



DPC: 24/30458756 DC

BSI Group Headquarters

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 16 August 2024

Origin: International

Latest date for receipt of comments: 09 October 2024

Project No. 2022/01587

Responsible committee: IST/40 Data management and interchange

Interested committees:

Title: Draft BS EN ISO/IEC 11404 BS EN ISO/IEC 11404 Information technology - General-Purpose Datatypes (GPD).

Please notify admin.start@bsigroup.com if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 09 October 2024

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email cservices@bsigroup.com. British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

E-mail: admin.start@bsigroup.com

Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

Template for comments and secretariat observations

Date: xx/xx/20xx	Document: ISO/DIS xxxx
------------------	------------------------

1	2	(3)	4	5	(6)	7
MB	Clause No./ Subclause No./Annex (e.g. 3.1)	Paragraph/ Figure/Table/Note	Type of comment	Comment (justification for change) by the MB	Proposed change by the MB	Secretariat observations on each comment submitted
	3.1	Definition 1	ed	Definition is ambiguous and needs clarifying.	Amend to read '...so that the mains connector to which no connection...'	
	6.4	Paragraph 2	te	The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK.	Delete reference to UV photometer.	



DRAFT International Standard

ISO/IEC DIS 11404

Information technology — General-Purpose Datatypes (GPD)

Technologies de l'information — Types de données à but général (GPD)

ICS: 35.060

ISO/IEC JTC 1/SC 32

Secretariat: **ANSI**

Voting begins on:
2024-08-16

Voting terminates on:
2024-11-08

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number
ISO/IEC DIS 11404:2024(en)

Full document details: 24/30458756 DC



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

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

Published in Switzerland

Contents

Page

Foreword	vi
0 Introduction.....	vii
1 Scope	1
2 Normative references	2
3 Terms and definitions	2
4 Conformance	8
4.1 Direct conformance.....	8
4.2 Indirect conformance	9
4.3 Conformance of a mapping standard.....	9
4.4 GPD program conformance	10
5 Conventions used in this International Standard	10
5.1 Formal syntax	10
5.2 Text conventions	11
6 Fundamental notions	11
6.1 Datatype	11
6.2 Value space.....	12
6.3 Datatype properties.....	12
6.3.1 Equality.....	13
6.3.2 Order	13
6.3.3 Bound	13
6.3.4 Cardinality	14
6.3.5 Exact and approximate	14
6.3.6 Numeric	15
6.4 Primitive and non-primitive datatypes	15
6.5 Datatype generator.....	15
6.6 Characterizing operations	16
6.7 Datatype families	17
6.8 Aggregate datatypes	17
6.8.1 Homogeneity	17
6.8.2 Size.....	18
6.8.3 Uniqueness	18
6.8.4 Aggregate-imposed identifier uniqueness	18
6.8.5 Aggregate-imposed ordering	18
6.8.6 Access method	18
6.8.7 Recursive structure.....	19
6.8.8 Structured and unstructured.....	19
6.8.9 Mandatory and optional components	19
6.9 Provisions associated with datatypes	20
7 Elements of the Datatype Specification Language	21
7.1 IDN character-set.....	22
7.2 Whitespace.....	23
7.3 Lexical objects.....	23
7.3.1 Identifiers	23
7.3.2 Digit-string	24
7.3.3 Character-literal and string-literal.....	24
7.3.4 Keywords	24
7.4 Annotations.....	25
7.5 Values	25

7.5.1	Independent values	25
7.5.2	Dependent values	26
7.6	GPD program text	27
8	Datatypes	28
8.1	Primitive datatypes	28
8.1.1	Boolean	30
8.1.2	State	30
8.1.3	Enumerated	31
8.1.4	Character	32
8.1.5	Ordinal	34
8.1.6	Date-and-Time	34
8.1.7	Integer	36
8.1.8	Rational	36
8.1.9	Scaled	37
8.1.10	Real	39
8.1.11	Float	41
8.1.12	Complex	43
8.1.13	Void	44
8.2	Subtypes and extended types	45
8.2.1	Range	45
8.2.2	Selecting	46
8.2.3	Excluding	46
8.2.4	Size	47
8.2.5	Explicit subtypes	48
8.2.6	Extended	48
8.3	Generated datatypes	49
8.3.1	Choice	50
8.3.2	Pointer	52
8.3.3	Procedure	53
8.4	Aggregate Datatypes	57
8.4.1	Record	58
8.4.2	Class	60
8.4.3	Set	61
8.4.4	Bag	62
8.4.5	Sequence	64
8.4.6	Array	65
8.4.7	Table	68
8.5	Defined datatypes	69
8.6	Provisions	70
8.6.1	General parameters for provisions	70
8.6.2	Aggregate-specific features	74
8.6.3	Aggregate-component-identifier uniqueness	74
8.6.4	Usage-specific features	74
9	Declarations	75
9.1	Type declarations	76
9.1.1	Renaming declarations	76
9.1.2	New datatype declarations	77
9.1.3	New generator declarations	77
9.2	Value declarations	77
9.3	Termination declarations	77
9.4	Normative datatype declarations	78
9.5	Lexical operations	78
9.5.1	Import	78
9.5.2	Macro	79
10	Defined datatypes and generators	79
10.1	Defined datatypes	79
10.1.1	Natural number	80
10.1.2	Modulo	80

10.1.3	Bit	81
10.1.4	Bit string	81
10.1.5	Character string	81
10.1.6	Time interval	83
10.1.7	Octet	83
10.1.8	Octet string	83
10.1.9	Private	84
10.1.10	Object identifier	84
10.2	Defined generators	86
10.2.1	Stack	86
10.2.2	Tree	87
10.2.3	Optional	88
11	Mappings	89
11.1	Outward Mappings	89
11.2	Inward Mappings	90
11.3	Reverse Inward Mapping	91
11.4	Support of Datatypes	92
11.4.1	Support of equality	92
11.4.2	Support of order	92
11.4.3	Support of bounds	92
11.4.4	Support of cardinality	92
11.4.5	Support for the exact or approximate property	93
11.4.6	Support for the numeric property	93
11.4.7	Support for the mandatory components	93
Annex A	(informative) Character-set standards	94
Annex B	(informative) Recommendation for the placement of annotations	96
11.5	B.1 Type-attributes	96
11.6	B.2 Component-attributes	96
11.7	B.3 Procedure-attributes	97
11.8	B.4 Argument-attributes	97
12	Annex C: (informative) Implementation notions of datatypes	98
12.1	C.1 StorageSize	98
12.2	C.2 Mode	98
12.3	C.3 Floating-Point	99
12.4	C.4 Fixed-Point	99
12.5	C.5 Tag	100
12.6	C.6 Discriminant	100
12.7	C.7 StorageSequence	100
12.8	C.8 Packed	100
12.9	C.9 Alignment	100
12.10	C.10 Form	100

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 the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member 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 shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 11404 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This third edition cancels and replaces the second edition (ISO/IEC 11404:2007), which has been technically revised.

