INTERNATIONAL STANDARD

ISO/IEC 11404

First edition
1996-12-15

Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes

Technologies de l'information — Langages de programmation, leur environnement et interfaces du logiciel système — Types de données indépendants du langage

Reference number
ISO/IEC 11404:1996(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 coommittees 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 JTC1. 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 11404 was prepared by Joint Technical Committee ISO/IEC JTC1, Information technology, Subcommittee SC22, *Programming languages, their environments and system software interfaces.*

Annexes A to G of this International Standard are for information only.

Introduction

Many specifications of software services and applications libraries are, or are in the process of becoming, International Standards. The interfaces to these libraries are often described by defining the form of reference, e.g. the "procedure call", to each of the separate functions or services in the library, as it must appear in a user program written in some standard programming language (Fortran, COBOL, Pascal, etc.). Such an interface specification is commonly referred to as the "<language> binding of <service>", e.g. the "Fortran binding of PHIGS" (ISO/IEC 9593-1:1990, *Information processing systems — Computer Graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS) language bindings — Part 1: FORTRAN*).

This approach leads directly to a situation in which the standardization of a new service library immediately requires the standardization of the interface bindings to every standard programming language whose users might reasonably be expected to use the service, and the standardization of a new programming language immediately requires the standardization of the interface binding to every standard service package which users of that language might reasonably be expected to use. To avoid this n-to-m binding problem, ISO/IEC JTC1 (Information Technology) assigned to SC22 the task of developing an International Standard for Language-Independent Procedure Calling and a parallel International Standard for Language-Independent Datatypes, which could be used to describe the parameters to such procedures.

This International Standard provides the specification for the Language-Independent Datatypes. It defines a set of datatypes, independent of any particular programming language specification or implementation, that is rich enough so that any common datatype in a standard programming language or service package can be mapped to some datatype in the set.

The purpose of this International Standard is to facilitate commonality and interchange of datatype notions, at the conceptual level, among different languages and language-related entities. Each datatype specified in this International Standard has a certain basic set of properties sufficient to set it apart from the others and to facilitate identification of the corresponding (or nearest corresponding) datatype to be found in other standards. Hence, this International Standard provides a single common reference model for all standards which use the concept *datatype*. It is expected that each programming language standard will define a mapping from the datatypes supported by that programming language into the datatypes specified herein, semantically identifying its datatypes with datatypes of the reference model, and thereby with corresponding datatypes in other programming languages.

It is further expected that each programming language standard will define a mapping from those Language-Independent (LI) Datatypes which that language can reasonably support into datatypes which may be specified in the programming language. At the same time, this International Standard will be used, among other applications, to define a "language-independent binding" of the parameters to the procedure calls constituting the principal elements of the standard interface to each of the standard services. The production of such service bindings and language mappings leads, in cooperation with the parallel Language-Independent Procedure Calling mechanism, to a situation in which no further "<language> binding of <service>" documents need to be produced: Each service interface, by defining its parameters using LI datatypes, effectively defines the binding of such parameters to any standard programming language; and each language, by its mapping from the LI datatypes into the language datatypes, effectively defines the binding to that language of parameters to any of the standard services.

Information technology — Programming languages, their environments and system software interfaces — Language-independent datatypes

1 Scope

This International Standard specifies the nomenclature and shared semantics for a collection of datatypes commonly occurring in programming languages and software interfaces, referred to as the Language-Independent (LI) Datatypes. It specifies both primitive datatypes, in the sense of being defined ab initio without reference to other datatypes, and non-primitive datatypes, in the sense of being wholly or partly defined in terms of other datatypes. The specification of datatypes in this International Standard is "language-independent" in the sense that the datatypes specified are classes of datatypes of which the actual datatypes used in programming languages and other entities requiring the concept *datatype* are particular instances.

This International Standard expressly distinguishes three notions of "datatype", namely:

- the conceptual, or abstract, notion of a datatype, which characterizes the datatype by its nominal values and properties;
- the structural notion of a datatype, which characterizes the datatype as a conceptual organization of specific component datatypes with specific functionalities; and
- the implementation notion of a datatype, which characterizes the datatype by defining the rules for representation of the datatype in a given environment.

