting Practices Manual, Release 4.3.1 (IFPUG CPM) produced by the International Function Point Users Group (IFPUG). Guidelines in this International Standard may differ from those in the IFPUG CPM at points where subjective judgments have to be replaced by the rules needed for automation. The IFPUG CPM was selected as the anchor for this International Standard because it is the most widely used functional measurement specification with a large supporting infrastructure maintained by a professional organization.

Information technology — Object management group — Interface definition language (IDL) 4.2

1 Scope

1.1 Overview

This International Standard specifies the OMG Interface Definition Language (IDL). IDL is a descriptive language used to define data types and interfaces in a way that is independent of the programming language or operating system/processor platform.

The IDL specifies only the syntax used to define the data types and interfaces. It is normally used in connection with other standards that further define how these types/interfaces are utilized in specific contexts and platforms:

- Separate "language mapping" standards define how the IDL-defined constructs map to specific programming languages, such as, C/C++, Java, C#, etc.
- Separate "serialization" standards define how data objects and method invocations are serialized into a format suitable for network transmission.
- Separate "middleware" standards, such as, DDS or CORBA leverage the IDL to define data-types, services, and interfaces.

The description of IDL grammar uses a syntax notation that is similar to Extended Backus-Naur Format (EBNF).

2 Conformance Criteria

This International Standard defines IDL such that it can be referenced by other standards. It contains no independent conformance points. It is up to the standards that depend on this International Standard to define their own conformance criteria. However, the general organization of the clauses (by means of atomic building blocks and profiles that group them) is intended to ease conformance description and scoping. That means that no standard using IDL 4.0 will be forced to be compliant with IDL constructs that are not relevant in its usage of IDL.

Conformance to this International Standard must follow these rules:

1. Future standards that use IDL shall reference this IDL International Standard or a future revision thereof.
2. Future revisions of current standards that use IDL may reference this IDL International Standard or a future revision thereof.
3. Reference to this International Standard shall result in a selection of building blocks possibly complemented by groups of annotations.

   a. All selected building blocks shall be supported entirely.

   b. Selected annotations shall be either supported as described in 8.2.2 Rules for Using Standardized Annotations, or fully ignored. In the latter case, the IDL-dependent standard shall not define a specific annotation, either with the same name and another meaning or with the same meaning and another name.

3 Normative References

The following referenced documents are indispensable for the application of this International Standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments):

- ISO/IEC 14882:2003, *Information Technology — Programming languages — C++*
- [RFC2119] IETF RFC 2119, "*Key words for use in RFCs to Indicate Requirement Levels*", S. Bradner, March 1997. Available from http://ietf.org/rfc/rfc2119
- [CORBA] Common Object Request Broker Architecture. OMG specification: formal/2012-11-12 (part1), formal/2012-11-14 (part2), formal/2012-11-16 (part3)

The following referenced documents were used as input to this International Standard:

- [DDS-XTypes]. Extensible and Dynamic Topic Types for DDS, Version 1.2. Available from: http://www.omg.org/spec/DDS-XTypes/1.2
- [DDS]. Data Distribution Service, Version 1.4. Available from: http://www.omg.org/spec/DDS/1.4
- [DDS-RPC]. Remote Procedure Call over DDS, Version 1.0. Available from: http://www.omg.org/spec/DDS-RPC/1.0

4 Terms and Definitions

In this International Standard:

- A building block is a consistent set of IDL rules that together form a piece of IDL functionality. Building blocks are atomic, meaning that if selected, they must be totally supported. Building blocks are described in Clause 7, IDL Syntax and Semantics.
- A group of annotations is a consistent set of annotations, expressed in IDL. Groups of annotations are described in Clause 8, Standardized Annotations.
- A profile is a selection of building blocks possibly complemented with groups of annotations that determines a specific IDL usage. Profiles are described in Clause 9, Profiles.

5 Symbols

The following abbreviations are used throughout this International Standard.

Acronym	Meaning
ASCII	American Standard Code for Information Interchange
BIPM	Bureau International des Poids et Mesures
CCM	CORBA Component Model
CORBA	Common Object Request Broker Architecture
DDS	Data Distribution Service
EBNF	Extended Backus Naur Form

Acronym	Meaning
IDL	Interface Definition Language
ISO	International Organization for Standardization
LwCCM	Lightweight CCM
OMG	Object Management Group
ORB	Object Request Broker
XTypes	eXtensible and dynamic topic Types (for DDS)

6 Additional Information

6.1 Acknowledgments

The following companies submitted this International Standard:

— Thales

— RTI

The following companies supported this International Standard:

— Mitre

— Northrop Grumman

— Remedy IT

6.2 History

Historically, IDL was designed to specify CORBA interfaces and subsequently CORBA components. For this reason the IDL standard was embedded in the CORBA documentation. However its expressive power made it very suitable for defining non-CORBA interfaces and data types. Consequently, it was used in the DDS standard and extended to support that usage. In recognition of these new usages, and expected future ones, IDL was separated into its own stand-alone standard, independent of its use by specific middleware technologies.