0 Introduction

0.1 Introduction to the third edition

The "float" datatype has been added to match the computational environments available in modern systems, see 8.1.11. In addition, the float type of various precisions to can support optimized Machine Learning (ML) calculations, classifications, and inferences.

0.2 Introduction to the second edition

This second edition of ISO/IEC 11404 incorporates recent technologies and improvements since the first edition (ISO/IEC 11404:1996). The following improvements have been incorporated into the second edition.

— Title change to reflect actual usage. The use of ISO/IEC 11404 is no longer simply a tool for communicating among programming languages (old title: *Language-independent datatypes*). ISO/IEC 11404 is used for formal description of conceptual datatypes in binding (or binding-independent) standards and used as formalization of metadata for data elements, data element concepts, and value domains (see ISO/IEC 11179-3). The old title was potentially misleading because readers might believe that ISO/IEC 11404 is only useful for programming languages. The new title, *General-Purpose Datatypes* captures the essence of ISO/IEC 11404 and its use.

— Incorporation of latest technologies. Provide enhancements to the use of ISO/IEC 11404 as a datatype nomenclature reference for current programming languages, interface languages and data representation languages, specifically Java, IDL, Express, and XML.

— Support for semi-structured and unstructured data aggregates. Semi-structured data and unstructured data includes aggregates where datatyping and navigation may be unknown or unspecified in advance. For example, some systems permit "discovery" (or "introspection") of data. In some cases, the datatype may be unknown in advance (e.g. at compilation time), but may be discovered and processed at runtime (e.g. via datatype libraries or metadata registries).

— Support for data longevity, versioning, and migration. There is a need to support, from a datatyping perspective, obsolete and reserved features, such as data elements and permissible values (enumerations and states). Marking features as "obsolete" allows processing, compilation, and runtime systems to "flag" or diagnose old (deprecated) features, while still maintaining compatibility, so that it is possible to support transitions from past to present. Similarly, marking features as "reserved" allows processing, compilation, and runtime systems to "flag" or diagnose potential incompatibilities with future systems, so that it is possible to support transitions from present to future.

— Extensibility of datatypes and value spaces. There is a need to support some kind of extensibility concept. For example: (1) a GPD specification of an aggregate contains the elements A and B. (2) An application creates an aggregate with the elements A, B, and C. (3) Are the application's "extensions" of the aggregate acceptable/in conformity with the GPD specification in (1)? The answer to (3) is dependent upon the intent and design of the specification in (1): in some cases extensions are permitted, in some cases extensions are not permitted. The extensibility concept would allow the user of GPD datatypes to describe the kind of extensions permitted. This feature is particularly important in (a) data conformance and (b) application runtime environments that permit "discovery" or "introspection". This feature is available via the "provision()" capability.

Features that are not incorporated within GPD include the following:

— Namespace capability. Given the larger number of declarations, a namespace capability is necessary.

— Data representation. Although these features are a part of GPD annotations, there is no standardization of data representation in these annotations. This step is an important link for data interoperability.

0.3 Introduction to the first edition (ISO/IEC 11404:1996)

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 JTC 1, *Information technology* assigned to SC 22 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.

110 Information technology — General-Purpose Datatypes (GPD)

111 1 Scope

112 This International Standard specifies the nomenclature and shared semantics for a collection of datatypes
 113 commonly occurring in programming languages and software interfaces, referred to as the General-
 114 Purpose Datatypes (GPD). It specifies both primitive datatypes, in the sense of being defined *ab initio*
 115 without reference to other datatypes, and non-primitive datatypes, in the sense of being wholly or partly
 116 defined in terms of other datatypes. The specification of datatypes in this International Standard is
 117 "general-purpose" in the sense that the datatypes specified are classes of datatype of which the actual
 118 datatypes used in programming languages and other entities requiring the concept "datatype" are
 119 particular instances. These datatypes are general in nature; thus, they serve a wide variety of information
 120 processing applications.

121 This International Standard expressly distinguishes three notions of datatype:

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

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

134 This International Standard defines a partial terminology for implementation notions of datatypes and
 135 provides for the use of this terminology in the definition of datatypes. The primary purpose of this
 136 terminology is to identify common implementation notions associated with datatypes and to distinguish
 137 them from conceptual notions.

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

141

142

143

144

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 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 8824 (all parts), *Information technology — Abstract Syntax Notation One (ASN.1)*

ISO/IEC 10646, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 13886, *Information technology — Language Independent Procedure Calling (LIPC)*

ISO/IEC 14977, *Information technology — Syntactic metalanguage — Extended BNF*

IETF RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*

3 Terms and definitions

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

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

3.1

actual parametric datatype

datatype appearing as a parametric datatype in a use of a datatype generator, in contrast to the formal-parametric-types appearing in the definition of the datatype generator

3.2

actual parametric value

value appearing as a parametric value in a reference to a datatype family or datatype generator, in contrast to the formal-parametric-values appearing in the corresponding definitions

3.3

aggregate datatype

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

3.4

annotation

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

3.5

approximate

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

3.6

bounded

property of a datatype, meaning both bounded above and bounded below

- 184 **3.7**
 185 **bounded above**
 186 property of a datatype indicating that there is a value U in the value space such that, for all values s in
 187 the value space, $s \leq U$
- 188 **3.8**
 189 **bounded below**
 190 property of a datatype indicating that there is a value L in the value space such that, for all values s in
 191 the value space, $s \geq L$
- 192 **3.9**
 193 **characterizing operations**
 194 $\langle \text{datatype} \rangle^1$ collection of operations on, or yielding, values of the datatype that distinguish this datatype
 195 from other datatypes with identical value spaces
- 196 **3.10**
 197 **characterizing operations**
 198 $\langle \text{datatype generator} \rangle$ collection of operations on, or yielding, values of any datatype resulting from an
 199 application of the datatype generator that distinguish this datatype generator from other datatype
 200 generators and produce identical value spaces from identical parametric datatypes
- 201 **3.11**
 202 **component datatype**
 203 datatype which is a parametric datatype to a datatype generator
- 204 NOTE A component datatype is a datatype on which the datatype generator operates.
- 205 **3.12**
 206 **datatype**
 207 set of distinct values, characterized by properties of those values, and by operations on those values
- 208 **3.13**
 209 **datatype declaration**
 210 means provided by this International Standard for the definition of a datatype which is not itself defined by
 211 this International Standard
- 212 **3.14**
 213 **datatype family**
 214 collection of datatypes which have equivalent characterizing operations and relationships, but value
 215 spaces that differ in the number and identification of the individual values
- 216 **3.15**
 217 **datatype generator**
 218 **generator**
 219 operation on datatypes, as objects distinct from their values, that generates new datatypes
- 220 **3.16**
 221 **defined datatype**
 222 datatype defined by a type-declaration.
- 223 **3.17**
 224 **defined generator**
 225 datatype generator defined by a type-declaration

1) Angle brackets indicate the subject field to which the concept belongs, in accordance with ISO 10241.

