INTERNATIONAL STANDARD

ISO/IEC 8613-10

First edition
1991-06-01

Information processing — Text and office systems — Office Document Architecture (ODA) and interchange format —

Part 10: Formal specifications

Traitement de l'information — Bureautique — Architecture des documents de bureau (ODA) et format d'échange —

Partie 10: Spécifications formelles

Reference number
ISO/IEC 8613-10 : 1991 (E)

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*.

Development of this International Standard has been in parallel with

— ECMA 101 : 1985, *Office document architecture*;

— CCITT Recommendation T.37 (1984): *Document interchange protocol for the telematic services*;

— CCITT Recommendations in the T.410 series (1988): *Open Document Architecture (ODA) and Interchange format*.

ISO 8613 consists of the following parts, under the general title *Information processing — Text and office systems — Office Document Architecture (ODA) and interchange format*:

— *Part 1: Introduction and general principles*

— *Part 2: Document structures*

— *Part 4: Document profile*

— *Part 5: Office document interchange format (ODIF)*

— *Part 6: Character content architectures*

— *Part 7: Raster graphics content architectures*

— *Part 8: Geometric graphics content architectures;*

— *Part 10: Formal specifications.*

NOTE — At present, there are no parts 3 and 9.

Further parts may be added to this International Standard.

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

Information processing — Text and office systems — Office Document Architecture (ODA) and interchange format —

Part 10: Formal specifications

1 Scope

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

In the context of ISO 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 8613 is designed to allow for extensions, including typographical features, colour, spreadsheets and additional types of content such as sound.

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

Furthermore, ISO 8613 allows for the interchange of documents containing one or more different types of content such as character text, images, graphics and sound.

This part of ISO/IEC 8613

— specifies a formal description technique appropriate for describing the technical specifications of the document structures (ISO 8613-2), the document profile (ISO 8613-4) and the content architectures (ISO 8613-6, -7 and -8);

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 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 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

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 ]... |
      = |≠|<|≤|>|≥|∈|∉|ê|⊂|⊆|⊃|⊇
```

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 .—
```

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	$\underline{not}$ $t_1 = t_2$
$t_1 < t_2$	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	$\underline{not}$ $t_1 \in t_2$
$t_1 \hat{\in} t_2$	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).

$[\rightarrow m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow]$	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* $\mid$ formula]	Denotes the collection of all constructs *var* which satisfy the formula. (This is an example for extensional-collection-term)
< *var* $\mid$ 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 $\underline{or}$ N *var* = n_2 $\underline{or}$ ... 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	$\forall x \forall y \exists z\ (\text{formula})$
or even as	$\forall x, y\ \exists z\ (\text{formula})$

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

The expression	$\forall x(x \in m\ \underline{impl}\ \text{formula})$
may be written as	$\forall x \in m\ (\text{formula})$
and	$\exists x(x \in m\ \underline{and}\ \text{formula})$
may be written as	$\exists x \in m\ (\text{formula})$

This notation can be combined with the previous one:

The expression	$\forall x(x \in m\ \underline{impl}\ \forall y(\exists z(z \in p\ \underline{and}\ \text{formula})))$

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, each of these formal specifications is a single formula in first-order predicate logic. This formula, being given near the beginning of each formal specification, is called the overall formula. The formula consists of other formulae which are connected by *and*:

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

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 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 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 the ISO 8613-2 are defined in 7.5.

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 (∀) or existential (∃) 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:

∀ *variable* (*predicate-symbol*(*variable*) *iff* *formula*)

A similar format is applied for n–ary predicates. There, the variables have been placed around the predicate symbol in a "natural" way, e.g. (*id*)IsIdContIn(*b*) is expected to be read as "*id* is an identifier of a content portion in the basic object *b*".

A "predicate" in first-order predicate logic is a propositional pattern where some places are left free for the insertion of individual entities. An example is the pattern "... is greater than ...". Once the free places have been occupied by entities, the pattern becomes a proposition for which the "truth value" can be evaluated. The proposition "5 is greater than 2" evaluates to True, whereas the propositions "3 is greater than 5" and "a Mercedes is greater than SC 18" evaluate to False. Note that a proposition may never evaluate to an undefined truth value.

A unary predicate is a pattern with only one free parameter place. Its definition is evaluated by substituting an individual entity for the parameter. This yields a proposition on the left-hand side of the *iff* and transforms the whole expression in parentheses into a proposition.

