INTERNATIONAL STANDARD

ISO/IEC 8613-10

Second edition
1995-10-15

Information technology — Open Document Architecture (ODA) and Interchange Format —

Part 10: Formal specifications

Technologies de l'information — Architecture de document ouverte (ODA) et format de transfert —

Partie 10: Spécifications formelles

Reference number
ISO/IEC 8613-10:1995(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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 8613-10 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 18, *Document processing and related communication*.

This second edition is a revision of the first edition (ISO/IEC 8613-10:1991). It also incorporates amendment 1:1991, amendment 2:1991, amendment 3:1992, amendment 4:1992 and amendment 5:1993.

ISO/IEC 8613 consists of the following parts, under the general title *Information technology — Open Document Architecture (ODA) and Interchange Format*:

- *Part 1: Introduction and general principles*
- *Part 2: Document structures*
- *Part 3: Abstract interface for the manipulation of ODA documents*
- *Part 4: Document profile*
- *Part 5: Open Document Interchange Format*
- *Part 6: Character content architectures*
- *Part 7: Raster graphics content architectures*
- *Part 8: Geometric graphics content architectures*
- *Part 9: Audio content architectures*
- *Part 10: Formal specifications*
- *Part 11: Tabular structures and tabular layout*
- *Part 12: Identification of document fragments*
- *Part 13: Spreadsheet*
- *Part 14: Temporal relationships and non-linear structures*

Annex A of this part of ISO/IEC 8613 is for information only.

Information technology — Open Document Architecture (ODA) and Interchange Format —

Part 10:
Formal specifications

1 Scope

The purpose of ISO/IEC 8613 is to facilitate the interchange of documents.

In the context of ISO/IEC 8613, documents are considered to be items such as memoranda, letters, invoices, forms and reports, which may include pictures and tabular material. The content elements used within the documents may include graphic characters, geometric graphics elements and raster graphics elements, all potentially within one document.

NOTE — ISO/IEC 8613 is designed to allow for extensions, including hypermedia features, spreadsheets and additional types of content such as audio and video.

In addition to the content types defined in this International Standard, ODA also provides for arbitrary content types to be included in documents.

ISO/IEC 8613 applies to the interchange of documents by means of data communications or the exchange of storage media.

It provides for the interchange of documents for either or both of the following purposes:

- to allow presentation as intended by the originator;
- to allow processing such as editing and reformatting.

The composition of a document in interchange can take several forms:

- formatted form, allowing presentation of the document;
- processable form, allowing processing of the document;
- formatted processable form, allowing both presentation and processing.

ISO/IEC 8613 also provides for the interchange of ODA information structures used for the processing of interchanged documents.

This part of ISO/IEC 8613

- specifies a formal description technique appropriate for describing the technical specifications of the document structures (ISO/IEC 8613-2), the document profile (ISO/IEC 8613-4) and the content architectures (currently ISO/IEC 8613-6, -7 and -8);
- gives formal specifications of the document structures, the document profile and the content architectures using this formal description technique.

The aim of the formal specifications of ODA (FODA) is to provide a precise and unambiguous interpretation of the technical specifications in other parts of ISO/IEC 8613 (currently parts 2, 4, 6, 7, and 8), using formal syntax and formal semantics.

FODA can be used

— as a basis for implementations of ISO/IEC 8613;

— as a validation tool for the verification of conforming systems;

— as a reference point for examining future extensions and revisions to ISO/IEC 8613.