226	3.18
227	exact
228	property of a datatype indicating that every value of the conceptual datatype is distinct from all others in
229	any valid computational model of the datatype
230	3.19
231	formal-parametric-type
232	identifier, appearing in the definition of a datatype generator, for which a datatype will be substituted in
233	any reference to a (defined) datatype resulting from the generator
234	3.20
235	formal-parametric-value
236	identifier, appearing in the definition of a datatype family or datatype generator, for which a value will be
237	substituted in any reference to a (defined) datatype in the family or resulting from the generator
238	3.21
239	general-purpose datatype
240	GPD
241	datatype defined by this International Standard
242	3.22
243	GPD-generated datatype
244	GPD datatype
245	datatype defined by the means of datatype definition provided by this International Standard
246	NOTE Although "GPD datatype" expands to "general-purpose datatype datatype" and might appear redundant, it
247	is to be read as "general-purpose-datatype datatype", where GPD is an adjective and datatype (standalone) is a
248	noun.
249	3.23
250	generated datatype
251	datatype defined by the application of a datatype generator to one or more previously-defined datatypes
252	3.24
253	generated internal datatype
254	datatype defined by the application of a datatype generator defined in a particular programming language
255	to one or more previously-defined internal datatypes
256	3.25
257	generator declaration
258	means provided by this International Standard for the definition of a datatype generator which is not itself
259	defined by this International Standard
260	3.26
261	instruction
262	provision that conveys an action to be performed
263	[ISO/IEC Guide 2]
264	3.27
265	internal datatype
266	datatype whose syntax and semantics are defined by some other standard, specification, language,
267	product, service or other information processing entity
268	3.28
269	inward mapping
270	conceptual association between the internal datatypes of a language and the general-purpose datatypes
271	which assigns to each general-purpose datatype either a single semantically equivalent internal datatype
272	or no equivalent internal datatype

273	3.29	
274	lower bound	
275	value L such that, for all values s in the value space in a datatype which is bounded below, $L \leq s$	
276	3.30	
277	mandatory requirement	
278	requirement of a normative document that must necessarily be fulfilled in order to comply with that	
279	document [adapted from ISO/IEC Guide 2]	
280	NOTE	A "mandatory requirement" is also known as an "exclusive requirement".
281	3.31	
282	mapping	
283	⟨datatypes⟩ formal specification of the relationship between the internal datatypes that are notions of, and	
284	specifiable in, a particular programming language and the general-purpose datatypes specified in this	
285	International Standard	
286	3.32	
287	mapping	
288	⟨values⟩ corresponding specification of the relationships between values of the internal datatypes and	
289	values of the general-purpose datatypes	
290	3.33	
291	meta-identifier	
292	name of a non-terminal symbol	
293	[ISO/IEC 14977]	
294	NOTE	See note in 5.1 concerning the context of the specialized usage of this term for describing the syntax of
295	ISO/IEC 11404 program text.	
296	3.34	
297	non-terminal symbol	
298	⟨EBNF⟩ syntactic part of the language being defined	
299	[ISO/IEC 14977]	
300	NOTE	See note in 5.1 concerning the context of the specialized usage of this term.
301	3.35	
302	normative datatype	
303	collection of specifications for datatype properties that may be simultaneously satisfied by more than one	
304	actual datatype	
305	3.36	
306	normative document	
307	document that provides rules, guidelines or characteristics for activities or their results	
308	[ISO/IEC Guide 2]	
309	NOTE 1	The term "normative document" is a generic term that covers such documents as standards and technical
310	specifications.	
311	NOTE 2	A "document" is to be understood as any medium with information recorded on or in it, such as a paper
312	document or program code.	
313	3.37	
314	optional requirement	
315	requirement of a normative document that must be fulfilled in order to comply with a particular option	
316	permitted by that document	

317 [ISO/IEC Guide 2]

318 NOTE An optional requirement may be either (1) one of two or more alternative requirements; or (2) an
319 additional requirement that must be fulfilled only if applicable and that may otherwise be disregarded.

320 **3.38**
321 **order**
322 mathematical relationship among values

323 NOTE See 6.3.2.

324 **3.39**
325 **ordered**
326 property of a datatype which is determined by the existence and specification of an order relationship on
327 its value space

328 **3.40**
329 **outward mapping**
330 conceptual association between the internal datatypes of a language and the general-purpose datatypes
331 that identifies each internal datatype with a single semantically equivalent general-purpose datatype

332 **3.41**
333 **parametric datatype**
334 datatype on which a datatype generator operates to produce a generated datatype

335 **3.42**
336 **parametric value (1)**
337 value which distinguishes one member of a datatype family from another

338 **3.43**
339 **parametric value (2)**
340 value which is a parameter of a datatype or datatype generator defined by a type-declaration

341 NOTE See 9.1.

342 **3.44**
343 **primitive datatype**
344 identifiable datatype that cannot be decomposed into other identifiable datatypes without loss of all
345 semantics associated with the datatype

346 **3.45**
347 **primitive internal datatype**
348 datatype in a particular programming language whose values, conceptually, are not constructed in any
349 way from values of other datatypes in the language

350 **3.46**
351 **provision**
352 expression of normative wording that takes the form of a statement, an instruction, a recommendation or a
353 requirement

354 NOTE 1 Adapted from ISO/IEC Guide 2.

355 NOTE 2 These types of provision are distinguished by the form of wording they employ; e.g. instructions are
356 expressed in the imperative mood, recommendations by the use of the auxiliary "should" and requirements by the use
357 of the auxiliary "shall".

