INTERNATIONAL STANDARD

ISO/IEC 14750

First edition
1999-03-15

Information technology — Open Distributed Processing — Interface Definition Language

Technologies de l'information — Traitement distribué ouvert — Langage de définition d'interface

Reference number
ISO/IEC 14750:1999(E)

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 14750 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 33, *Distributed application services*, in collaboration with ITU-T. The identical text is published as ITU-T Recommendation X.920.

Annex A forms a normative part of this International Standard. Annex B is for information only.

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for the standardization of Open Distributed Processing (ODP). 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.

One of the components of the architecture (described in RM-ODP Part 3: Architecture) (see ITU-T Rec. X.903 | ISO/IEC 10746-3) is a language that is suitable for describing the signature of computational operation interfaces. This Recommendation | International Standard contains such an Interface Definition Language, called ODP-IDL.

NOTE – This Recommendation | International Standard is technically aligned with the CORBA Interface Definition Language specification.

Annex A is normative and provides a standard set of exceptions for a particular ODP distribution infrastructure.

Annex B is informative and provides the CORBA encoding of a type called TypeCode representing type descriptions.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN DISTRIBUTED PROCESSING – INTERFACE DEFINITION LANGUAGE

1 Scope

This Recommendation | International Standard is intended to provide the ODP Reference Model (see ITU-T Rec. X.902 | ISO/IEC 10746-2 and ITU-T Rec. X.903 | ISO/IEC 10746-3) with a language and environment neutral notation to describe computational operation interface signatures. Use of this notation does not imply use of specific supporting mechanisms and protocols.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology – Open distributed processing – Reference Model: Foundations.*
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology – Open distributed processing – Reference Model: Architecture.*

2.2 Additional references

- ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange.*
- ISO/IEC 8859-1:1998, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1.*

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.902 | ISO/IEC 10746-2:

- object;
- interface;
- interface signature.

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.903 | ISO/IEC 10746-3:

- operation.

4 ODP IDL syntax and semantics

ODP IDL (the Interface Definition Language) is the language used to describe the interface signatures for interfaces that client objects call and object implementations provide. An interface definition written in ODP IDL completely defines the interface signature and fully specifies each operation's parameters.

An ODP 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 the preprocessing is a sequence of tokens. Such a sequence of tokens, that is, a file after preprocessing, is called a translation unit.

ODP IDL obeys the same lexical rules as C++[1)], although new keywords are introduced to support distribution concepts. It also provides full support for standard C++ preprocessing features. The ODP IDL specification is expected to track relevant changes to C++ introduced by the ISO/IEC standardization effort.

The description of ODP IDL's lexical conventions is presented in 4.1. A description of ODP IDL preprocessing is presented in 4.2. The scope rules for identifiers in an ODP IDL specification are described in 4.13 on.

The ODP IDL grammar is a subset of ISO/IEC C++ with additional constructs to support the operation invocation mechanism. ODP IDL is a descriptive language; it supports C++ syntax for constant, type, and operation declarations; it does not include any algorithmic structures or variables. The grammar is presented in 4.3.

This clause describes ODP IDL semantics and gives the syntax for ODP IDL grammatical constructs. The description of ODP IDL grammar uses a syntax notation that is similar to Extended Backus-Naur format (EBNF). Table 1 lists the symbols used in this format and their meaning.

Table 1 – ODP IDL EBNF format

Symbol	Meaning
::=	Is defined to be
|	Alternatively
<text>	Non-terminal
"text"	Literal
*	The preceding syntactic unit can be repeated zero or more times.
+	The preceding syntactic unit can be repeated one or more times.
{}	The enclosed syntactic units are grouped as a single synctactic unit.
[]	The enclosed syntactic unit is optional – may occur zero or more times.

4.1 Lexical conventions

This subclause[2)] presents the lexical conventions of ODP IDL. It defines tokens in an ODP IDL specification and describes comments, identifiers, keywords, and literals – integer, character, and floating point constants and string literals.

ODP IDL uses the ISO/IEC Latin-1 (ISO/IEC 8859-1) character set. This character set is divided into alphabetic characters (letters), digits, graphic characters, the space (blank) character and formatting characters. Table 2 shows the ODP IDL alphabetic characters; upper- and lower-case equivalencies are paired. Table 3 shows the digits and Table 4 shows the graphic characters.

The formatting characters are shown in Table 5.

---

1) Ellis, Margaret A. and Bjarne Stroustrop, *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1990, ISBN 0-201-51459-1.

2) This sublcause is an adaptation of *The Annotated C++ Reference Manual*, Chapter 2; it differs in the list of legal keywords and punctuation.