If a discrepancy between the natural language text and the formal specifications should be discovered, the natural language text should be regarded as the valid interpretation of this International Standard until the discrepancy is resolved.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 8613. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 8613 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 2022:1994, *Information technology — Character code structure and extension techniques.*

ISO/IEC 6429:1992, *Information technology — Control functions for coded character sets.*

ISO/IEC 6937:1994, *Information technology — Coded graphic character set for text communication — Latin alphabet.*

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

ISO/IEC 8613-1:1994, *Information technology — Open Document Architecture (ODA) and Interchange format: Introduction and general principles.*

ISO/IEC 8613-2:1995, *Information technology — Open Document Architecture (ODA) and Interchange format: Document structures.*

ISO/IEC 8613-4:1994, *Information technology — Open Document Architecture (ODA) and Interchange format: Document profile.*

ISO/IEC 8613-6:1994, *Information technology — Open Document Architecture (ODA) and Interchange format: Character content architectures.*

ISO/IEC 8613-7:1994, *Information technology — Open Document Architecture (ODA) and Interchange format: Raster graphics content architectures.*

ISO/IEC 8613-8:1994, *Information technology — Open Document Architecture (ODA) and Interchange format: Geometric graphics content architectures.*

ISO/IEC 8632-1:1992, *Information technology — Computer graphics — Metafile for the storage and transfer of picture description information — Part 1: Functional specification.*

ISO/IEC 8632-3:1992, *Information technology — Computer graphics — Metafile for the storage and transfer of picture description information — Part 3: Binary encoding.*

ISO/IEC 9541-2:1991, *Information technology — Font information interchange — Part 2: Interchange format.*

3 Definitions

For the purposes of this part of ISO/IEC 8613, the definitions given in ISO/IEC 8613-1 apply.

4 Syntax and semantics of the specification language

This clause describes the formal description technique used for the formal specifications.

NOTE — A tutorial on this formal description technique is given in annex A.

4.1 Basic concepts

ISO/IEC 8613 describes document structures, the document profile and the content architectures in terms of abstract information constructs which are drawn from the following structural categories:

— An ODA construct may be an **atomic** construct, e.g. an attribute name or a natural number within an object identifier.

— An ODA construct may be a **composite** construct, i.e. may consist of other constructs. With respect to their interrelationship, three kinds of composition are distinguished. An ODA construct may be

a) a **set**;
b) a **function** (mapping);
c) a **sequence** (list, string);

of other ODA constructs.

For example, a specific layout description is a set (of constituents), a constituent is a nomination (see below) which is a function or mapping (from attribute names onto attribute values), and an attribute value of 'subordinates' or of 'object identifier' is a sequence (of atomic natural numbers).

It is these very structures which are captured by the language used for the formal specifications of ISO/IEC 8613. The language used is called IMCL, *Information Modelling by Composition Language*. The semantics of the specification language consist of the following abstract elements:

— the universe which is a non-empty set of **entities** of the following kinds:

a) constructs;
b) spots;
c) spotsets (i.e. sets of spots);
d) the entity UNDEF ("undefined");

— functions from the universe to the universe, that is, operators on entities of the universe;

— predicates in the universe, that is, predicates on entities of the universe.

A **construct** is an information object which is one of the following:

— an atomic construct or **atom**, for short;
— a composite construct or **compound**, for short, which may be
a) a **collection**, which is an unordered set of component constructs;
b) a **nomination**, which is a function that can be regarded as an unordered set of ordered pairs where each pair consists of a *name* and a *value*;
c) a **catenation**, which is a sequence of component constructs.

The special terminology for composite constructs is to distinguish them from other sets, functions or sequences.

In order to be able to address components in constructs of arbitrary compositional structure, the concept of a **spot** is introduced. This concept is an abstract counterpart for the intuitive idea associated with pointing into an information structure at some position and saying "here". However, in general the "here" is not identified uniquely by the component construct as such (e.g. in a word, the same letter may occur several times), but rather by the context in which it appears. To deal conceptually with the idea of "here" requires a way to identify contexts.

The concept of a spot allows the distinction to be made between a considered construct and its position within a comprising composite construct of which it is a component. For example, the character string "data" (a catenation) has the component constructs 'd', 'a' and 't'. Whereas 'd' and 't' appear at one spot each, the 'a' appears at two spots, namely at the second and at the fourth position counted from the front end. So, "data" has four

component spots, but only three component constructs. If a construct is considered outside any context, it is said to be at its **ownspot**.