This International Standard completes the definition of IDL as a separate standard, an effort started with IDL 3.5.

IDL 3.5 gathered in a single standard all the CORBA-dedicated IDL, formerly specified as a collection of clauses within the CORBA 3 standard.

IDL 4.0 extended that corpus with the other source for IDL definitions, namely "Extensible and Dynamic Topic Types for DDS" in order to group all IDL constructs in a single comprehensive standard. It organized the IDL description into modular "Building Blocks" so that different levels of compliance are easier to specify and that future evolutions, if needed, can be made without side-effects on existing IDL usages.

IDL 4.1 improved the definition of the bitset type, added a bitmap type, and resolved some inconsistencies in the grammar.

IDL 4.2 added support for 8-bit integer types, added size-explicit keywords for integer types, enhanced the readability, and reordered the building blocks to follow a logical dependency progression.

7 IDL Syntax and Semantics

7.1 Overview

This clause describes OMG Interface Definition Language (IDL) middleware[1]-agnostic semantics[2] and defines the syntax for IDL grammatical constructs.

OMG IDL is a language that allows unambiguous specification of the interfaces that client objects[3] may use and (server) object implementations provide as well as all needed related constructs such as exceptions and data types. Data types are needed to specify parameters and return value of interfaces' operations. They can be used also as first class constructs.

IDL is a purely descriptive language. This means that actual programs that use these interfaces or create the associated data types cannot be written in IDL, but in a programming language, for which mappings from IDL constructs have been defined. The mapping of IDL constructs to a programming language will depend on the facilities available in that programming language. For example, an IDL exception might be mapped to a structure in a language that has no notion of exceptions, or to an exception in a language that does. The binding of IDL constructs to several programming languages is described in separate standards.

The clause is organized as follows:

- The description of IDL's lexical conventions is presented in 7.2 Lexical Conventions.
- A description of IDL preprocessing is presented in 7.3 Preprocessing
- The grammar itself is presented in 7.4 IDL Grammar
- The scoping rules for identifiers in an IDL standard are described in 7.5.

IDL-specific pragmas may appear anywhere in a standard; the textual location of these pragmas may be semantically constrained by a particular implementation.

A source file containing specifications written in IDL shall have a "**.idl**" extension.

The description of IDL grammar uses a syntax notation that is similar to Extended Backus-Naur Format (EBNF). However, to allow composition of specific parts of the description, while avoiding redundancy; a new operator (**::+**) has been added. This operator allows adding alternatives to an existing definition. For example, assuming the rule **x ::= y**, the rule **x ::+ z** shall be interpreted as **x ::= y | z**.

Table 7.1 lists the symbols used in this EBNF format and their meaning.

[1] In this document the word *middleware* refers to any piece of software that will make use of IDL-derived artifacts. CORBA and DDS implementations are examples of middleware. The word *compiler* refers to any piece of software that produces these IDL-derived artifacts based on an IDL specification.

[2] I.e., abstract semantics that is applicable to all IDL usages. When needed, middleware-specific interpretations of that abstract semantics will be given afterwards in dedicated clauses.

[3] Accordingly, *client objects* should be understood here as abstract clients, i.e., entities invoking operations provided by object implementations, regardless of the means used to perform this invocation or even whether those implementations are co-located or remotely accessible.

Table 7.1 — IDL EBNF

Symbol	Meaning
`::=`	Is defined to be *(left part of the rule is defined to be right part of the rule)*
`|`	Alternatively
`::+`	Is added as alternative *(left part of the rule is completed with right part of the rule as a new alternative)*
`<text>`	Nonterminal
`"text"`	Literal
`*`	The preceding syntactic unit can be repeated zero or more times
`+`	The preceding syntactic unit must be repeated at least once
`{}`	The enclosed syntactic units are grouped as a single syntactic unit
`[]`	The enclosed syntactic unit is optional – may occur zero or one time

7.2 Lexical Conventions

This sub clause[4] presents the lexical conventions of IDL. It defines tokens in an IDL specification and describes comments, identifiers, keywords, and literals - integer, character, and floating point constants and string literals.