This International Standard defines the abstract notions of many commonly used primitive and non-primitive datatypes which possess the structural notion of atomicity. This International Standard does not define all atomic datatypes; it defines only those which are common in programming languages and software interfaces. This International Standard defines structural notions for the specification of other non-primitive datatypes and provides a means by which datatypes not defined herein can be defined structurally in terms of the LI datatypes defined herein.

This International Standard defines a partial vocabulary for implementation notions of datatypes and provides for, but does not require, the use of this vocabulary in the definition of datatypes. The primary purpose of this vocabulary is to identify common implementation notions associated with datatypes and to distinguish them from conceptual notions. Specifications for the use of implementation notions are deemed to be outside the scope of this International Standard, which is concerned solely with the identification and distinction of datatypes.

This International Standard specifies the required elements of mappings between the LI datatypes and the datatypes of some other language. This International Standard does not specify the precise form of a mapping, but rather the required information content of a mapping.

2 Conformance

An information processing product, system, element or other entity may conform to this International Standard either directly, by utilizing datatypes specified in this International Standard in a conforming manner (2.1), or indirectly, by means of mappings between internal datatypes used by the entity and the datatypes specified in this International Standard (2.2).

NOTE — The general term **information processing entity** is used in this clause to include anything which processes information and contains the concept of *datatype*. Information processing entities for which conformance to this International Standard may be appropriate include other standards (e.g. standards for programming languages or language-related facilities), specifications, data handling facilities and services, etc.

2.1 Direct conformance

An information processing entity which **conforms directly** to this International Standard shall:

i) specify which of the datatypes and datatype generators specified in Clauses 8 and 10 are provided by the entity and which are not, and which, if any, of the declaration mechanisms in Clause 9 it provides; and

ii) define the value spaces of the LI datatypes used by the entity to be identical to the value-spaces specified by this International Standard; and

iii) use the notation prescribed by clauses 7 through 10 of this International Standard to refer to those datatypes and to no others; and

iv) to the extent that the entity provides operations other than movement or translation of values, define operations on the LI datatypes which can be derived from, or are otherwise consistent with, the characterizing operations specified by this International Standard.

NOTES

1. This International Standard defines a syntax for the denotation of values of each datatype it defines, but, in general, requirement *(iii)* does not require conformance to that syntax. Conformance to the value-syntax for a datatype is required only in those cases in which the value appears in a *type-specifier*, that is, only where the value is part of the identification of a datatype.

2. The requirements above prohibit the use of a *type-specifier* defined in this International Standard to designate any other datatype. They make no other limitation on the definition of additional datatypes in a conforming entity, although it is recommended that either the form in Clause 8 or the form in Clause 10 be used.

3. Requirement (*iv*) does not require all characterizing operations to be supported and permits additional operations to be provided. The intention is to permit addition of semantic interpretation to the LI datatypes and generators, as long as it does not conflict with the interpretations given in this International Standard. A conflict arises only when a given characterizing operation could not be implemented or would not be meaningful, given the entity-provided operations on the datatype.

4. Examples of entities which could conform directly are language definitions or interface specifications whose datatypes, and the notation for them, are those defined herein. In addition, the verbatim support by a software tool or application package of the datatype syntax and definition facilities herein should not be precluded.

2.2 Indirect conformance

An information processing entity which **conforms indirectly** to this International Standard shall:

i) provide mappings between its internal datatypes and the LI datatypes conforming to the specifications of Clause 11 of this International Standard; and

ii) specify for which of the datatypes in Clause 8 and Clause 10 an inward mapping is provided, for which an outward mapping is provided, and for which no mapping is provided.

NOTES

1. Standards for existing programming languages are expected to provide for indirect conformance rather than direct conformance.

2. Examples of entities which could conform indirectly are language definitions and implementations, information exchange specifications and tools, software engineering tools and interface specifications, and many other entities which have a concept of datatype and an existing notation for it.

2.3 Conformance of a mapping standard

In order to conform to this International Standard, a standard for a mapping shall include in its conformance requirements the requirement to conform to this International Standard.

NOTES

1. It is envisaged that this International Standard will be accompanied by other standards specifying mappings between the internal datatypes specified in language and language-related standards and the LI datatypes. Such mapping standards are required to comply with this Interna-

tional Standard.