Spots are usually selected by selection criteria. However, a selection criterion need not be unique. Thus, the objects most naturally dealt with are not even spots, but rather sets of spots or **spotsets**. Consequently, there are no expressions for single spots, but for singleton spotsets, instead, i.e. for spotsets with only one spot (see ∈ in 4.3 and ˆ in 4.4).

It should be noted that the specification language is built on first-order predicate logic and mathematical set theory.

4.2 Syntax of the specification language

This subclause defines the syntax of the specification language, i.e., each expression in the formal specifications is built using the syntax rules given in this clause. The semantics of the terminal symbols appearing in the syntax rules are specified in 4.3 to 4.5.

Remarks on the meta-language:

The symbol pairs { }, [] and –. .– as well as the symbols ::= , | , ... , and ≡ belong to the meta-language. They have the following meanings:

::= separates the meta-variable to be defined (left-hand side) from the meta-language expression which defines it (right-hand side)

{ } delimit a syntactical unit

[ ] delimit a syntactical unit and indicate that this syntactical unit is optional, i.e. may also be absent
NOTE 1: The meta-language symbols [and] are different from the special-characters [and] used in the production rules for empty-constant, explicit-composition-term and extensional-collection-term.

–. .– delimit a comment in the meta-language text

| separates alternative syntactical units, i.e. indicates a choice of exactly one of the syntactical units
e.g. $\{a \mid m \mid p\}x$ means
ax or mx or px
NOTE 2: The meta-language symbol | is different from the special-character | used in terms denoting sets.

... is a convenient notation for recursive definitions: the symbol follows a syntactical unit which may appear one or more times, i.e. which may be repeated several times
e.g. $\{yf\}$... means
yf or $yfyf$ or $yfyfyf$ etc.
e.g. $i[so]$... means
i or iso or $isoso$ etc.

≡ A space in the language definition requires one or more blanks in the defined language expression. Just the reverse is indicated by the symbol ≡, which requires an immediate juxtaposition of the neighboring strings of the specification language. Where syntactical uniqueness is not affected, blanks may be omitted in expressions of the specification language (e.g. before and after parentheses).

For the sake of readability, the symbols "(" and ")" are used as meta-variables (instead of word-symbols such as "left-del" or "right-del"). All other meta-variables are strings of lower-case letters (with the hyphen for linking the components of a meta-variable).

Production rules:

expression ::=
formula | term

formula ::=
prime-formula |
not formula | formula { *and* | *or* | *impl* | *iff* | *xor* } formula |
∃ var (formula) | ∀ var (formula) |
(formula)

NOTE 3: The terminal symbols used in this production rule have the usual semantics of first-order predicate logic: *not* is the logical negation, *and*, *or*, *xor* (exclusive or), *impl* (implies) and *iff* (if and only if) are the usual logical connectors, ∀ (for all) and ∃ (exists) are the logical quantifiers.

prime-formula ::=
[parameter-part] predicate-symbol-part ...
[parameter-part predicate-symbol-part ...] ...[parameter-part]

predicate-symbol-part ::=
upper-case-letter [≡ letter |≡ digit]... ≡ lower-case-letter [≡ letter |≡ digit]... |
= |≠|<|≤|>|≥|∈|∉|∈̂|⊂|⊆|⊃|⊇

NOTE 4: The semantics of the terminal symbols (=, ≠, ... ⊇) in this production rule are specified in 4.3.

term ::=
var |
constant |
operator-term |
explicit-composition-term |
conditional-term |
extensional-collection-term |
extensional-spotset-term |
spot-selection-term |
(term)

var ::=
lower-case-letter [≡ letter |≡ digit]... [≡ subscript-digit]...

