INTERNATIONAL STANDARD

ISO/IEC 14395

First edition
1996-06-01

Information technology — Test methods for measuring conformance to directory services C language interfaces — Binding for Application Program Interface (API)

Technologies de l'information — Méthodes d'essai pour mesurer la conformité aux interfaces de langage C des services de l'annuaire — Liant pour interface de programme d'application (API)

Reference number
ISO/IEC 14395:1996(E)

Contents

TABLES

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.

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 14395 was prepared by IEEE (as IEEE Std 1328.2-1993) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Annex A of this International Standard is for information only.

Introduction

(This introduction is not a normative part of ISO/IEC 14395, Information technology—Test methods for measuring conformance to directory services C language interfaces—Binding for Application Program Interface (API), but is included for information only.)

The purpose of this International Standard is to define test methods for the C language binding contained in ISO/IEC 14394 {8} for the language-independent specification contained in ISO/IEC 14392 {6} of the application program interface (API) to directory services.

A directory is a distributed collection of information, that programs can access in order to make queries or updates. ISO/IEC 14392 {6} defines, in programming language independent terms, an API to directory services. This API is known as the Directory Services API (DS API). ISO/IEC 14394 {8} defines a C language binding for the DS API.

The DS API is intended to be used to provide access to a range of directory services that are instances of a common abstract model. That model is defined in the 1988 CCITT X.500 Series recommendations and ISO/IEC 9594: 1990. ISO/IEC 14392 {6} prescribes how the DS API is to be used to access the particular directory service defined in ISO/IEC 9594: 1990 and indicates how it may be used to access other directory services that conform to the same abstract model. Nothing in ISO/IEC 14392 {6} or in this International Standard requires that the implementation of the interface or the Directory itself actually make use of the Directory Access Protocol (DAP), the Directory System Protocol (DSP), or other parts of the model, just so long as it provides the defined service.

The interface is designed for operational interactions with a directory, rather than for management interactions such as knowledge management or schema management. Also, security features are not generally visible in the interface in order to permit flexibility in security policies. It is intended that an application program should be able to use the interface to access a single directory service or to access several directory services at the same time.

Related Standards

ISO/IEC 14392 {6} is intended to provide the basis for the definition of programming language bindings to which implementations and applications can conform. A specification for such a language binding, for the C programming language, is contained in ISO/IEC 14394 {8}. This International Standard applies to test methods for measuring conformance to that programming language binding specification.

ISO/IEC 14393 {7} specifies test methods for the language-independent specification contained in ISO/IEC 14392 {6}. A set of test methods for the C binding to ISO/IEC 14392 {6} must satisfy the requirements of ISO/IEC 14393 {7}, as well as conforming to this International Standard.

The API defined in ISO/IEC 14392 {6} uses the mechanism for OSI abstract data Manipulation (OM) defined in ISO/IEC 14360 {B13}. ISO/IEC 14362 {3} defines the requirements that apply to test methods for measuring conformance to ISO/IEC 14360 {B13}. A C language binding to ISO/IEC 14360 {B13} is defined in ISO/IEC 14364 {4}. ISO/IEC 14366 {5} defines the requirements that apply to test methods for measuring conformance to ISO/IEC 14364 {4}.

IEEE Std 1003.3-1991 {9} defines the general requirements that shall apply to test methods for measuring conformance to POSIX. A set of test methods used to measure conformance to ISO/IEC 14392 {6} must satisfy the requirements of IEEE Standard 1003.3, as well as conforming to this International Standard.

Overview

For each section of ISO/IEC 14394 {8}, this International Standard contains a corresponding section containing test assertions, in accordance with IEEE Std 1003.3-1991 {9}. A set of test methods conforms to this International Standard if it tests all of the test assertions.

This International Standard is based on IEEE Std 1328.2-1993 {B15}, which was prepared by the Namespace and Directory Services Working Group (P1224.2, formerly P1003.17), sponsored by the Portable Applications Standards Committee of the IEEE Computer Society.