2. Such mapping standards may define "generic" mappings, in the sense that for a given internal datatype the standard specifies a parametrized LI datatype in which the parametric values are not derived from parametric values of the internal datatype nor specified by the standard itself, but rather are required to be specified by a "user" or "implementor" of the mapping standard. That is, instead of specifying a particular LI datatype, the mapping specifies a family of LI datatypes and requires a further user or implementor to specify which member of the family applies to a particular use of the mapping standard. This is always necessary when the internal datatypes themselves are, in the intention of the language standard, either explicitly or implicitly parametrized. For example, a programming language standard may define a datatype INTEGER with the provision that a conforming processor will implement some range of Integer; hence the mapping standard may map the internal datatype INTEGER to the LI datatype :

```
integer range (min..max),
```

and require a conforming processor to provide values for "min" and "max".

3 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 current valid International Standards.

ISO/IEC 8601:1988, *Data elements and interchange formats — Information interchange —Representation of dates and times.*

ISO/IEC 8824:1990, *Information technology — Open Systems Interconnection — Specification of Abstract Syntax Notation One (ASN.1).*

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane.*

4 Definitions

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

NOTE — These definitions may not coincide with accepted mathematical or programming language definitions of the same terms.

4.1 actual parametric datatype: a datatype appearing as a parametric datatype in a use of a datatype generator, as opposed to the *formal-parametric-types* appearing in the definition of the datatype generator.

4.2 actual parametric value: a value appearing as a parametric value in a reference to a datatype family or datatype generator, as opposed to the *formal-parametric-values* appearing in the corresponding definitions.

4.3 aggregate datatype: a generated datatype each of whose values is made up of values of the component datatypes, in the sense that operations on all component values are meaningful.

4.4 annotation: a descriptive information unit attached to a datatype, or a component of a datatype, or a procedure (value), to characterize some aspect of the representations, variables, or operations associated with values of the datatype which goes beyond the scope of this International Standard.

4.5 approximate: a property of a datatype indicating that there is not a 1-to-1 relationship between values of the conceptual datatype and the values of a valid computational model of the datatype.

4.6 bounded: a property of a datatype, meaning both *bounded above* and *bounded below.*

4.7 bounded above: a property of a datatype indicating that there is a value U in the value space such that, for all values s in the value space, s $\leq$ U.

4.8 bounded below: a property of a datatype indicating that there is a value L in the value space such that, for all values *s* in the value space, L $\leq$ s.

4.9 characterizing operations:
(of a datatype): a collection of operations on, or yielding, values of the datatype, which distinguish this datatype from

other datatypes with identical value spaces;

(of a datatype generator): a collection of operations on, or yielding, values of any datatype resulting from an application of the datatype generator, which distinguish this datatype generator from other datatype generators which produce identical value spaces from identical parametric datatypes.

4.10 component datatype: a datatype which is a parametric datatype to a datatype generator, i.e. a datatype on which the datatype generator operates.

4.11 datatype: a set of distinct values, characterized by properties of those values and by operations on those values.

4.12 datatype declaration:

(1) the means provided by this International Standard for the definition of a LI datatype which is not itself defined by this International Standard;

(2) an instance of use of this means.

4.13 datatype family: a collection of datatypes which have equivalent characterizing operations and relationships, but value spaces which differ in the number and identification of the individual values.

4.14 datatype generator: an operation on datatypes, as objects distinct from their values, which generates new datatypes.

4.15 defined datatype: a datatype defined by a type-declaration.

4.16 defined generator: a datatype generator defined by a type-declaration.

4.17 exact: a property of a datatype indicating that every value of the conceptual datatype is distinct from all others in any valid computational model of the datatype.

4.18 formal-parametric-type: an identifier, appearing in the definition of a datatype generator, for which a LI datatype will be substituted in any reference to a (defined) datatype resulting from the generator.

4.19 formal-parametric-value: an identifier, appearing in the definition of a datatype family or datatype generator, for which a value will be substituted in any reference to a (defined) datatype in the family or resulting from the generator.

4.20 generated datatype: a datatype defined by the application of a datatype generator to one or more previously-defined datatypes.

4.21 generated internal datatype: a datatype defined by the application of a datatype generator defined in a particular programming language to one or more previously-defined internal datatypes.