Table 2 – The 114 alphabetic characters (letters)

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 diaeresis
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
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 diaeresis
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 diaeresis
Qq	Upper/Lower-case Q	Ðð	Upper/Lower-case Icelandic eth
Rr	Upper/Lower-case R	Ññ	Upper/Lower-case N with tilde
Ss	Upper/Lower-case S	Òò	Upper/Lower-case O with grave accent
Tt	Upper/Lower-case T	Óó	Upper/Lower-case O with accute accent
Uu	Upper/Lower-case U	Ôô	Upper/Lower-case O with circumflex accent
Vv	Upper/Lower-case V	Õõ	Upper/Lower-case O with tilde
Ww	Upper/Lower-case W	Öö	Upper/Lower-case O with diaeresis
Xx	Upper/Lower-case X	Øø	Upper/Lower-case O with oblique stroke
Yy	Upper/Lower-case Y	Ùù	Upper/Lower-case U with grave accent
Zz	Upper/Lower-case Z	Úú	Upper/Lower-case U with acute accent
		Ûû	Upper/Lower-case U with circumflex accent
		Üü	Upper/Lower-case U with diaeresis
		Ýý	Upper/Lower-case Y with acute accent
		Þþ	Upper/Lower-case Icelandic thorn
		ß	Lower-case German sharp S
		ÿ	Lower-case Y with diaeresis

Table 3 – Decimal digits

0 1 2 3 4 5 6 7 8 9

Table 4 – The 65 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
'	Apostrophe	§	Section/paragraph sign
(	Left parenthesis	¨	Diaeresis
)	Rigth 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	x	Multiplication sign
		÷	Division sign

Table 5 – The Formatting Characters

Description	Abbreviation	ISO/IEC 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

4.1.1 Tokens

There are five kinds of tokens: identifiers, keywords, literals, operators, and other separators. Blanks, horizontal and vertical tabs, newlines, formfeeds, 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.

4.1.2 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.

4.1.3 Identifiers

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

Identifiers that differ only in case collide and yield a compilation error. An identifier for a definition must be spelled consistently (with respect to case) throughout a specification.

When comparing two identifiers to see if they collide:

- Upper- and lower-case letters are treated as the same letter. Table 2 defines the equivalence mapping of upper- and lower-case letters.
- The comparison does *not* take into account equivalences between digraphs and pairs of letters (e.g. "æ" and "ae" are not considered equivalent) or equivalences between accented and non-accented letters (e.g. "à" and "a" are not considered equivalent).
- All characters are significant.

There is only one name space for ODP IDL identifiers. Using the same identifier for a constant and an interface, for example, produces a compilation error.

4.1.4 Keywords

The identifiers listed in Table 6 reserved for use as keywords, and may not be used otherwise. The keyword `Object` in ODP IDL is used to represent an interface type whereas the keyword `interface` is used to indicate the start of an interface declaration in an interface signature template. The keyword "`Object`" can be used as a type specifier. The keyword `attribute` defines a method giving access to a portion of the state of an object. An attribute definition is logically equivalent to declaring a pair of accessor methods; one to retrieve the value of the attribute and one to set the value of the attribute.

The keyword Exception is used to represent the ODP concept of unsuccessful named termination, the exception name being the termination name.

Keywords obey the rules for identifiers (see 1.3) and must be written exactly as shown in the above list. For example, "`boolean`" is correct; "`Boolean`" is not. ODP IDL specifications use the characters shown in Table 7 as punctuation.

In addition, the tokens listed in Table 8 are used by the preprocessor.

Table 6 – Keywords

any	default	in	oneway	struc	wchar
attribute	double	inout	out	switch	wstring
boolean	enum	interface	raises	TRUE	
case	exception	long	readonly	typedef	
char	FALSE	module	sequence	unsigned	
const	fixed	Object	short	union	
context	float	octet	string	void	

Table 7 – Punctuation tokens

;	{	}	:	::	,	=	+	-	(	)	<	>	[	]	
’	"	\	|	^	&	*	/	%	~	<<	>>				

Table 8 – Preprocessor tokens

#	##	!	||	&&	include	pragma	define

4.1.5 Literals

Wide character and wide string literals are specified exactly like character and string literals. All character and string literals, both wide and non-wide, may only be specified (portably) using the characters found in the ISO/IEC 8859-1 character set. Note that these extensions for international characters only affect the specification of literals in the ODP IDL and not the rest of ODP IDL source files. That is, the interface names, operation names, type names, etc., will continue to be limited to the ISO/IEC 8859-1 character set.