An IDL specification logically consists of one or more files. A file is conceptually translated in several phases.

The first phase is preprocessing, which performs file inclusion and macro substitution. Preprocessing is controlled by directives introduced by lines having # as the first character other than white space. The result of preprocessing is a sequence of tokens. Such a sequence of tokens, that is, a file after preprocessing, is called a translation unit.

IDL uses the ASCII character set, except for string literals and character literals, which use the ISO Latin-1 (8859-1) character set. The ISO Latin-1 character set is divided into alphabetic characters (letters) digits, graphic characters, the space (blank) character, and formatting characters. Table 7.2 shows the ISO Latin-1 alphabetic characters; upper and lower case equivalences are paired. The ASCII alphabetic characters are shown in the left-hand column of Table 7.2.

Table 7.2 — Characters

Char.	Description	Char.	Description
Aa	Upper/Lower-case A	**Àà**	Upper/Lower-case A with grave accent
Bb	Upper/Lower-case B	**Áá**	Upper/Lower-case A with acute accent
Cc	Upper/Lower-case C	**Ââ**	Upper/Lower-case A with circumflex accent
Dd	Upper/Lower-case D	**Ãã**	Upper/Lower-case A with tilde
Ee	Upper/Lower-case E	**Ää**	Upper/Lower-case A with dieresis
Ff	Upper/Lower-case F	**Åå**	Upper/Lower-case A with ring above
Gg	Upper/Lower-case G	**Ææ**	Upper/Lower-case dipthong A with E
Hh	Upper/Lower-case H	**Çç**	Upper/Lower-case C with cedilla
Ii	Upper/Lower-case I	**Èè**	Upper/Lower-case E with grave accent

[4] This sub clause is an adaptation of The Annotated C++ Reference Manual, Clause 2; it differs in the list of legal keywords and punctuation.

Char.	Description	Char.	Description
Jj	Upper/Lower-case J	Éé	Upper/Lower-case E with acute accent
Kk	Upper/Lower-case K	Êê	Upper/Lower-case E with circumflex accent
Ll	Upper/Lower-case L	Ëë	Upper/Lower-case E with dieresis
Mm	Upper/Lower-case M	Ìì	Upper/Lower-case I with grave accent
Nn	Upper/Lower-case N	Íí	Upper/Lower-case I with acute accent
Oo	Upper/Lower-case O	Îî	Upper/Lower-case I with circumflex accent
Pp	Upper/Lower-case P	Ïï	Upper/Lower-case I with dieresis
Qq	Upper/Lower-case Q	Ññ	Upper/Lower-case N with tilde
Rr	Upper/Lower-case R	Òò	Upper/Lower-case O with grave accent
Ss	Upper/Lower-case S	Óó	Upper/Lower-case O with acute accent
Tt	Upper/Lower-case T	Ôô	Upper/Lower-case O with circumflex accent
Uu	Upper/Lower-case U	Õõ	Upper/Lower-case O with tilde
Vv	Upper/Lower-case V	Öö	Upper/Lower-case O with dieresis
Ww	Upper/Lower-case W	Øø	Upper/Lower-case O with oblique stroke
Xx	Upper/Lower-case X	Ùù	Upper/Lower-case U with grave accent
Yy	Upper/Lower-case Y	Úú	Upper/Lower-case U with acute accent
Zz	Upper/Lower-case Z	Ûû	Upper/Lower-case U with circumflex accent
		Üü	Upper/Lower-case U with dieresis
		ß	Lower-case German sharp S
		ÿ	Lower-case Y with dieresis

Table 7.3 lists the decimal digit characters.

Table 7.3 — Decimal Digits

0	1	2	3	4	5	6	7	8	9

Table 7.4 shows the graphic characters.

Table 7.4 — Graphic Characters

Char.	Description	Char.	Description
!	exclamation point	¡	inverted exclamation mark
"	double quote	¢	cent sign
#	number sign	£	pound sign
$	dollar sign	¤	currency sign
%	percent sign	¥	yen sign
&	ampersand	¦	broken bar