4.22 generator: a datatype generator (q.v.).

4.23 generator declaration:

(1) the means provided by this International Standard for the definition of a datatype generator which is not itself defined by this International Standard;

(2) an instance of use of this means.

4.24 internal datatype: a datatype whose syntax and semantics are defined by some other standard, language, product, service or other information processing entity.

4.25 inward mapping: a conceptual association between the internal datatypes of a language and the LI datatypes which assigns to each LI datatype either a single semantically equivalent internal datatype or no equivalent internal datatype.

4.26 LI datatype:

(1) a datatype defined by this International Standard, or

(2) a datatype defined by the means of datatype definition provided by this International Standard.

4.27 lower bound: in a datatype which is bounded below, the value L such that, for all values s in the value space, L ≤ s.

4.28 mapping:

(of datatypes): a formal specification of the relationship between the (internal) datatypes which are notions of, and specifiable in, a particular programming language and the (LI) datatypes specified in this International Standard;

(of values): a corresponding specification of the relationships between values of the internal datatypes and values of the LI datatypes.

4.29 order: a mathematical relationship among values (see 6.3.2).

4.30 ordered: a property of a datatype which is determined by the existence and specification of an order relationship on its value space.

4.31 outward mapping: a conceptual association between the internal datatypes of a language and the LI datatypes which identifies each internal datatype with a single semantically equivalent LI datatype.

4.32 parametric datatype: a datatype on which a datatype generator operates to produce a generated-datatype.

4.33 parametric value:
(1) a value which distinguishes one member of a datatype family from another, or
(2) a value which is a parameter of a datatype or datatype generator defined by a *type-declaration* (see 9.1).

4.34 primitive datatype: an identifiable datatype that cannot be decomposed into other identifiable datatypes without loss of all semantics associated with the datatype.

4.35 primitive internal datatype: a datatype in a particular programming language whose values are not viewed as being constructed in any way from values of other datatypes in the language.

4.36 representation:
(of a LI datatype): the mapping from the value space of the LI datatype to the value space of some internal datatype of a computer system, file system or communications environment;
(of a value): the image of that value in the representation of the datatype.

4.37 subtype: a datatype derived from another datatype by restricting the value space to a subset whilst maintaining all characterizing operations.

4.38 upper bound: in a datatype which is bounded above, the value U such that, for all values s in the value space, s ≤ U.

4.39 value space: the set of values for a given datatype.

4.40 variable: a computational object to which a value of a particular datatype is associated at any given time; and to which different values of the same datatype may be associated at different times.

5 Conventions Used in this International Standard

5.1 Formal syntax

This International Standard defines a formal datatype specification language. The following notation, derived from Backus-Naur form, is used in defining that language. In this clause, the word *mark* is used to refer to the characters used to define the syntax, while the word *character* is used to refer to the characters used in the actual datatype specification language. Table 5-1 summarizes the syntactic metanotation.

Table 5-1 — Metanotation Marks

"	(QUOTATION MARK)	delimits a terminal symbol
'	(APOSTROPHE)	delimits a terminal symbol
{ }	(CURLY BRACKETS)	delimit a repeated sequence (zero or more occurrences)
[ ]	(SQUARE BRACKETS)	delimit an optional sequence (zero or one occurrence)
|	(VERTICAL LINE)	delimits an alternative sequence
=	(EQUALS SIGN)	separates a non-terminal symbol from its definition
.	(FULL STOP)	terminates a production

A **terminal symbol** is a sequence of marks beginning with either a QUOTATION MARK (") or an APOSTROPHE mark (') and terminated by the next occurrence of the same mark. The terminal symbol represents the occurrence of the sequence of characters in an implementation character-set corresponding to the marks enclosed by (but not including) the QUOTATION MARK or APOSTROPHE delimiters.

A **non-terminal symbol** is a sequence of marks, each of which is either a letter or the HYPHEN-MINUS (-) mark, terminated by the first mark which is neither a letter nor a HYPHEN-MINUS. A non-terminal symbol represents any sequence of terminal symbols which satisfies the *production* for that non-terminal symbol. For each non-terminal symbol there is exactly one production in clauses 7, 8, 9, and 10.

