INTERNATIONAL STANDARD

ISO/IEC 13816

Second edition
2007-10-01

Information technology — Programming languages, their environments and system software interfaces — Programming language ISLISP

Technologies de l'information — Langages de programmation, leurs environnements et interfaces de logiciel système — Langage de programmation ISLISP

Reference number
ISO/IEC 13816:2007(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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. 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.

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.

ISO/IEC 13816 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

This second edition cancels and replaces the first edition (ISO/IEC 13816:1997), which has been technically revised.

Introduction

The programming language ISLisp is a member of the Lisp family.

The following factors influenced the establishment of design goals for ISLisp:

1. A desire of the international Lisp community to standardize on those features of Lisp upon which there is widespread agreement.
2. The existence of the incompatible dialects Common-Lisp, EuLisp, Le-Lisp, and Scheme (mentioned in alphabetical order).
3. A desire to affirm Lisp as an industrial language.

This led to the following design goals for ISLisp:

1. ISLisp shall be compatible with existing Lisp dialects where feasible.
2. ISLisp shall have as a primary goal to provide basic functionality.
3. ISLisp shall be object-oriented.
4. ISLisp shall be designed with extensibility in mind.
5. ISLisp shall give priority to industrial needs over academic needs.
6. ISLisp shall promote efficient implementations and applications.

Information technology — Programming languages, their environments and system software interfaces — Programming language ISLISP

1 Scope

This International Standard specifies syntax and semantics of the computer programming language ISLisp by specifying requirements for a conforming ISLisp processor and a conforming ISLisp text.

This International Standard does not specify:

(a) the size or complexity of an ISLisp text that exceeds the capacity of any specific data processing system or the capacity of a particular processor, nor the actions to be taken when the corresponding limits are exceeded;

(b) the minimal requirements of a data processing system that is capable of supporting an implementation of a processor for ISLisp;

(c) the method of preparation of an ISLisp text for execution and the method of activation of this ISLisp text, prepared for execution;

(d) the typographical presentation of an ISLisp text published for human reading;

(e) extensions that might or might not be provided by the implementation.

2 Normative references

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

- ISO/IEC TR 10034:1990, *Guidelines for the preparation of conformity clauses in programming language standards*
- IEEE standard 754-1985. *Standard for binary floating point arithmetic*

3 Compliance of ISLisp processors and text

An ISLisp processor complying with the requirements of this International Standard shall:

(a) accept and implement all features of ISLis p specified in this International Standard;

(b) reject any text that contains any textual usage which this International Standard explicitly defines to be a violation (see §9);

(c) be accompanied by a document that provides the definitions of all implementation-defined features;

(d) be accompanied by a document that separately describes any features accepted by the processor that are not specified in this International Standard; these extensions shall be described as being "extensions to ISLisp as specified by ISO/IEC 13816:2007(E)."

A complying ISLisp text shall not rely on implementation-dependent features. However, a complying ISLisp text may rely on implementation-defined features required by this International Standard.

A complying ISLisp text shall not attempt to create a lexical variable binding for any named constant defined in this International Standard. It is a violation if any such attempt is made.

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

4.1
abstract class
class that by definition has no direct instances

4.2
activation
computation of a function

Note: Every activation has an activation point, an activation period, and an activation end. The activator, which is a function application form prepared for execution, starts the activation at the activation point.

4.3
accessor
association of a reader and a writer for a slot of an instance

4.4
argument position
occurrence of a text unit as an element in a form excluding the first one

4.5
binding
concept that has both a syntactic and a semantic aspect, where

- syntactically, “binding” describes the relation between an identifier and a binding ISLISP form, and
- semantically, “binding” describes the relation between a variable, its denoting identifier, and an object (or, the relation between a variable and a location)

Note 1: The property of being bound can be checked textually by relating defining and applied identifier occurrences.

Note 2: Semantically, the binding relation might be imagined to be materialized in some entity, the binding. Such a binding entity is constructed at run time and destroyed later, or might have indefinite extent.

4.6
class
object that determines the structure and behavior of a set of other objects called its instances

Note: The behavior is the set of operations that can be performed on an instance.

4.7
condition
object that represents a situation that has been (or might be) detected by a running program

4.8
definition point
textual point of an ISLISP text that is the start of an identifier’s representation of an ISLISP object

4.9
direct instance
instance of a class but not an instance of one of its subclasses

Note: Every ISLISP object is direct instance of exactly one class, which is called “its class”. The set of all direct instances together with their behavior constitute a class.

4.10
dynamic
having an effect that is determined only through program execution and that cannot, in general, be determined statically

4.11
dynamic variable
variable whose associated binding is determined by the most recently executed active block that established it, rather than statically by a lexically apparent block according to the lexical principle

4.12
evaluation
computation of a form prepared for execution which results in a value and/or a side-effect

4.13
execution
sequence of (sometimes nested) activations

4.14
extension
implementation-defined modification to the requirements of this International Standard that does not invalidate any ISLisp text complying with this International Standard (except by prohibiting the use of one or more particular spellings of identifiers), does not alter the set of actions which are required to signal errors, and does not alter the status of any feature designated as implementation dependent

4.15
form
single, syntactically valid unit of program text, capable of being prepared for execution

4.16
function
ISLisp object that is called with arguments, performs a computation (possibly having side-effects), and returns a value

4.17
generic function
function whose application behavior is determined by the classes of the values of its arguments and which consists – in general – of several methods

4.18
identifier
lexical element (lexeme) which designates an ISLisp object

Note: In the data structure representation of ISLisp texts, identifiers are denoted by symbols.