A uniform format has also been adopted for the definition of most operators. It relies on the conditional-term of the specification language:

∀ *variable* (*operator-symbol*(*variable*) = IF *formula* THEN *term* ELSE UNDEF)

An "operator" is a term pattern where some places are left free for the insertion of individual entities. An example is the pattern "... plus ...". Once the free places have been occupied by entities, the pattern becomes a term, i.e. an expression which evaluates to or denotes an individual entity. The term "5 plus 2" denotes the entity 7, whereas the term "WG 3 plus SC 18" denotes the entity which is represented by UNDEF in the formal specification language. Therefore, a term evaluates to "undefined", if the operator is not defined for the inserted entities.

It should be noted that the formal specifications cannot be applied *directly* to the Office Document Interchange Format (ISO 8613-5) and the reader should not necessarily expect to see any direct correspondences. The formal specifications are based on the textual description of the concepts of ISO 8613 and do not necessarily reflect the ODIF encoding of ODA documents. Those clauses in ISO 8613 which specify the ODIF encoding usually impose additional rules for the make-up of the interchanged data stream. These additional rules are considered outside of the scope of the formal specifications: The ODIF encoding specifies these additional rules formally and there is therefore no need for repeating these additional rules in FODA.

6 Commonly used definitions

This clause contains those definitions which are not specific to the formal specification of the document structures, the document profile or one of the content architectures only.

Semiformal Description 1.1

Predicate "is a non-empty collection"

An entity *col* is a non-empty collection (set of constructs) if it is a collection which is not empty.

Definition 1.1

```
$\forall\, \mathit{col}$
$(_0\, \mathrm{IsNeCol}(\mathit{col})\ \underline{\mathit{iff}}$
    $\mathrm{IsCol}(\mathit{col})\ \underline{\mathit{and}}\ \mathit{col} \neq [\,]_0)$
```

Semiformal Description 1.2

Predicate "is a non-empty nomination"

An entity *nom* is a non-empty nomination (mapping of names onto constructs) if it is a nomination which is not empty.

Definition 1.2

```
$\forall\, \mathit{nom}$
$(_0\, \mathrm{IsNeNom}(\mathit{nom})\ \underline{\mathit{iff}}$
    $\mathrm{IsNom}(\mathit{nom})\ \underline{\mathit{and}}\ \mathit{nom} \neq [:]_0)$
```

Semiformal Description 1.3

Predicate "is a non-empty catenation"

An entity *cat* is a non-empty catenation (sequence of constructs) if it is a catenation which is not empty.

Definition 1.3

```
$\forall\, \mathit{cat}$
$(_0\, \mathrm{IsNeCat}(\mathit{cat})\ \underline{\mathit{iff}}$
    $\mathrm{IsCat}(\mathit{cat})\ \underline{\mathit{and}}\ \mathit{cat} \neq [\rightarrow]_0)$
```

Semiformal Description 1.4

Predicate "is an empty collection"

An entity *col* is an empty collection if it contains no components.

Definition 1.4

```
$\forall\, \mathit{col}$
$(_0\, \mathrm{IsEmptyCol}(\mathit{col})\ \underline{\mathit{iff}}$
    $\mathit{col} = [\,]_0)$
```

Semiformal Description 1.9

Predicate "is a pair of positive integers"

A pair of positive integers is a catenation of two positive integers.

Definition 1.9

```
$\forall q$
$({}_0$IsPairOfPosInt$(q)$ iff
  $\exists l, r$
  $({}_1 q = [\to l \to r \to]$ and IsNat$(l)$ and IsNat$(r){}_1){}_0)$
```

Semiformal Description 1.10

Predicate "is an octet string"

An octet string is an atomic construct in the formal specification.

Definition 1.10

```
$\forall v$
$({}_0$IsOctetString$(v)$ iff
  IsAtom$(v){}_0)$
```

Semiformal Description 1.11

Function "position from the front end"

PF returns the position number of the indicated component, the count beginning with the first component and ending at and including the considered component. The operand p is required to denote a set of exactly one spot immediately inward of a catenation.

Definition 1.11

```
$\forall p$
$({}_0$PF$(p) =$
  IF IsSingle$(p)$ and IsCat(C $p^\bullet$)
  THEN LENGTH$^{1.16}$ (F $p$) + 1
  ELSE UNDEF${}_0)$
```

Semiformal Description 1.12

Function "position from the rear end"

PR returns the position number of the indicated component, the count beginning with the last component and ending at and including the considered component. The operand p is required to denote a set of exactly one spot immediately inward of a catenation.

Definition 1.12

```
$\forall p$
$({}_0$PR$(p) =$
  IF IsSingle$(p)$ and IsCat(C $p^\bullet$)
  THEN LENGTH$^{1.16}$ (R $p$) + 1
  ELSE UNDEF${}_0)$
```

Semiformal Description 1.17

Function "collection of component constructs"

COLC returns the collection of the component constructs at the spots of the (possibly empty) spotset *ss*.