358 **3.47**
359 **recommendation**
360 provision that conveys advice or guidance

361	[ISO/IEC Guide 2]
362	3.48
363	regular value
364	element of a value space that is consistent with a datatype's properties and characterizing operations
365	3.49
366	representation
367	⟨general-purpose datatype⟩ mapping from the value space of the general-purpose datatype to the value
368	space of some internal datatype of a computer system, file system or communications environment
369	3.50
370	representation
371	⟨value⟩ sign(s) of that value in the representation of the datatype
372	NOTE In this context, the term "sign" is used in its terminological sense (i.e. a symbol) and not in its
373	mathematical sense (i.e. positive or negative).
374	3.51
375	requirement
376	provision that conveys criteria to be fulfilled
377	[ISO/IEC Guide 2]
378	3.52
379	sentence
380	⟨EBNF⟩ sequence of symbols that represents the start symbol
381	[ISO/IEC 14977]
382	NOTE See note in 5.1 concerning the context of the specialized usage of this term.
383	3.53
384	sentinel value
385	element of a value space that is not completely consistent with a datatype's properties and characterizing
386	operations
387	NOTE A numeric datatype, which includes characterizing operations such as Equal and InOrder , may include
388	sentinel values such as not-a-number , indeterminate , not-applicable , +infinity , -infinity and so on. These
389	characterizing operations are not defined for sentinel values.
390	3.54
391	sequence
392	⟨EBNF⟩ ordered list of zero or more items
393	[ISO/IEC 14977]
394	NOTE See note in 5.1 concerning the context of the specialized usage of this term.
395	3.55
396	start symbol
397	⟨EBNF⟩ non-terminal symbol that is defined by one or more syntax rules but does not occur in any other
398	syntax rule
399	[ISO/IEC 14977]
400	NOTE See note in 5.1 concerning the context of the specialized usage of this term.

401 **3.56**
402 **statement**
403 provision that conveys information

404 [ISO/IEC Guide 2]

405 **3.57**
406 **subsequence**
407 〈EBNF〉 sequence within a sequence

408 [ISO/IEC 14977]

409 NOTE See note in 5.1 concerning the context of the specialized usage of this term.

410 **3.58**
411 **subtype**
412 datatype derived from another datatype by restricting the value space to a subset whilst maintaining all
413 characterizing operations

414 **3.59**
415 **terminal symbol**
416 〈EBNF〉 sequence of one or more characters forming an irreducible element of a language

417 [ISO/IEC 14977]

418 NOTE See note in 5.1 on the context of the specialized usage of this term.

419 **3.60**
420 **upper bound**
421 value U such that, for all values s in the value space in a datatype which is bounded above, $s \leq U$

422 **3.61**
423 **value space**
424 set of values for a given datatype

425 **3.62**
426 **variable**
427 computational object to which a value of a particular datatype is associated at any given time; and to
428 which different values of the same datatype may be associated at different times

429 **4 Conformance**

430 An information processing product, system, element or other entity may conform to this International
431 Standard either directly, by utilizing datatypes specified in this International Standard in a conforming
432 manner (4.1), or indirectly, by means of mappings between internal datatypes used by the entity and the
433 datatypes specified in this International Standard (4.2).

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

438 **4.1 Direct conformance**

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

1. specify which of the datatypes and datatype generators specified in Clause 8 and Clause 10 are provided by the entity and which are not, and which, if any, of the declaration mechanisms in Clause 9 it provides; and
2. define the value spaces of the general-purpose datatypes used by the entity to be identical to the value spaces specified by this International Standard; and
3. use the notation prescribed by Clause 7 through Clause 10 of this International Standard to refer to those datatypes and to no others; and
4. to the extent that the entity provides operations other than movement or transformation of values, define operations on the general-purpose datatypes which can be derived from, or are otherwise consistent with, the characterizing operations specified by this International Standard.

NOTE 1 This International Standard defines a syntax for the denotation of values of each datatype it defines, but, in general, requirement 3 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.

NOTE 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.

NOTE 3 Requirement 4 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 general-purpose 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.

NOTE 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.

4.2 Indirect conformance

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

1. provide mappings between its internal datatypes and the general-purpose datatypes conforming to the specifications of Clause 11 of this International Standard; and
2. 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.

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

NOTE 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.

4.3 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.

NOTE 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 general-purpose datatypes. Such mapping standards are required to comply with this International Standard.

NOTE 2 Such mapping standards may define "generic" mappings, in the sense that for a given internal datatype the standard specifies a parameterized general-purpose 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 general-purpose datatype, the mapping specifies a family of general-purpose 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 parameterized. 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 general-purpose datatype:

`integer range (min..max)`

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

4.4 GPD program conformance

A GDP conforming program² is a specification that uses datatypes and datatype values and their syntaxes as specified in this International Standard. Such a specification may be self-contained: there is no requirement for the existence of a GDP conformant implementation that can produce or operate on the specification. The requirements of this International Standard to which the specification conforms shall be clearly identified, either in the specification itself, or in documentation that is unambiguously identified in the specification.

NOTE A GDP conforming program is a special case of directly conforming entity.

5 Conventions used in this International Standard

5.1 Formal syntax

This International Standard defines a formal datatype specification language. The notation defined in ISO/IEC 14977, Extended Backus-Naur Form (EBNF), is used in defining that language. Table 5-1 summarizes the ISO/IEC 14977 EBNF syntactic metalanguage.

NOTE The terms meta-identifier, non-terminal symbol, sentence, sequence, start symbol, subsequence, and terminal symbol have special meaning in the context of EBNF notation (see Clause 3, Definitions).

Table 5-1 — Summary of ISO/IEC 14977 EBNF Syntactic Metalanguage Notation

Representation	ISO/IEC 10646 Character Names	Metalanguage Symbol
' '	apostrophe	first quote symbol
" "	quotation mark	second quote symbol
(* *)	left parenthesis with asterisk, asterisk with right parenthesis	start/end comment symbols
()	left parenthesis, right parenthesis	start/end group symbols
[]	left square bracket, right square bracket	start/end option symbols
{ }	left curly bracket, right curly bracket	start/end repeat symbols
? ?	question mark	special sequence symbol
-	hyphen-minus	except symbol
,	comma	concatenate symbol
=	equals sign	defining symbol
	vertical line	definition separator symbol
*	asterisk	repetition symbol
;	semicolon	terminator symbol

² A GDP conforming program might be an 11404 GPD datatype definition or a data declaration based upon an 11404 GPD datatype.