constant ::=
standard-constant |
nonstandard-constant

standard-constant ::=
UNDEF |
empty-constant |
number-atom-constant

empty-constant ::=
[ ] –. empty collection .– |
[:] –. empty nomination .– |
[→] –. empty catenation .– |
< > –. empty spotset .–

number-atom-constant ::=
[+ ≡| − ≡] digit [≡ digit]... [≡ . ≡ digit [≡ digit]...]

nonstandard-constant ::=
' ≡ character [≡ character]... ≡ ' –. restriction on apostrophe occurrence .–

operator-term ::=
[parameter-part] operator-symbol-part ...
[parameter-part operator-symbol-part ...]...[parameter-part]

operator-symbol-part ::=
upper-case-letter [≡ upper-case-letter |≡ digit |≡ _]... |
^ | + | − | * | / | ∪ | ∩ | \ | // | • | • |↓|↑

NOTE 5: The semantics of the terminal symbols (^, +, ... ↑) in this production rule are specified in 4.4.

explicit-composition-term ::=
[term [; term]...] –. collection .– |
[term : term [; term : term]...] –. nomination .– |
[→ term [→ term]... →] –. catenation .– |
" ≡ character [≡ character]... ≡ " –. catenation of characters, restriction on quote occurrence .–

conditional-term : :=
IF formula THEN term ELSE term

NOTE 6: The semantics of the terminal symbols (IF , THEN and ELSE) in this production rule are specified in 4.5.

extensional-collection-term : :=
[var | formula] —. collection of constructs for which the formula holds.—

extensional-spotset-term : :=
< var | formula > —. union of singleton spotsets for which the formula holds.—

spot-selection-term : :=
term spot-selection-clause |
elliptic-spot-selection-term

spot-selection-clause : :=
<var || formula> |
<formula> —. *xs* is assumed for var .— |
<name-specification [, name-specification] ... >

elliptic-spot-selection-term : :=
term {• | • |↓|↑} name-specification

NOTE 7: The semantics of the terminal symbols (•, •, ↓ and ↑) in this production rule are specified in 4.5.

name-specification : :=
nonstandard-constant | var

parameter-part : :=
term | (term [, term]...)

character : :=
letter | digit | subscript-digit | special-character

letter : :=
upper-case-letter | lower-case-letter

upper-case-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

lower-case-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

digit : :=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

subscript-digit : :=
$_0$ |$_1$|$_2$|$_3$|$_4$|$_5$|$_6$|$_7$|$_8$|$_9$

special-character : :=
. | , | ; | + | - | —. etc. .—

4.3 Predicate symbols with built-in semantics

A sequence of predicate-symbol-parts is referred to as a **predicate symbol**. For each n-ary predicate symbol there is an n-ary predicate on the universe of the specification language, i.e. an n-ary relation on entities of the universe. Some predicate symbols have built-in semantics which are introduced by the following.

NOTE — The predicate-symbol-parts are syntactically distinguished from operator-symbol-parts and variables.

True	means	the valid fact (something stated as being true)
False	means	the invalid fact (something stated as being false)
IsAtom(t)	means	t is an atomic construct or atom, for short
IsNat(t)	means	t is a natural number (1, 2, ...; zero excluded)
IsInt(t)	means	t is an integer number (... -2, -1, 0, 1, 2, ...)
IsReal(t)	means	t is a real number
IsCol(t)	means	t is a collection
IsNom(t)	means	t is a nomination
IsCat(t)	means	t is a catenation
IsSpotset(t)	means	t is a spotset
IsSingle(t)	means	t is a singleton spotset
$t_1 = t_2$	means	t_1 is equal to t_2 (all entities)
$t_1 \neq t_2$	means	*not* $t_1 = t_2$
$t_1 < t_2$	means	t_1 is less than t_2 (numbers)
$t_1 \leq t_2$	means	t_1 is less than or equal to t_2
$t_1 > t_2$	means	t_1 is greater than t_2
$t_1 \geq t_2$	means	t_1 is greater than or equal to t_2
$t_1 \in t_2$	means	t_1 is element of t_2 (collections)
$t_1 \notin t_2$	means	*not* $t_1 \in t_2$
$t_1 \hat{\in} t_2$	means	t_1 is singleton spotset and subset of t_2 (spotsets)
$t_1 \subset t_2$	means	t_1 is subset of t_2 (collections or spotsets)
$t_1 \subseteq t_2$	means	t_1 is subset of or equal to t_2 (collections or spotsets)
$t_1 \supset t_2$	means	t_2 is subset of t_1 (collections or spotsets)
$t_1 \supseteq t_2$	means	t_2 is subset of or equal to t_1 (collections or spotsets)