4.19
immutable binding
binding in which the relation between an identifier and the object represented by this identifier cannot be changed

Note: It is a violation if there is attempt to change an immutable binding (error-id. *immutable-binding*).

4.20
immutable object
object which is not subject to change, either because no operator is provided that is capable of effecting such change, or because some constraint exists which prohibits the use of an operator that might otherwise be capable of effecting such a change

Note: Except as explicitly indicated otherwise, a conforming processor is not required to detect attempts to modify immutable objects; the consequences are undefined if an attempt is made to modify an immutable object.

4.21
implementation defined
feature, possibly differing between different ISLisp processors, but completely defined for every processor

4.22
implementation dependent
feature, possibly differing between different ISLISP processors, but not necessarily defined for any particular processor

Note: A conforming ISLISP text must not depend upon implementation-dependent features.

4.23
inheritance
relation between a class and its superclass which maps structure and behavior of the superclass onto the class

Note: ISLISP supports a restricted form of multiple inheritance; *i.e.,* a class may have several direct superclasses at once.

4.24
instance
⟨class⟩ either a direct instance of a class or an instance of one of its subclasses

4.25
literal
object whose representation occurs directly in a program as a constant value

4.26
metaclass
class whose instances are themselves classes

4.27
method
case of a generic function for a particular parameter profile, which defines the class-specific behavior and operations of the generic function

4.28
object
anything that can be created, destroyed, manipulated, compared, stored, input, or output by the ISLISP processor

Note 1: In particular, functions are ISLISP objects.

Note 2: Objects that can be passed as arguments to functions, can be returned as values, can be bound to variables, and can be part of structures, are called *first-class objects*.

4.29
operator
first element of a compound form, which is either a reserved name that identifies the form as a special form, or the name of a macro, or a lambda expression, or else an identifier in the function namespace

4.30
operator position
occurrence of a text unit as the first element in a form

4.31
parameter profile
parameter list of a method, where each formal parameter is accompanied by its class name

Note: If a parameter is not accompanied by a class name, it belongs to the most general class.

4.32
place
location where objects can be stored and retrieved later

Note: Places are designated by forms which are permitted as the first argument of `setf`. If used this way an object is stored in the place. If the form is not used as first argument of `setf` the stored object is retrieved. The cases are listed in the description of `setf`.

4.33
process
execution of an ISLisp text prepared for execution

4.34
processor
system or mechanism, that accepts an ISLisp text (or an equivalent data structure) as input, prepares it for execution, and executes the result to produce values and side-effects

4.35
program
aggregation of expressions to be evaluated, the specific nature of which depends on context

Note: Within this International Standard, the term “program” is used only in an abstract way; there is no specific syntactic construct that delineates a program.

4.36
scope
⟨identifier⟩ the textual part of a program where the meaning of that identifier is defined; *i.e.,* there exists an ISLisp object designated by this identifier

4.37
slot
named component of an instance which can be accessed using the slot accessors

Note: The structure of an instance is defined by the set of its slots.

4.38
text
text that complies with the requirements of this International Standard (*i.e.,* with the syntax and static semantics of ISLisp)

Note: An ISLisp text consists of a sequence of toplevel forms.

4.39
toplevel form
any form that either is not nested in any other form or is nested only in `progn` forms

4.40
toplevel scope
scope in which a complete ISLisp text unit is processed

4.41
writer
method associated with a slot of a class, whose task is to bind a value with a slot of an instance of that class

5 Notation and conventions

For a clear definition of, and a distinction between, syntactic and semantic concepts, several levels of description abstraction are used in the following.

There is a correspondence from ISLISP textual units to their ISLISP data structure representations. Throughout this International Standard the text and the corresponding ISLISP objects (data structures) are addressed simultaneously. ISLISP text can be seen as an external specification of ISLISP data structures. To distinguish between the two representations different concepts are used. When textual representation is discussed, textual elements (such as *identifiers*, *literals*, and *compound forms*) are used; when ISLISP objects are discussed, *objects* (such as *symbols* and *lists*) are used.

The constituents of ISLISP text are called ***forms***. A ***form*** can be an *identifier*, a *literal*, or a *compound form*. A *compound form* can be a *function application form*, a *macro form*, a *special form*, or a *defining form*.

An *identifier* is represented by a *symbol*. A *compound form* is represented by a non-null *list*. A *literal* is represented by neither a symbol nor a list, and so is neither an *identifier* nor a *compound form*; for example, a number is a literal.

An object is ***prepared for execution***; this might include transformation or compilation, including macro expansion. The method of preparation for execution and its result are not defined in this International Standard (with exception of the violations to be detected). After successful preparation for execution the result is ready for ***execution***. The combination of preparation for execution and subsequent execution implements ISLISP's ***evaluation model***. The term "evaluation" is used because ISLISP is an expression language—each form has a value which is used to compute the value of the containing form. The results obtained when an entity is prepared for execution are designated throughout this International Standard by the construction "prepared entity"; *e.g.,* "prepared form," "prepared special form."

Example: A "`cond` special form" becomes a "prepared `cond`" by preparation for execution.

In the examples, the metasymbol "⇒" designates the result of an actual evaluation. For example:

$$\texttt{(+ 3 4)} \Rightarrow \texttt{7}$$

The metasymbol "→" identifies the class that results from the evaluation of a form having a given pattern. For example:

$$\texttt{(+ } i_1 \; i_2\texttt{)} \rightarrow \texttt{<}\mathit{integer}\texttt{>}$$

Given a form pattern (usually defined by its constant parts, the function name or special operator), → relates it to the class to which the result of the evaluation of all matching forms belong.