512 EXAMPLE 1 The following syntax rules illustrate repetition (asterisk and curly brackets) and option square brackets:

```
513 aa = "A" ;
514 bb = 3 * aa, "B" ;
515 cc = 3 * [aa], "C" ;
516 dd = {aa}, "D" ;
517 ee = aa, {aa}, "E" ;
518 ff = 3 * aa, 3 * [aa], "F" ;
```

519 Terminal strings defined by these rules are as follows:

```
521 aa: A
522 bb: AAAB
523 cc: C AC AAC AAAC
524 dd: D AD AAD AAAD AAAAD etc.
525 ee: AE AAE AAEE AAAEE AAAAAE etc.
526 ff: AAFF AAAFF AAAAAF AAAAAAF
```

527 EXAMPLE 2 The following syntax rules illustrate a definitions list (vertical line), an exception (hyphen-minus), and comments (parentheses and asterisks):

```
530 letter = "A" | "B" | "C" | "D" | "E" | "F"
531 | "G" | "H" | "I" | "J" | "K" | "L" | "M"
532 | "N" | "O" | "P" | "Q" | "R" | "S" | "T"
533 | "U" | "V" | "W" | "X" | "Y" | "Z" ;
534 vowel = "A" | "E" | "I" | "O" | "U" ;
535 consonant = letter - vowel ; (* These examples are from ISO/IEC 14977 *)
```

536 Terminal strings defined by these rules are as follows:

```
538 letter: A B C D E F G H I J etc.
539 vowel: A E I O U
540 consonant: B C D F G H J K L M etc.
```

542 5.2 Text conventions

543 Within the text:

- 544 — A reference to a terminal symbol syntactic object consists of the terminal symbol in fixed width courier,
545 e.g. `type`.
- 546 — A reference to a non-terminal symbol syntactic object consists of the non-terminal-symbol in fixed
547 width italic courier, e.g. *type-declaration*.
- 548 — Mathematical notation, properties, and characterizing operations are in bold, e.g., **InOrder(x,y)**.
- 549 — Non-italicized words which are identical or nearly identical in spelling to a non-terminal-symbol refer to
550 the conceptual object represented by the syntactic object. In particular, **xxx-type** refers to the
551 syntactic representation of an "xxx datatype" in all occurrences.

552 6 Fundamental notions

553 6.1 Datatype

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

558 The term general-purpose datatype is used to mean a datatype defined by this International Standard.
 559 The term general-purpose datatypes (plural) refers to some or all of the datatypes defined by this
 560 International Standard. The term GPD datatype refers to datatypes generated or defined by means
 561 specified in this International Standard.

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

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

567 6.2 Value space

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

570 — enumerated outright, or

571 — defined axiomatically from fundamental notions, such as ontologies, or

572 — defined as the subset of those values from some already defined value space which have a given set
 573 of properties, or

574 — defined as a combination of arbitrary values from some already defined value spaces by a specified
 575 construction procedure.

576 NOTE 1 This International Standard defines the concept "value space", which is just a set of values. It extends
 577 that notion to "datatype" by adding computational properties supported by characterizing operations. ISO/IEC 11179,
 578 *Information technology — Metadata Registries (MDR)*, introduces the concept "value domain". A "value domain" is a
 579 set of <value, meaning> pairs, each pair consisting of a value and its conceptual interpretation. That is, ISO/IEC
 580 11179 extends the notion value space to "value domain" by adding its meaning for users and applications.

581 A distinct value may belong to the value space of more than one datatype, so long as it properly supports
 582 the properties and characterizing operations of each of them (see 6.6).

583 A value space contains regular values (elements of a value space that are consistent with a datatype's
 584 properties and characterizing operations). A datatype may also have sentinel values: elements that can
 585 be said to 'belong' to the datatype but that may not be completely consistent with the properties and
 586 characterizing operations of the datatype. For the purpose of this International Standard, sentinel values
 587 do not belong to the value space of the datatype. If a datatype has sentinel values, then there shall
 588 always be a form of the Equal operator to distinguish these sentinel values from regular values (see also
 589 8.2.6).

590 NOTE 2 A numeric datatype, which includes characterizing operations such as **Equal** and **InOrder**, may include
 591 sentinel values such as **not-a-number**, **indeterminate**, **not-applicable**, **+infinity**, **-infinity**, and so on. These
 592 characterizing operations are not defined for sentinel values.

593 6.3 Datatype properties

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

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

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

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

609 6.3.1 Equality

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

611 — for any two instances (**a**, **b**) of values from the value space, either **a** *is equal to* **b**, denoted $a = b$, or **a**
612 *is not equal to* **b**, denoted $a \neq b$;

613 — there is no pair of instances (**a**, **b**) of values from the value space such that both $a = b$ and $a \neq b$;

614 — for every value **a** from the value space, $a = a$;

615 — for any two instances (**a**, **b**) of values from the value space, $a = b$ if and only if $b = a$;

616 — for any three instances (**a**, **b**, **c**) of values from the value space, if $a = b$ and $b = c$, then $a = c$.

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

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

619 6.3.2 Order

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

622 — for every pair of values (**a**, **b**) from the value space, either $a \leq b$ or $b \leq a$, or both;

623 — for any two values (**a**, **b**) from the value space, if $a \leq b$ and $b \leq a$, then $a = b$;

624 — for any three values (**a**, **b**, **c**) from the value space, if $a \leq b$ and $b \leq c$, then $a \leq c$.

625 For convenience, the notation $a < b$ is used herein to denote the simultaneous relationships: $a \leq b$ and
626 $a \neq b$.

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

629 — for any two values (**a**, **b**) from the value space, **InOrder(a, b)** is *true* if $a \leq b$, and *false* otherwise.

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

634 6.3.3 Bound

635 A datatype is said to be **bounded above** if it is ordered and there is a value U in the value space such that,
636 for all values s in the value space, $s \leq U$. The value U is then said to be an **upper bound** of the value
637 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, $L \leq s$. 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 $U1 \leq U2$ and $U2 \leq U1$, 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 evinces 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 $c \cdot |v|$, 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 (8.1.12) 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 axiomatically 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.

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

725 6.6 Characterizing operations

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

729 The set of characterizing operations for a datatype generator comprises those operations on, or yielding
 730 values of, any datatype resulting from an application of the datatype generator that distinguish this
 731 datatype generator from other datatype generators which produce identical value spaces from identical
 732 parametric datatypes.

733 NOTE 1 Characterizing operations are needed to distinguish datatypes whose value spaces differ only in what the
 734 values are called. For example, the value spaces (**one, two, three, four**), (**1, 2, 3, 4**), and (**red, yellow, green, blue**) all
 735 have four distinct values and all the names (symbols) are different. But one can claim that the first two support the
 736 characterizing operation **Add()**, while the last does not:

737 $Add(one, two) = three$; and $Add(1,2) = 3$; but $Add(red, yellow) \neq green$

738 It is this characterizing operation (**Add**) which enables one to recognize that the first two datatypes are the same
 739 datatype, while the last is a different datatype.

740 NOTE 2 The characterizing operations for an aggregate datatype are compositions of characterizing operations for
 741 its datatype generator with characterizing operations for its component datatypes. Such operations are, of course,
 742 only sufficient to identify the datatype as a structure.