The unary predicate symbols mean predicates for expressing that an entity belongs to a certain class or "type" of entities, i.e. has a particular property. The binary predicate symbols refer to predicates which indicate whether or not a particular relationship holds for two entities.

4.4 Operator symbols with built-in semantics

A sequence of operator-symbol-parts is referred to as an **operator symbol**. For each n-ary operator symbol there is an n-ary operator or function from the universe to the universe of the specification language, i.e., a mapping from n-ary tuples of entities onto entities of the universe. Some operator symbols have built-in semantics which are introduced by the following.

NOTE — The operator-symbol-parts are syntactically distinguished from predicate-symbol-parts and variables. For all operators it holds that the result is UNDEF, if a parameter term does not meet the requirement stated below.

C t	If t denotes a singleton spotset, C t denotes the component construct at the spot given by t.
N t	If t denotes a singleton spotset of a spot that is a component of a nomination ("immediately inward" of a nomination is the formal term), then N t denotes the name construct of the component as it is within the nomination.
F t	If t denotes a set of exactly one spot immediately inward of a catenation spot, F t denotes the front part of this catenation up to but excluding the component given by t (catenation of components with lower position than t).

CARD t	If t denotes a collection or a spotset, CARD t denotes its cardinality, i.e., the number of component constructs or the number of spots.
R t	If t denotes a set of exactly one spot immediately inward of a catenation spot, R t denotes the rear part of this catenation up to but excluding the component given by t (catenation of components with higher position than t).
$t_1 + t_2$ $t_1 - t_2$ $t_1 * t_2$ t_1/t_2	If t_1 and t_2 denote numbers, the terms denote numbers as known from arithmetic. The usual precedence rules for arithmetical operators apply.
$t_1 \cup t_2$ $t_1 \cap t_2$ $t_1 \setminus t_2$	If t_1 and t_2 denote either collections or spotsets, the terms denote their set-theoretic union, intersection or difference, i.e. either collections or spotsets.
$t_1 // t_2$	If t_1 and t_2 denote catenations, $t_1 // t_2$ denotes the catenation obtained by concatenating the two catenations in the given order.
$\hat{}t$	If t denotes a construct, then $\hat{}t$ ("ownspot of t") denotes the singleton spotset containing the ownspot of t.
$t\bullet$	If t denotes a spotset containing no atom spots (spots with atoms), then $t\bullet$ ("next inwards") denotes the set of all spots which are immediately inward of the spots of the spotset t.
$t\downarrow$	If t denotes a spotset, $t\downarrow$ denotes the set of all terminal spots inward of or - for atoms and empty constructs - equal to the spots given by t (read "most inward").
$t\bullet$	If t denotes a spotset without ownspots, $t\bullet$ denotes the set of all spots which are immediately outward of the spots given by t (read "next outward").
$t\uparrow$	If t denotes a spotset, $t\uparrow$ denotes the set of all ownspots outward of or - for ownspots - equal to the spots given by t (read "most outward").
$t\bullet$ $t\downarrow$ $t\bullet$ $t\uparrow$	If t denotes the empty spotset, the operator-terms denote the empty spotset, too.