Information technology—Test methods for measuring conformance to directory services C language interfaces—Binding for Application Program Interface (API)

Section 1: General

1.1 Scope

This International Standard defines requirements for test methods for measuring conformance to ISO/IEC 14394 {8}.

ISO/IEC 14394 {8} contains a programming language binding specification for ISO/IEC 14392 {6}, using the C programming language. ISO/IEC 14393 {7} contains language-independent requirements for test methods for measuring conformance to programming language binding specifications for ISO/IEC 14392 {6}, such as that contained in ISO/IEC 14394 {8}. This International Standard contains C language-specific requirements for the test methods. Taken in conjunction with the requirements imposed by ISO/IEC 14393 they constitute the requirements that shall be satisfied by test methods used for measuring conformance to ISO/IEC 14394 {8}.

1.2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

{1} ISO/IEC 9899:1990,[1] *Programming languages—C.*

{2} ISO/IEC 9945-1: 1990[2] (IEEE Std 1003.1: 1990),[3] *Information technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface (API) [C language].*

{3} ISO/IEC 14362: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation—Application Program Interface (API) [Language independent].*

{4} ISO/IEC 14364: 1996, *Information technology—Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API).*

{5} ISO/IEC 14366: 1996, *Information technology—Test methods for measuring conformance to Open Systems Interconnection (OSI) abstract data manipulation C language interfaces—Binding for Application Program Interface (API).*

{6} ISO/IEC 14392: 1996, *Information technology—Directory services—Application Program Interface (API) [Language independent].*

{7} ISO/IEC 14393: 1996, *Information technology—Test methods for measuring conformance to directory services—Application Program Interface (API) [Language independent].*

{8} ISO/IEC 14394: 1996, *Information Technology—Directory services C language interfaces—Binding for Application Program Interface (API).*

{9} IEEE Std 1003.3-1991, *IEEE Standard for Information Technology—Test Methods for Measuring Conformance to POSIX.*

---

1) ISO/IEC documents can be obtained from the ISO Central Secretariat, 1 Rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

2) ISO/IEC 9945-1: 1990 is currently under revision.

3) IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

1.3 Conformance

A set of test methods that conforms to this International Standard shall conform to IEEE Std 1003.3-1991 {9}, with references in IEEE Std 1003.3-1991 {9} to the "POSIX.n test method specification" being interpreted as references to this International Standard, and references in IEEE Std 1003.3-1991 {9} to "the POSIX standard for which conformance is being measured" being interpreted as references to ISO/IEC 14394 {8}.

In addition to meeting the conformance criteria defined in IEEE Std 1003.3-1991 {9}, a set of test methods that conforms to this International Standard shall test all documentation assertions defined in this International Standard.

NOTE: Conformance to IEEE Std 1003.3-1991 {9} implies that the test methods will test all other assertions defined in this International Standard.

A set of test methods that conforms to this International Standard shall also conform to ISO/IEC 14393 {7}.

This page intentionally left blank

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographical Conventions

Many technical terms, such as "object" and "attribute," are used in describing both OSI abstract data manipulation and directory services. In the first case, they refer to constructs of the OSI abstract data manipulation interface, while in the second case they refer to constructs of the directory service to which the interface provides access. The meanings ascribed to these terms in these cases are often similar but different. Throughout this International Standard, care is taken to distinguish between these meanings, for example by qualifying the term with "OM" or "directory" as appropriate (as with OM classes and directory classes and with OM attributes and directory attributes). The unqualified term "attribute" denotes the directory construct, while the phrase "OM attribute" denotes the OSI abstract data manipulation construct. The phrase "object class" denotes the directory construct, while the phrase "OM class" denotes the OSI abstract data manipulation construct.

The term "operation" is used in two different senses, one by the language-independent specification conventions and one by the directory service descriptions. When used in the sense of the directory, it is always qualified by the term "directory," in the phrase "directory operation." The unqualified term "operation" denotes the language-independent specification construct. Sometimes, to reinforce the distinction, the phrase "interface operation" is used to denote the language-independent specification construct, instead of "operation."