743 NOTE 3 The characterizing operations on a datatype may be:

744 — niladic operations which yield values of the given datatype,

745 — monadic operations which map a value of the given datatype into a value of the given datatype or into a value of
 746 datatype Boolean,

747 — dyadic operations which map a pair of values of the given datatype into a value of the given datatype or into a
 748 value of datatype Boolean,

749 — n-adic operations³ which map ordered n-tuples of values, each of which is of a specified datatype, which may be
 750 the given datatype or a parametric datatype, into values of the given datatype or a parametric datatype.

751 NOTE 4 In general, there is no unique collection of characterizing operations for a given datatype. This
 752 International Standard specifies one collection of characterizing operations for each datatype (or datatype generator)
 753 which is sufficient to distinguish the (resulting) datatype from all other datatypes with value spaces of the same
 754 cardinality. While some effort has been made to minimize the collection of characterizing operations for each
 755 datatype, no assertion is made that any of the specified collections is minimal.

756 NOTE 5 **Equal** is always a characterizing operation on datatypes with the equality property.

757 NOTE 6 **InOrder** is always a characterizing operation on ordered datatypes (see 6.3.2).

758 Characterizing operations should be expressed in the syntax of ISO/IEC 13886.

³ The term "n-adic" is a general term, which includes niladic, monadic, and dyadic.

6.7 Datatype families

If there is a 1-to-1 symbol substitution which maps the entire value space of one datatype (the *domain*) into a subset of the value space of another datatype (the *range*) in such a way that the value relationships and characterizing operations of the domain datatype are preserved in the corresponding value relationships and characterizing operations of the range datatype, and if there are no additional characterizing operations on the range datatype, then the two datatypes are said to belong to the same family of datatypes. An individual member of a family of datatypes is distinguished by the symbol set making up its value space. In this International Standard, the symbol set for an individual member of a datatype family is specified by one or more values, called the parametric values of the datatype family.

6.8 Aggregate datatypes

An aggregate datatype is a generated datatype, each of whose values is, in principle, made up of values of the parametric datatypes. The parametric datatypes of an aggregate datatype or its generator are also called component datatypes. An aggregate datatype generator generates a datatype by

- applying an algorithmic procedure to the value spaces of its component datatypes to yield the value space of the aggregate datatype, and

- providing a set of characterizing operations specific to the generator.

Unlike other generated datatypes, it is characteristic of aggregate datatypes that the component values of an aggregate value are accessible through characterizing operations.

Aggregate datatypes of various kinds are distinguished one from another by properties which characterize relationships among the component datatypes and relationships between each component and the aggregate value. This subclause defines those properties.

The properties specific to an aggregate are independent of the properties of the component datatypes. (The fundamental properties of arrays, for example, do not depend on the nature of the elements.) In principle, any combination of the properties specified in this subclause defines a particular form of aggregate datatype, although most are only meaningful for homogeneous aggregates (see 6.8.1) and there are implications of some direct access methods (see 6.8.6).

6.8.1 Homogeneity

An aggregate datatype is *homogeneous*, if and only if all components must belong to a single datatype. If different components may belong to different datatypes, the aggregate datatype is said to be *heterogeneous*. The component datatype of a homogeneous aggregate is also called the *element datatype*.

NOTE 1 Homogeneous aggregates view all their elements as serving the same role or purpose. Heterogeneous aggregates divide their elements into different roles.

NOTE 2 The aggregate datatype is homogeneous if its components all belong to the same datatype, even if the element datatype is itself an heterogeneous aggregate datatype. Consider the datatype `label_list` defined by:

```
type label = choice (state(name, handle)) of ((name): characterstring, (handle): integer);
type label_list = sequence of (label);
```

Formally, a `label_list` value is a homogeneous series of label values. One could argue that it is really a series of heterogeneous values, because every label value is of a choice datatype (see 8.3.1). The `choice` datatype generator is clearly heterogeneous because it is capable of introducing variation in element type. But `sequence` (see 8.4.4) is homogeneous because it itself introduces no variation in element type.

801 **6.8.2 Size**

802 The **size** of an aggregate-value is the number of component values it contains. The size of the aggregate
 803 datatype is fixed, if and only if all values in its value space contain the same number of component values.
 804 The size is variable, if different values of the aggregate datatype may have different numbers of
 805 component values. Variability is the more general case; fixed-size is a constraint.

806 **6.8.3 Uniqueness**

807 An aggregate-value has the uniqueness property if and only if no value of the element datatype occurs
 808 more than once in the aggregate-value. The aggregate datatype has the uniqueness property, if and only
 809 if all values in its value space do.

810 **6.8.4 Aggregate-imposed identifier uniqueness**

811 An aggregate-value has the identifier uniqueness property if and only if no identifier (e.g., label, index) of
 812 the element datatype occurs more than once in the aggregate-value. The aggregate datatype has the
 813 identifier uniqueness property, if and only if all values in its value space do.

814 **6.8.5 Aggregate-imposed ordering**

815 An aggregate datatype has the ordering property, if and only if there is a canonical first element of each
 816 non-empty value in its value-space. This ordering is (externally) imposed by the aggregate value, as
 817 distinct from the value-space of the element datatype itself being (internally) **ordered** (see 6.3.2). It is also
 818 distinct from the value-space of the aggregate datatype being ordered.

819 EXAMPLE The type-generator **sequence** has the ordering property. The datatype **characterstring** is defined as
 820 **sequence of (character(repertoire))**. The ordering property of **sequence** means that in every value of type
 821 **characterstring**, there is a first character value. For example, the first element value of the **characterstring** value
 822 "computation" is 'c'. This is different from the question of whether the element datatype **character(repertoire)**
 823 is ordered: is 'a' < 'c'? It is also different from the question of whether the value space of datatype **characterstring**
 824 is ordered by some collating-sequence, e.g. is "computation" < "Computer"?

825 **6.8.6 Access method**

826 The access method for an aggregate datatype is the property which determines how component values
 827 can be extracted from a given aggregate-value.

828 An aggregate datatype has a direct access method, if and only if there is an aggregate-imposed mapping
 829 between values of one or more "index" (or "key") datatypes and the component values of each aggregate
 830 value. Such a mapping is required to be single-valued, i.e. there is at most one element of each
 831 aggregate value which corresponds to each (composite) value of the index datatype(s). The dimension of
 832 an aggregate datatype is the number of index or key datatypes the aggregate has.