Literals of the new integer and floating-point types are specified as described in this subclause (*Integer, Literals* and *Floating-Point Literals*).

4.1.5.1 Integer literals

An integer literal consisting of a sequence of digits is taken to be decimal (base ten) unless it begins with 0 (digit zero). A sequence of digits starting with 0 is taken to be an octal integer (base eight). The digits 8 and 9 are not octal digits. A sequence of digits preceeded by 0x or 0X is taken to be a hexadecimal integer (base sixteen). The hexadecimal digits include a or A through f or F with decimal values ten through fifteen, respectively. For example, the number twelve can be written 12, 014 or 0XC.

4.1.5.2 Character literals

A character literal is one or more characters enclosed in single quotes, as in ’x’. Character literals have type`char`.

A character is an 8-bit quantity with a numerical value between 0 and 255 (decimal). The value of a space, alphabetic, digit or graphic characterliteral is the numerical value of the character as defined in the ISO/IEC Latin-1 (ISO/IEC 8859-1) character set standard (see Tables 2, 3, and 4). The value of a null is 0. The value of a formatting character literal is the numerical value of the character as defined in the ISO/IEC 646 standard (see Table 5). The meaning of all other characters is implementation-dependent.

Non-graphic characters must be represented using escape sequences as defined below in Table 9. Note that escape sequences must be used to represent singlequote and backlash characters in character literals.

If the character following a backlash is not one of those specified, the behaviour is undefined. An escape sequence specifies a single character.

The escape \000 consists of the backlash followed by one, two, or three octal digits that are taken to specify the value of the desired character. The escape \xhh consists of the backlash followed by x followed by one or two hexadecimal digits that are taken to specify the value of the desired character. A sequence of octal or hexadecimal digits is terminated by the first character that is not an octal digit or a hexadecimal digit respectively. The value of a character constant is implementation dependent if it exceeds that of the largest character.

Table 9 – Escape sequences

Description	Escape sequence
Newline	\n
Horizontal tab	\t
Vertical tab	\v
Backspace	\b
Carriage return	\r
Form feed	\f
Alert	\a
Backslash	\\
Question mark	\?
Single quote	\’
Double quote	\"
Octal number	\ooo
Hexadecimal number	\xhh

4.1.5.3 Floating-point literals

A floating-point consists of an integer part; a decimal point, a fraction part, an e or E and an optionally signed integer exponent. The integer and fraction parts both consist of a sequence of decimal (base ten) digits. Either the integer part or the fraction part (but not both) may be missing; either the decimal point or the letter e (or E) and the exponent (but not both) may be missing.

4.1.5.4 Fixed-point literal

A fixed-point decimal literal consists of an integer part, a decimal point, a fraction part and a d or D. The integer and fraction parts both consist of a sequence of decimal (base 10) digits. Either the integer part or the fraction part (but not both) may be missing; the decimal point [but not the letter d (or D)] may be missing.

4.1.5.5 String literals

A string literal is a sequence of characters (as defined in 4.1.5.2) surrounded by double quotes, as in "...".

Adjacent string literals are concatenated. Characters in concatenated strings are kept distinct. For example,

"\xA" "B" contains the two characters ’\xA’ and ’B’ after concatenation (and not the single hexadecimal character ’\xAB’).

The size of a string literal is the number of character literals enclosed by the quotes, after concatenation. The size of the literal is associated with the literal.

4.2 Preprocessing

A preprocessing notation can be used as a module notation, in order to organize specifications and to be able to refer to parts of a specification in a given specification. Therefore, the source file inclusion #include must be understood as a generic way of including a given module of specification and is not linked with any particular filing system.

ODP IDL preprocessing which is specified in the *ANSI/ISO C++ Standard* provides macrosubstitution, conditional compilation, and source file inclusion. In addition, directives are provided to control line numbering in diagnostics and for symbolic debugging, to generate a diagnostic message with a given token sequence, and to perform implementation-dependent actions (the # pragma directive). Certain predefined names are available. These facilities are conceptually handled by a preprocessor, which may or may not actually be implemented as a separate process.

Lines beginning with # (also called "directives") communicate with this preprocessor. White space may appear before the #. These lines have syntax independent of the rest of the ODP IDL; they may appear anywhere and have effects that last (independent of the ODP IDL scoping rules) until the end of the translation unit. The textual location of ODP IDL-specific pragmas may be semantically constrained.

A preprocessing directive (or any line) may be continued on the next line in a source file by placing a backslash character ("\"), immediately before the newline at the end of the line to be continued. The preprocessor effects the continuation by deleting the backslash and the newline before the input sequence is divided into tokens. A backslash character may not be the last character in a source file.