The normal evaluation order for expressions is from left to right, with the following exceptions:

— If a term is enclosed in opening and closing parentheses this term is evaluated first;
— Operators have precedence over predicates;
— Between operators the precedence order is:

1.: $\hat{}$ (ownspot of)
2.: $\bullet$ $\bullet$ $\downarrow$ $\uparrow$ (next inward, next outward, most inward, most outward)
3.: spot-selection-clause
4.: spot-selection-term
5.: other operators

4.5 Other terms

Apart from operator-terms there are other compound terms which result in constructs or spotsets (or UNDEF). Their built-in semantics are introduced by the following.

$[t_1; t_2; t_3]$	If t_i denote constructs, the whole term denotes the collection which contains the constructs t_i as components. (This is an example for explicit-composition-term)
[ ]	Denotes the empty collection. (This is an example for empty-constant)
$[n_1 : c_1; n_2 : c_2]$	If n_i and c_i denote constructs, where all n_i are distinct, the whole term denotes the nomination which contains the constructs c_i as components under the (unique) names n_i. (This is an example for explicit-composition-term)
[:]	Denotes the empty nomination. (This is an example for empty-constant)

$[\rightarrow m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow]$

If m_i denote constructs, the whole term denotes the catenation which contains the constructs m_i as components — also referred to as members — in the indicated sequence. (This is an example for explicit-composition-term)

"ODA Part 2"

Denotes the catenation [→ 'O' → 'D' → 'A' → ' ' → 'P' → 'a' → 'r' → 't' → ' ' → '2' →]. A string of characters enclosed in quotes denotes the catenation of those characters. A pair of quotes in the string stands for a single one in the catenation. (This is an example for explicit-composition-term)

[→]

Denotes the empty catenation. (This is an example for empty-constant)

< >

Denotes the empty spotset. (This is an example for empty-constant)

IF formula THEN t_1 ELSE t_2

If t_1 and t_2 are terms, the whole term denotes the same as t_1 or t_2, depending on whether the formula is True or False, respectively. (This is an example for conditional-term)

[*var* | formula]

Denotes the collection of all constructs *var* which satisfy the formula. (This is an example for extensional-collection-term)

< *var* | formula >

Denotes the spotset which is the union of all singleton spotsets *var* which satisfy the formula. (This is an example for extensional-spotset-term)

t <*var* || formula >

If *t* denotes a (possibly empty) spotset, the whole term denotes the union of all singleton spotsets *var* which contain a spot taken from *t* and for which the formula is True. (This is an example for spot-selection-term)

Three elliptic notations are provided for frequently occurring spot-selection-clauses:

t < formula >

If a variable *var* is not introduced explicitly, the abbreviated term *t* < formula > is evaluated for the standard variable *xs* (singleton set of the spot under examination or examination spot, for short). (This is an example for spot-selection-term)

t <$n_1, n_2, \ldots$>

If the formula has the structure
N *var* = n_1 *or* N *var* = n_2 *or* ...
where n_i are name-specifications, the formula may be abbreviated as a list of name-specifications. (This is an example for spot-selection-term)

$t_{\bullet} n \quad t \downarrow n \quad t^{\bullet} n \quad t \uparrow n$

If there is only one name-specification *n* used for spot selection, an elliptic-spot-selection-term is provided as an abbreviation of special spot-selection-terms (ending with $_\bullet$, $^\bullet$ etc.). The *n* stands for <N $xs = n$> (see name qualification in programming languages). (This is an example for elliptic-spot-selection-term)

4.6 Notational simplifications

The common notational simplifications for successive logical quantifications can be used. The following examples explain these "short-hand" notations which are usually applied in first-order predicate logic:

The expression $\forall x(\forall y(\exists z(\text{formula})))$
may be written as $\forall x \forall y \exists z$ (formula)
or even as $\forall x, y \; \exists z$ (formula)