833 An aggregate datatype is said to be indexed, if and only if it has a direct access method, every index
 834 datatype is ordered, and an element of the aggregate value is actually present and defined for every
 835 (composite) value in the value space of the index datatype(s). Every indexed aggregate datatype has a
 836 fixed size, because of the 1-to-1 mapping from the index value space. In addition, an indexed datatype
 837 has a "partial ordering" in each dimension imposed by the order relationship on the index datatype for that
 838 dimension; in particular, an aggregate datatype with a single ordered index datatype implicitly has the
 839 ordering imposed by sequential indexing.

840 An aggregate datatype is said to be keyed, if and only if it has a direct access method, but either the index
 841 datatypes or the mapping do not meet the requirements for indexed. That is, the **index** (or **key**) datatypes
 842 need not be ordered, and a value of the aggregate datatype need not have elements corresponding to all
 843 of the key values.

844 An aggregate datatype is said to have only indirect access methods if there is no aggregate-imposed
 845 index mapping. Indirect access may be by position (if the aggregate datatype has ordering), by value of

846 the element (if the aggregate datatype has uniqueness), or by some implementation-dependent selection
847 mechanism, modeled as random selection.

848 NOTE 1 The access methods become characterizing operations on the aggregate types. It is preferable to define
849 the types by their intrinsic properties and to see these access properties be derivable characterizing operations.

850 NOTE 2 The sequence datatype generator (see 8.4.4) is said to have indirect access because the only way a
851 given element value (or an element value satisfying some given condition) can be found is to traverse the list in order
852 until the desired element is the "Head". In general, therefore, one cannot access the desired element without first
853 accessing all (undesired) elements appearing earlier in the sequence. On the other hand, Array (see 8.4.5) has direct
854 access because the access operation for a given element is "find the element whose index is i" – the ith element can
855 be accessed without accessing any other element in the given Array. Of course, if the Array element which satisfies a
856 condition not related to the index value is wanted, access would be indirect.

857 6.8.7 Recursive structure

858 A datatype is said to be recursive if a value of the datatype can contain (or refer to) another value of the
859 datatype. In this International Standard, a recursive capability is supported by the type-declaration facility
860 (see 9.1), and recursive datatypes can be described using type-declaration in combination with choice
861 datatypes (8.3.1) or pointer datatypes (8.3.2). Thus recursive structure is not considered to be a property
862 of aggregate datatypes per se.

863 EXAMPLE LISP has several "atomic" datatypes, collected under the generic datatype "atom", and a "list" datatype
864 which is a sequence of elements each of which can be an atom or a list. This datatype can be described using the
865 Tree datatype generator defined in 10.2.2.

866 6.8.8 Structured and unstructured

867 Aggregate datatypes are:

- 868 — conceptually structured, having both the component datatypes and the access method specified, or
- 869 — conceptually semi-structured, having either the component datatypes or the access method specified,
870 but not both, or
- 871 — conceptually unstructured, having neither the component datatype nor the access method specified.

872 6.8.9 Mandatory and optional components

873 The components of an aggregate datatype may not all be required to have a valid value of the datatype,
874 i.e., the actual value space of the datatype may include values for which some of the component values
875 are unspecified.

876 When a component of the datatype is required to have a valid value in order for the aggregate value to be
877 a valid value of the datatype, the component is said to be a mandatory component.

878 When a component of the datatype is not required to have a valid value in order for the aggregate value to
879 be a valid value of the datatype, the component is said to be an optional component.

880 NOTE 1 This property applies to fields of records, members of classes, and elements of sequences, tables, and
881 arrays.

882 NOTE 2 See examples in 6.9.

6.9 Provisions associated with datatypes

A *provision* is the fundamental unit of normative wording⁴ in a normative document, such as a standard or specification. A provision is an "expression of normative wording that takes the form of a statement, an instruction, a recommendation or a requirement". Auxiliary verbs such as "shall" (mandatory requirement), "should" (recommendation), and "may" (optional requirement) are used in normative wording to express provisions.

This International Standard contains many provisions. Some provisions apply to datatypes in general, e.g., a datatype consists of a value space, properties, and characterizing operations — a "statement" provision. Some provisions apply to specific datatypes, e.g., a mapping to the GPD `integer` datatype shall be a datatype that is numeric — a "requirement" provision. Declarations may contain provisions described via annotations (outside the scope of this International Standard). Declarations may contain provisions associated with datatype families, as described by the `provision()` *type-attribute*.

A *normative datatype* is a collection of specifications for datatype properties that may be simultaneously satisfied by more than one actual datatype. A related concept concerns *conformity* to a normative datatype: a datatype conforms to a normative datatype if it satisfies all of the properties specified by the normative datatype, i.e., a normative datatype does not have a specific value space, but it may specify properties that any conforming value space must have. Similarly, a normative datatype may specify operations that must be supported by a conforming datatype, without that set of operations itself being sufficient to characterize any one datatype.

EXAMPLE 1 The normative datatype **Any** can be satisfied by any GPD datatype, with any value space. The only requirement is that **Equal** is defined on the value space.

EXAMPLE 2 The normative datatype `address_label_standard` is a record that contains 6 components.

```
// shorthand for "mandatory data element" provision
normative MDE = provision(obligation=require, target=type, scope=identifier, subset=defined),

// shorthand for "optional data element" provision
normative ODE = provision(obligation=permit, target=type, scope=identifier, subset=defined),

// shorthand for "extended data element" provision
normative XDE = provision(obligation=permit, target=type, scope=identifier,
subset=undefined),

normative address_label_standard =
record XDE
(
    name MDE: characterstring,
    address MDE: characterstring,
    city MDE: characterstring,
    state_province MDE: characterstring,
    postal_code MDE: characterstring,
    country_code ODE: characterstring,
),
```

It is not possible to instantiate a normative datatype directly, but it is possible to instantiate an implementation (of the normative datatype) that conforms to the normative datatype. The following are examples of datatypes (implementations) that conform to the normative datatype `address_label_standard`. It is possible to instantiate the following datatypes.

```
// address_label_1 conforms because it has all the mandatory data elements
type address_label_1 =
record
(
    name: characterstring, // mandatory data element
    address: characterstring, // mandatory data element
    city: characterstring, // mandatory data element
    state_province: characterstring, // mandatory data element
```