The reader is urged to be cautious, and reference to 2.2 (Definitions) may be useful.

Language-independent concrete OM class names, OM attribute names, and OM attribute value names appear in bold font while abstract OM class names appear in bold italic font. They are all spelled with hyphens between words. The first letters of language-independent OM class and OM attribute names are capitalized (e.g., **Filter-Item**).

Language-independent datatype, operation argument, and error names appear in Helvetica font, are lowercased, and are spelled with underscores between words (e.g., ds_abandon).

Each OM class has an associated datatype whose name is obtained from the class name by converting uppercase to lowercase, converting hyphens to underscores,

adding the prefix "ds_", and adding the suffix "_type". (Thus, ds_filter_item_type is the datatype corresponding to OM class **Filter-Item**).

Function synopses and other extended pieces of C code appear in constant width (courier) font while C language names embedded in ordinary text appear in italic font. Underscores are used to separate words in C language names. The C language names are derived mechanically from the corresponding language-independent names in a manner described in 2.1.4.

A C language function, datatype, or code fragment appears in *italic* font. A function name is indicated by following parentheses [e.g., *ds_abandon()*].

A C language symbolic constant, other than an error name, is surrounded by braces (e.g., {DS_MAX_OUTSTANDING_OPERATIONS}).

A C language symbolic constant for an error name is surrounded by brackets (e.g., [DS_E_ADMIN_LIMIT_EXCEEDED]).

When important terms are introduced, they appear in italics. Italics are also used in 2.2 for cross-references.

The use of fonts in this International Standard is as follows:

(1) The Helvetica font is used for:

— Language-independent operation names, such as ds_add_entry

— Language-independent datatype names, such as ds_status_type

— Language-independent error names, such as Abandon-Failed

(2) The *italic* font is used for:

— Language-independent operation arguments, such as *session*

— C language names embedded in ordinary text

— The introduction of important terms

— Cross-references in 2.2

(3) The **bold** font is used for:

— Language-independent concrete OM class names, such as **Filter-Item**

— Language-independent OM attribute names, such as **Filter-Item-Type**

— Language-independent OM attribute values, such as **approximate-match**

(4) The ***bold italic*** font is used for:

— Language-independent abstract OM class names, such as ***Common-Results***

(5) The `constant width` (Courier) font is used for:

— References to terms defined in the X.500 directory standards

— Function synopses and other extended pieces of C code.

2.1.2 Representation of Strings of Numbers

Strings of numbers (such as ASN.1 Object Identifier encodings) are represented in the form "\xaa\xbb\xcc . . ", where each term \xnn represents an element of the string, and the value of that element expressed in hexadecimal notation is nn. So, for example, the string consisting of the three numbers 1, 10, 100 is represented as "\x01\x0a\x64".

2.1.3 Language-Independent Conventions

The language-independent specification for this International Standard is contained in ISO/IEC 14392 {6}. The language-independent specification conventions apply to that standard and to language-independent specification terms used in this International Standard. These conventions are described in ISO/IEC 14360 {B13}.

2.1.4 C Language Binding Conventions

2.1.4.1 Introduction

ISO/IEC 14394 {8} specifies C identifiers for all the elements of the interface, so that application programs written in C can access the Directory. These elements include function names, typedef names, and constants. All the C identifiers are mechanically derived from the language-independent names as explained below.

2.1.4.2 C Naming Conventions

The interface uses part of the C public namespace for its facilities. All identifiers start with the letters *ds*, *DS*, or *OMP*. More detail about the conventions used is given in Table 2-1. The interface reserves all identifiers starting with the letters *dsP* for private (i.e., internal) use by implementations of the interface. It also reserves *all* identifiers starting with the letters *dsX* or *DSX* for extensions of the interface. If the <xds.h> header is included, the application shall not use any identifier starting with these letters.

ISO/IEC 14360 {B13} uses similar, though not identical, naming conventions. All OSI abstract data manipulation identifiers are prefixed by the letters *OM* or *om*.