A **sequence** of symbols represents exactly one occurrence of a (group of) terminal symbol(s) represented by each symbol in the sequence in the order in which the symbols appear in the sequence, and no other symbols.

A **repeated sequence** is a sequence of terminal and/or non-terminal symbols enclosed between a LEFT CURLY BRACKET mark ({) and a RIGHT CURLY BRACKET mark (}). A repeated sequence represents any number of consecutive occurrences of the sequence of symbols so enclosed, including no occurrence.

An **optional sequence** is a sequence of terminal and/or non-terminal symbols enclosed between a LEFT SQUARE BRACKET mark ([) and a RIGHT SQUARE BRACKET mark (]). An optional sequence represents either exactly one occurrence of the sequence of symbols so enclosed or no symbols at all.

An **alternative sequence** is a sequence of terminal and/or non-terminal symbols preceded by a VERTICAL LINE (|) mark and followed by either a VERTICAL LINE mark or a FULL STOP mark (.). An alternative sequence represents the occurrence of either the sequence of symbols so delimited or the sequence of symbols preceding the (first) VERTICAL LINE mark.

A **production** defines the valid sequences of symbols which a non-terminal symbol represents. A production has the form:

non-terminal-symbol = valid-sequence .

where *valid-sequence* is any sequence of terminal symbols, non-terminal symbols, optional sequences, repeated sequences and alternative sequences. The EQUALS SIGN (=) mark separates the non-terminal symbol being defined from the valid-sequence which represents its definition. The FULL STOP mark terminates the valid-sequence.

5.2 Text conventions

Within the text:

- A reference to a terminal symbol syntactic object consists of the terminal symbol in quotation marks, e.g. "type".
- A reference to a non-terminal symbol syntactic object consists of the non-terminal-symbol in italic script, e.g. *type-declaration*.
- Non-italicized words which are identical or nearly identical in spelling to a non-terminal-symbol refer to the conceptual object represented by the syntactic object. In particular, *xxx-type* refers to the syntactic representation of an "xxx datatype" in all occurrences.

6 Fundamental Notions

6.1 Datatype

A **datatype** is a a set of distinct values, characterized by properties of those values and by operations on those values. Characterizing operations are included in this International Standard solely in order to identify the datatype. In this International Standard, characterizing operations are purely informative and have no normative impact.

NOTE — Characterizing operations are included in order to assist in the identification of the appropriate datatypes for particular purposes, such as mapping to programming languages.

The term **LI datatype** (for Language-Independent datatype) is used to mean a datatype defined by this International Standard. **LI datatypes** (plural) refers to some or all of the datatypes defined by this International Standard.

The term **internal datatype** is used to mean a datatype whose syntax and semantics are defined by some other standard, language, product, service or other information processing entity.

NOTE — The datatypes included in this standard are "common", not in the sense that they are directly supported by, i.e. "built-in" to, many languages, but in the sense that they are common and useful generic concepts among users of datatypes, which include, but go well beyond, programming languages.

6.2 Value space

A **value space** is the collection of values for a given datatype. The value space of a given datatype can be defined in one of the following ways:

- enumerated outright, or
- defined axiomatically from fundamental notions, or
- defined as the subset of those values from some already defined value space which have a given set of properties, or
- defined as a combination of arbitrary values from some already defined value spaces by a specified construction procedure.

Every distinct value belongs to exactly one datatype, although it may belong to many subtypes of that datatype (see 8.2).

6.3 Datatype properties

The model of datatypes used in this International Standard is said to be an "abstract computational model". It is "computational" in the sense that it deals with the manipulation of information by computer systems and makes distinctions in the typing of information units which are appropriate to that kind of manipulation. It is "abstract" in the sense that it deals with the perceived properties of the information units themselves, rather than with the properties of their representations in computer systems.

NOTES

1. It is important to differentiate between the values, relationships and operations for a datatype and the representations of those values, relationships and operations in computer systems. This International Standard specifies the characteristics of the conceptual datatypes, but it only provides a means for specification of characteristics of representations of the datatypes.

2. Some computational properties derive from the *need for the information units to be representable* in computers. Such properties are deemed to be appropriate to the abstract computational model, as opposed to purely *representational* properties, which derive from the *nature of specific representations of the information units.*

3. It is not proper to describe the datatype model used herein as "mathematical", because a truly mathematical model has no notions of "access to information units" or "invocation of processing elements", and these notions are important to the definition of characterizing operations for datatypes and datatype generators.