⁴ Provisions, in general, may be expressed in natural language text and/or specialized notation. For the GPD "provision()", the provision is expressed as a set of name-value pairs.

```

938         postal_code: characterstring, // mandatory data element
939     ),
940
941     // address_label_2 conforms because it has all the mandatory data elements,
942     // and the optional data element (present in address label 2) conforms to
943     // the requirements in the normative datatype
944     type address_label_2 =
945     record
946     (
947         name: characterstring,
948         address: characterstring,
949         city: characterstring,
950         state_province: characterstring,
951         postal_code: characterstring,
952         country_code: characterstring, // optional data element
953     ),
954
955     // address_label_3 conforms because it has the data elements
956     // of address_label_2 and the XDE permits the definition of
957     // additional data elements
958     type address_label_3 =
959     record
960     (
961         name: characterstring,
962         address: characterstring,
963         city: characterstring,
964         state_province: characterstring,
965         postal_code: characterstring,
966         country_code: characterstring,
967         telephone_number: characterstring, // extended data element
968     ),
969
970

```

The following are examples of datatypes that do not conform to the datatype.

```

971     // address_label_4 does not conform because it is missing
972     // mandatory data elements "state_province" and "postal_code"
973     type address_label_4 =
974     record
975     (
976         name: characterstring,
977         address: characterstring,
978         city: characterstring,
979     ),
980
981     // address_label_5 does not conform because its optional data element
982     // conflicts with the definition of the normative datatype
983     type address_label_5 =
984     record
985     (
986         name: characterstring,
987         address: characterstring,
988         city: characterstring,
989         state_province: characterstring,
990         postal_code: characterstring,
991         country_code: integer,
992     ),
993

```

7 Elements of the Datatype Specification Language

This International Standard defines a datatype specification language, in order to formalize the identification and declaration of datatypes conforming to this International Standard. The language is a subset of the Interface Definition Notation defined in ISO/IEC 13886:1996, *Information technology — Language-Independent Procedure Calling (LIPC)*.⁵ This clause defines the basic syntactic objects used in that language.

⁵ The IDN is only one feature of ISO/IEC 13886. The primary purpose of ISO/IEC 13886 is to specify a technique for language-independent procedure calling.

7.1 IDN character-set

The following productions define the character-set of the datatype specification language.

```

letter      = "a" | "b" | "c" | "d" | "e" |
              "f" | "g" | "h" | "i" | "j" |
              "k" | "l" | "m" | "n" | "o" |
              "p" | "q" | "r" | "s" | "t" |
              "u" | "v" | "w" | "x" | "y" |
              "z" | "A" | "B" | "C" | "D" |
              "E" | "F" | "G" | "H" | "I" |
              "J" | "K" | "L" | "M" | "N" |
              "O" | "P" | "Q" | "R" | "S" |
              "T" | "U" | "V" | "W" | "X" |
              "Y" | "Z" ;
digit       = "0" | "1" | "2" | "3" | "4" |
              "5" | "6" | "7" | "8" | "9" ;
special     = "(" | (* left parenthesis *)
              ")" | (* right parenthesis *)
              "." | (* full stop *)
              "," | (* comma *)
              ":" | (* colon *)
              ";" | (* semicolon *)
              "=" | (* equals sign *)
              "/" | (* solidus *)
              "*" | (* asterisk *)
              "-" | (* hyphen-minus *)
              "{" | (* left curly bracket *)
              "}" | (* right curly bracket *)
              "[" | (* left square bracket *)
              "]" | (* right square bracket *)
underscore = "_" ; (* low line *)
apostrophe  = "'" ; (* apostrophe *)
quote       = '"' ; (* quotation mark *)
escape      = "!" ; (* exclamation point *)
space       = " " ; (* space *)
non-quote-character = letter |
                      digit |
                      special |
                      underscore |
                      apostrophe |
                      space ;
bound-character = non-quote-character |
                  quote ;
added-character = ? not defined by this International Standard ? ;

```

These productions are nominal. Lexical productions are always subject to minor changes from implementation to implementation, in order to handle the vagaries of available character-sets. The following rules, however, always apply:

1. The bound-characters, and the escape character, are required in any implementation to be associated with particular members of the implementation character set.
2. The character space is required to be bound to the "space" member of ISO/IEC 10646, but it only has meaning within character-literals and string-literals.
3. A bound-character is required to be associated with the member having the corresponding symbol, if any, in any implementation character-set derived from ISO/IEC 10646.
4. An added-character is any other member of the implementation character-set which is bound to the member having the corresponding symbol in an ISO/IEC 10646 character-set. An added-character may be referenced by name, by 8-digit short UCS identifier, or by 4-digit short UCS identifier, as specified by ISO/IEC 10646. For example, "!"**QUOTATION MARK**!", "!"**U00000022**!", and "!"**U+0022**!" are all equivalent: a string literal that contains the one character, a quotation mark.

7.2 Whitespace

A sequence of one or more space characters, horizontal tabs, end of line characters, or newline characters except within a character-literal or string-literal (see 7.3), shall be considered whitespace. Any use of this International Standard may define any other characters or sequences of characters not in the above character set to be whitespace as well, such as vertical tabulators, end of page indicators, etc..

A comment is either of:

— Any sequence of characters beginning with the sequence `/*` (solidus, asterisk) and terminating with the first occurrence thereafter of the sequence `*/` (asterisk solidus).

— Any sequence of characters beginning with the sequence `//` (solidus, solidus) and terminating with the occurrence thereafter of end-of-line character sequence.

Every character of a comment shall be considered whitespace.

With respect to interpretation of a syntactic object under this International Standard, any annotation (see 7.4) is considered whitespace.

Any two lexical objects which occur consecutively may be separated by whitespace, without effect on the interpretation of the syntactic construction. Whitespace shall not appear within lexical objects.

Any two consecutive keywords or identifiers, or a keyword preceded or followed by an identifier, shall be separated by whitespace.

7.3 Lexical objects

The lexical objects are all terminal symbols except those defined in 7.1, and the objects identifier, digit-string, character-literal, string-literal.

7.3.1 Identifiers

An *identifier* is a terminal symbol used to name a datatype or datatype generator, a component of a generated datatype, or a value of some datatype.

```

identifier           = initial-letter-like, { pseudo-letter-like } ;
initial-letter-like = letter-like |
                       special-like ;
letter-like         = letter |
                       ISO/IEC-10176-extended-letter ;
pseudo-letter-like = letter-like |
                       digit-like |
                       underscore ;
digit-like          = digit |
                       ISO/IEC-10176-extended-digit ;
special-like        = underscore |
                       ISO/IEC-10176-extended-special ;

```

NOTE ISO/IEC 10176 describes the notion of classes of letter-like characters (outside of ISO/IEC 646 letters A through Z), digit-like characters (outside of ISO/IEC 646 digits 0 through 9), and special characters that are all used in identifiers.

Multiple identifiers with the same spelling are permitted, as long as the object to which the identifier refers can be determined by the following rules:

1. An identifier *X* declared by a *type-declaration* or *value-declaration* shall not be declared in any other declaration.