A further abbreviation is used to help emphasize the "essential part" of a quantified formula:

The expression $\forall x(x \in m$ <u>*impl*</u> formula)
may be written as $\forall x \in m$ (formula)
and $\exists x(x \in m$ <u>*and*</u> formula)
may be written as $\exists x \in m$ (formula)

This notation can be combined with the previous one:

The expression $\forall x(x \in m$ <u>*impl*</u> $\forall y(\exists z(z \in p$ <u>*and*</u> formula)))

may be written as $\forall\, x \in m,\ y\ \exists\, z \in p$ (formula)

However, note that the expression

$\forall\, y, x \in m\ \exists\, z \in m$ (formula)

includes the restriction $y \in$ m and has therefore to be expanded to

$\forall\, y(y \in m\ \underline{impl}\ \forall\, x(x \in m\ \underline{impl}\ \exists\, z \in m$ (formula)))

The same abbreviations are used for $\hat{\in}$.

5 Structure of the formal specifications

This clause outlines the general concepts for the formal specifications. Those terms which are used at several places throughout the formal specifications are contained in clause 6.

Formally speaking, the formal specifications consist of a single formula in first-order predicate logic. This formula is called the overall formula. The formula consists of other formulae which are connected by *and*:

$$\text{formula}_1 \ \underline{and} \ \text{formula}_2 \ \underline{and} \ \text{formula}_3 \ \underline{and} \ \ldots \ \text{formula}_n$$

This overall formula is distributed over clauses 7 to 11. Its outline is as follows:

... IsInterchangeSet$^{2.1}(doc)$ ...
and ... IsGenericDocumentDescription$^{2.2}(gdoc)$ ...

(see clause 7)

and ... (*cont*)DescribesContPortOf$^{2.153}(b)$

and ... IsDocumentProfile$^{4.1}(cst)$ ...

(see clause 8)

and ... IsCIEColour$^{4.96}(v)$ ...

and ... IsDefaultableCharacterContentArchitectureAttribute$^{6.1}(att)$ ...

(see clause 9)

and ... IsGraphicCharacter$^{6.113}(v)$ ...

and ... IsDefaultableRasterGraphicsContentArchitectureAttribute$^{7.1}(att)$ ...

(see clause 10)

and ... IsInterleavingFormatValue$^{7.32}(v)$ ...

... IsDefaultableGeometricGraphicsContentArchitectureAttribute$^{8.1}(att)$ ...

(see clause 11)

and ... IsRegisteredEdgeType$^{8.61}(v)$ ...

The complete part of each section of the overall formula is given in the indicated clauses. The overall formula has been split according to the individual parts of ISO/IEC 8613.

In the present context, each of these formulae is called a "definition" and identified through a unique reference number. A definition defines either a concept used in the narrative part of ISO/IEC 8613 or a concept which has a subsidiary function in the network of definitions in that it has been separated and "encapsulated" to render other definitions more readable.

The definitions are grouped into several clauses and subclauses. For example, within the formal specifications of the document structures the definitions relating to the sets of constituents are contained in 7.1, those relating to constituents in 7.2 and those relating to attributes in 7.3. In addition, concepts which are not used in the formal specification but are used in the text of ISO/IEC 8613-2 are defined in 7.5. The definitions relating to the document profile are given in clause 8, those relating to the character content architectures in clause 9, etc.

The "factorization" of definitions is only for the convenience of authors and readers; it does not in any way impair the formal rigidity of the approach.

Variables occurring in the definitions are always bound by universal ($\forall$) or existential ($\exists$) quantifiers. Therefore, once a value has been chosen for a variable it has to be retained throughout the scope of the quantifier wherever the variable appears.

All predicates apart from those pertaining to the specification language are defined using the same format. For unary predicates the format is:

$\forall$ *variable* (*predicate-symbol*(*variable*) *iff formula*)