6.3.1 Equality

In every value space there is a notion of **equality**, for which the following rules hold:

- for any two instances (a, b) of values from the value space, either a *is equal to* b, denoted $a = b$, or a *is not equal to* b, denoted $a \neq b$;
- there is no pair of instances (a, b) of values from the value space such that both $a = b$ and $a \neq b$;
- for every value a from the value space, $a = a$;
- for any two instances (a, b) of values from the value space, $a = b$ if and only if $b = a$;
- for any three instances (a, b, c) of values from the value space, if $a = b$ and $b = c$, then $a = c$.

On every datatype, the operation Equal is defined in terms of the equality property of the value space, by:

- for any values a, b drawn from the value space, Equal(a,b) **is** *true* if $a = b$, and *false* otherwise.

6.3.2 Order

A value space is said to be **ordered** if there exists for the value space an **order** relation, denoted $\leq$, with the following rules:

- for every pair of values (a, b) from the value space, either $a \leq b$ or $b \leq a$, or both;
- for any two values (a, b) from the value space, if $a \leq b$ and $b \leq a$, then $a = b$;
- for any three values (a, b, c) from the value space, if $a \leq b$ and $b \leq c$, then $a \leq c$.

For convenience, the notation a < b is used herein to denote the simultaneous relationships: a ≤ b and a ≠ b.

A datatype is said to be **ordered** if an order relation is defined on its value space. A corresponding characterizing operation, called InOrder, is then defined by:

- for any two values (a, b) from the value space, InOrder(a, b) is *true* if a ≤ b, and *false* otherwise.

NOTE — There may be several possible orderings of a given value space. And there may be several different datatypes which have a common value space, each using a different order relationship. The chosen order relationship is a characteristic of an ordered datatype and may affect the definition of other operations on the datatype.

6.3.3 Bound

A datatype is said to be **bounded above** if it is ordered and there is a value U in the value space such that, for all values s in the value space, s ≤ U. The value U is then said to be an **upper bound** of the value space. Similarly, a datatype is said to be **bounded below** if it is ordered and there is a value L in the space such that, for all values *s* in the value space, L ≤ s. The value L is then said to be a **lower bound** of the value space. A datatype is said to be **bounded** if its value space has both an upper bound and a lower bound.

NOTE — The upper bound of a value space, if it exists, must be unique under the equality relationship. For if U1 and U2 are both upper bounds of the value space, then U1 ≤ U2 and U2 ≤ U1, and therefore U1 = U2, following the second rule for the order relationship. And similarly the lower bound, if it exists, must also be unique.

On every datatype which is bounded below, the niladic operation Lowerbound is defined to yield that value which is the lower bound of the value space, and, on every datatype which is bounded above the niladic operation Upperbound is defined to yield that value which is the upper bound of the value space.

6.3.4 Cardinality

A value space has the mathematical concept of cardinality: it may be finite, denumerably infinite (countable), or non-denumerably infinite (uncountable). A datatype is said to have the cardinality of its value space. In the computational model, there are three significant cases:

- datatypes whose value spaces are finite,
- datatypes whose value spaces are exact (see 6.3.5) and denumerably infinite,
- datatypes whose value spaces are approximate (see 6.3.5), and therefore have a finite or denumerably infinite computational model, although the conceptual value space may be non-denumerably infinite.

Every conceptually finite datatype is necessarily exact. No computational datatype is non-denumerably infinite.

NOTE — For a denumerably infinite value space, there always exist representation algorithms such that no two distinct values have the same representation and the representation of any given value is of finite length. Conversely, in a non-denumerably infinite value space there always exist values which do not have finite representations.

6.3.5 Exact and approximate

The computational model of a datatype may limit the degree to which values of the datatype can be distinguished. If every value in the value space of the conceptual datatype is distinguishable in the computational model from every other value in the value space, then the datatype is said to be **exact**.