The C identifiers are derived from the language-independent names used throughout this International Standard by a purely mechanical process that depends on the kind of name:

— The C function names are identical to the language-independent operation names. Within text, they are italicized and followed by "()".
Thus

— ds_receive_result

becomes

— *ds_receive_result()*

Table 2-1 – C Naming Conventions

Item	Prefix
Reserved for implementors	*dsP*
Reserved for interface extensions	*dsX*
Reserved for interface extensions	*DSX*
Reserved for implementors	*OMP*
Functions	*ds_*
Error "problem" values	*DS_E_*
OM class names	*DS_C_*
OM value length limits	*DS_VL_*
OM value number limits	*DS_VN_*
Other constants	*DS_*
Attribute Type	*DS_A_*
Object Class	*DS_O_*

— C function input parameters are identical to the corresponding language-independent argument names. C function output parameters are derived from the argument names by adding "*_return*" as a suffix. Thus, the output argument

  — *operation_status*

becomes

  — *operation_status_return*

(Within text, both language-independent argument names and C function parameter names are italicized.)

— The names of constants identifying OM classes are derived from the class names by converting lowercase to uppercase, converting hyphens to underscores, and adding the prefix "*DS_C_*".
Thus

  — **Read-Result**

becomes

  — {DS_C_READ_RESULT}

— The names of C language constants that denote language-independent specification choice values are derived from the choice value names by converting lowercase to uppercase and adding the prefix "*DS_*".
Thus

  — **Default-Context**

becomes

  — {DS_DEFAULT_CONTEXT}

— Enumeration tags are derived from the name of the corresponding OM syntax by adding the prefix "*DS_*". Hyphens are converted to underscores, but the case of letters is left unchanged.
Thus

  — **Enum(Limit-Problem)**

becomes

  — *DS_Limit_Problem*

— For enumeration constants, OM attributes and all other constants except errors, lowercase is converted to uppercase, hyphens are converted to underscores, and the prefix "*DS_*" is added.
Thus

— **O-Residential-Person**
becomes
— {DS_O_RESIDENTIAL_PERSON}

— Errors are treated as a special case. Constants that are the possible values of the OM attribute **Problem** of a subclass of the OM class **Error** have hyphens converted to underscores, are made entirely uppercase, and are prefixed by "*DS_E_*".
Thus

— alias-dereferencing-problem
becomes
— [DS_E_ALIAS_DEREFERENCING_PROBLEM]

— The constants in the "Value Length" and "Value Number" columns of the OM class definition tables are also assigned identifiers. (They have no names in the language-independent specification.) When the upper limit in one of these columns is not "1" (one), it is given a name consisting of the OM attribute name prefixed by "*DS_VL_*" for value length or "*DS_VN_*" for value numbers.

— The sequence of octets for each object identifier is also assigned an identifier for internal use by certain OM macros. These identifiers are all uppercase and are prefixed by "*OMP_O_*". See ISO/IEC 14360 {B13} for further details on the use of object identifiers.

— In some cases, the transformations described above result in identifiers that are more than 32 characters in length. The ISO C Standard {1} only requires identifiers to be unique within the first 32 characters. The abbreviations listed in Table 2-2 are applied to substrings of identifiers to reduce their length.

Note that hyphens are translated to underscores.

2.1.4.3 Use and Implementation of Interfaces

If an argument to a function has an invalid value (such as a value outside the domain of the function, a pointer outside the address space of the program, or a null pointer), the behavior is undefined.

Any function declared in a header may be implemented as a macro defined in the header, so a library function should not be declared explicitly if its header is included. Any macro definition of a function can be suppressed locally by enclosing the name of the function in parentheses, because the name is not then followed by the left parenthesis that indicates expansion of a macro function name. For the same syntactic reason, it is permitted to take the address of a library function even if it is also defined as a macro. The use of *#undef* to remove any macro definition will also ensure that the identifier refers to an actual function. Any invocation of a library function that is implemented as a macro shall expand to code that evaluates each of its arguments exactly once, fully protected by parentheses where necessary, so it is generally safe to use arbitrary expressions as arguments.