Char.	Description	Char.	Description
'	apostrophe	§	section/paragraph sign
(	left parenthesis	¨	dieresis
)	right parenthesis	©	copyright sign
*	asterisk	ª	feminine ordinal indicator
+	plus sign	«	left angle quotation mark
,	comma	¬	not sign
-	hyphen, minus sign		soft hyphen
.	period, full stop	®	registered trade mark sign
/	solidus	¯	macron
:	colon	°	ring above, degree sign
;	semicolon	±	plus-minus sign
<	less-than sign	²	superscript two
=	equals sign	³	superscript three
>	greater-than sign	´	acute
?	question mark	µ	micro
@	commercial at	¶	pilcrow
[	left square bracket	·	middle dot
\	reverse solidus	¸	cedilla
]	right square bracket	¹	superscript one
^	circumflex	º	masculine ordinal indicator
_	low line, underscore	»	right angle quotation mark
`	grave	¼	vulgar fraction 1/4
{	left curly bracket	½	vulgar fraction 1/2
|	vertical line	¾	vulgar fraction 3/4
}	right curly bracket	¿	inverted question mark
~	tilde	×	multiplication sign
		÷	division sign

The formatting characters are shown in Table 7.5.

Table 7.5 — Formatting Characters

Description	Abbreviation	ISO 646 Octal Value
alert	BEL	007
backspace	BS	010
horizontal tab	HT	011
newline	NL, LF	012
vertical tab	VT	013
form feed	FF	014
carriage return	CR	015

7.2.1 Tokens

There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators.

Blanks, horizontal and vertical tabs, newlines, form feeds, and comments (collective, "white space") as described below are ignored except as they serve to separate tokens. Some white space is required to separate otherwise adjacent identifiers, keywords, and constants.

If the input stream has been parsed into tokens up to a given character, the next token is taken to be the longest string of characters that could possibly constitute a token.

7.2.2 Comments

The characters /* start a comment, which terminates with the characters */. These comments do not nest.

The characters // start a comment, which terminates at the end of the line on which they occur.

The comment characters //, /*, and */ have no special meaning within a // comment and are treated just like other characters. Similarly, the comment characters // and /* have no special meaning within a /* comment.

Comments may contain alphabetic, digit, graphic, space, horizontal tab, vertical tab, form feed, and newline characters.

7.2.3 Identifiers

An identifier is an arbitrarily long sequence of ASCII alphabetic, digit, and underscore (_) characters. The first character must be an ASCII alphabetic character. All characters are significant.

IDL identifiers are case insensitive. However, all references to a definition must use the same case as the defining occurrence. This allows natural mappings to case-sensitive languages.

7.2.3.1 Collision Rules

When comparing two identifiers to see if they collide:

- Upper- and lower-case letters are treated as the same letter. Table 7.2 defines the equivalence mapping of upper- and lower-case letters.
- All characters are significant.

Identifiers that differ only in case collide, and will yield a compilation error under certain circumstances. An identifier for a given definition must be spelled identically (e.g., with respect to case) throughout a specification.

There is only one namespace for IDL identifiers in each scope. Using the same identifier for a constant and an interface, for example, produces a compilation error.

EXAMPLE

```
module M {
      typedef long Foo;
      const long thing = 1;
      interface thing {                         // Error: reuse of identifier thing
           void doit (
                      in Foo foo                // Error: Foo and foo collide…
                      );                        //          … and refer to different things
           readonly attribute long Attribute;  // Error: Attribute collides with keyword…
                                                //           … attribute
           };
      };
```

7.2.3.2 Escaped Identifiers

As all languages, IDL uses some reserved words called *keywords* (see 7.2.4).

As IDL evolves, new keywords that are added to the IDL language may inadvertently collide with identifiers used in existing IDL and programs that use that IDL. Fixing these collisions will require not only the IDL to be modified, but programming language code that depends upon that IDL will have to change as well. The language mapping rules for the renamed IDL identifiers will cause the mapped identifier names (e.g., method names) to be changed.

To minimize the amount of work, users may lexically "escape" identifiers by prepending an underscore (_) to an identifier. *This is a purely lexical convention that ONLY turns off keyword checking*. The resulting identifier follows all the other rules for identifier processing. For example, the identifier **_AnIdentifier** is treated as if it were **AnIdentifier**.

EXAMPLE


```
module M {
           interface thing {
                      attribute boolean abstract;      // Error: abstract collides with keyword abstract
                      attribute boolean _abstract;     // OK: abstract is an identifier
                      };
           };
```

NOTE To avoid unnecessary confusion for readers of IDL, it is recommended that IDL specifications only use the escaped form of identifiers when the non-escaped form clashes with a newly introduced IDL keyword. It is also recommended that interface designers avoid defining new identifiers that are known to require escaping. Escaped literals are only recommended for IDL that expresses legacy items, or for IDL that is mechanically generated.

7.2.4 Keywords

The identifiers listed in Table 7.6 are reserved for use as keywords and may not be used for another purpose, unless escaped with a leading underscore.