Certain mathematical datatypes having values which do not have finite representations are said to be **approximate**, in the following sense:

Let *M* be the mathematical datatype and *C* be the corresponding computational datatype, and let P be the mapping from the value space of *M* to the value space of *C*. Then for every value *v'* in *C*, there is a corresponding value *v* in *M* and a real value *h* such that P(*x*) = *v'* for all *x* in *M* such that | *v* - *x* | < *h*. That is, *v'* is the approximation in *C* to all values in *M* which are "within distance *h* of value *v*". Furthermore, for at least one value *v'* in *C*, there is more than one value *y* in *M* such that P(*y*) = *v'*. And thus *C* is *not* an exact model of *M*.

In this International Standard, all approximate datatypes have computational models which specify, via parametric values, a **degree of approximation**, that is, they require a certain minimum set of values of the mathematical datatype to be distinguishable in the computational datatype.

NOTE — The computational model described above allows a mathematically dense datatype to be mapped to a datatype with fixed-length representations and nonetheless evince intuitively acceptable mathematical behavior. When the real value *h* described above is constant over the value space, the computational model is characterized as having "bounded absolute error" and the result is a scaled datatype (8.1.9). When *h* has the form *c* • | v |, where *c* is constant over the value space, the computational model is characterized as having "bounded relative error", which is the model used for the Real (8.1.10) and Complex (8.1.11) datatypes.

6.3.6 Numeric

A datatype is said to be **numeric** if its values are conceptually quantities (in some mathematical number system). A datatype whose values do not have this property is said to be **non-numeric**.

NOTE — The significance of the numeric property is that the representations of the values depend on some *radix*, but can be algorithmically transformed from one radix to another.

6.4 Primitive and non-primitive datatypes

In this International Standard, datatypes are categorized, for syntactic convenience, into:

- **primitive** datatypes, which are defined ab initio without reference to other datatypes, and
- **generated** datatypes, which are specified, and partly defined, in terms of other datatypes.

In addition, this International Standard identifies structural and abstract notions of datatypes. The structural notion of a datatype characterizes the datatype as either:

- conceptually **atomic**, having values which are intrinsically indivisible, or
- conceptually **aggregate**, having values which can be seen as an organization of specific component datatypes with specific functionalities.

All primitive datatypes are conceptually atomic, and therefore have, and are defined in terms of, well-defined abstract notions. Some generated datatypes are conceptually atomic but are dependent on specifications which involve other datatypes. These too are defined in terms of their abstract notions. Many other datatypes may represent objects which are conceptually atomic, but are themselves conceptually aggregates, being organized collections of accessible component values. For aggregate datatypes, this International Standard defines a set of basic structural notions (see 6.8) which can be recursively applied to produce the value space of a given generated datatype. The only abstract semantics assigned to such a datatype by this International Standard are those which characterize the aggregate value structure itself.

NOTE — The abstract notion of a datatype is the semantics of the values of the datatype itself, as opposed to its utilization to represent values of a particular information unit or a particular abstract object. The abstract and structural notions provided by this International Standard are sufficient to define its role in the universe of discourse between two languages, but *not* to define its role in the universe of discourse between two *programs*. For example, Array datatypes are supported as such by both Fortran and Pascal, so that Array of Real has sufficient semantics for procedure calls between the two languages. By comparison, both linear operators and lists of Cartesian points may be represented by Array of Real, and Array of Real is insufficient to distinguish those meanings in the programs.

6.5 Datatype generator

A **datatype generator** is a conceptual operation on one or more datatypes which yields a datatype. A datatype generator operates on datatypes to generate a datatype, rather than on values to generate a value. Specifically, a datatype generator is the combination of:

- a collection of criteria for the number and characteristics of the datatypes to be operated upon,
- a construction procedure which, given a collection of datatypes meeting those criteria, creates a new value space from the value spaces of those datatypes, and
- a collection of characterizing operations which attach to the resulting value space to complete the definition of a new datatype.

The application of a datatype generator to a specific collection of datatypes meeting the criteria for the datatype generator forms a **generated datatype**. The generated dataype is sometimes called the **resulting** datatype, and the collection of datatypes to which the datatype generator was applied are called its **parametric datatypes**.

6.6 Characterizing operations

The set of **characterizing operations for a datatype** comprises those operations on or yielding values of the datatype which distinguish this datatype from other datatypes having value spaces which are identical except possibly for substitution of symbols.