Definition 1.17

```
$\forall ss$
$(_0$COLC $(ss) =$
  IF IsSpotset$(ss)$
  THEN [$g \mid \exists p \hat{\in} ss \; (g = \mathrm{C}\; p)$]
  ELSE  UNDEF$_0)$
```

Semiformal Description 1.18

Function "name set of a nomination"

NAMS returns the name set of the given nomination *n*, i.e. the collection of names of the components of the nomination.

Definition 1.18

```
$\forall n$
$(_0$NAMS$(n) =$
  IF IsNom$(n)$
  THEN [$m \mid \exists p \hat{\in} \hat{}n \bullet (m = \mathrm{N}\; p)$]
  ELSE  UNDEF$_0)$
```

Semiformal Description 1.19

Predicate "is the placeholder for any attribute value"

An entity *v* is the placeholder for the value of a defaultable attribute if it is the atom with this particular interpretation.

NOTE — The term 'placeholder for any attribute value' is not an ISO 8613 attribute value but a construct which is introduced to achieve consistency throughout the formal specifications and to distinguish between mandatory and defaultable attributes or parameters. The actual value is dependent on the respective attribute and the defaulting rules and will be determined during the processing of a document (editing process, layout process, imaging process) whenever a value for a defaultable attribute or parameter is needed.

Definition 1.19

```
$\forall v$
$(_0$IsPlaceholder$(v)$ *iff*
  $v =$ 'placeholder for any attribute value'$_0)$
```

Semiformal Description 1.20

Predicate "is an ISO 6937 minimal subrepertoire string"

An ISO 6937 minimal subrepertoire string is a catenation of ISO 6937 minimal subrepertoire characters.

Definition 1.20

```
$\forall v$
$(_0$IsISO6937MSString$(v)$ *iff*
  IsNeCat$^{1.3}(v)$ *and* $\forall m \hat{\in} \hat{}v \bullet$ (IsISO6937MSCharacter$^{1.21}$(C $m$))$_0)$
```

Semiformal Description 1.25

Predicate "is a positive real number"

A positive real number is a real number whose value is greater than zero.

Definition 1.25

1 $\forall v$
2 $(_0 \mathrm{IsPosReal}(v)$ *iff*
3 $\quad \mathrm{IsReal}(v)$ *and* $v > 0.0_0)$

and ... IsBasicLayoutObjectClassDescription$^{2.35}$(*cst*) ...
and ... IsBasicObjectClassDescription$^{2.36}$(*cst*) ...
and ... IsObjectClassDescription$^{2.37}$(*cst*) ...

and ... IsRootDescription$^{2.38}$(*cst*) ...

and ... IsLogicalRootDescription$^{2.39}$(*cst*) ...
and ... IsCompositeLogicalObjectDescription$^{2.40}$(*cst*) ...
and ... IsBasicLogicalObjectDescription$^{2.41}$(*cst*) ...
and ... IsLogicalObjectDescription$^{2.42}$(*cst*) ...

and ... IsLayoutRootDescription$^{2.43}$(*cst*) ...
and ... IsPageSetDescription$^{2.44}$(*cst*) ...
and ... IsCompositePageDescription$^{2.45}$(*cst*) ...
and ... IsBasicPageDescription$^{2.46}$(*cst*) ...
and ... IsPageDescription$^{2.47}$(*cst*) ...
and ... IsFrameDescription$^{2.48}$(*cst*) ...
and ... IsBlockDescription$^{2.49}$(*cst*) ...
and ... IsLayoutObjectDescription$^{2.50}$(*cst*) ...

and ... IsCompositeObjectDescription$^{2.51}$(*cst*) ...
and ... IsBasicObjectDescription$^{2.52}$(*cst*) ...
and ... IsCompositeLayoutObjectDescription$^{2.53}$(*cst*) ...
and ... IsBasicLayoutObjectDescription$^{2.54}$(*cst*) ...
and ... IsObjectDescription$^{2.55}$(*cst*) ...

and ... IsContentPortionDescription$^{2.56}$(*cst*) ...
and ... IsCharacterContentPortionDescription$^{2.57}$(*cst*) ...
and ... IsRasterGraphicsContentPortionDescription$^{2.58}$(*cst*) ...
and ... IsGeometricGraphicsContentPortionDescription$^{2.59}$(*cst*) ...

and ... IsLayoutStyle$^{2.60}$(*cst*) ...
and ... IsPresentationStyle$^{2.61}$(*cst*) ...

-.attributes: .-

and ... IsAttributeSet$^{2.62}$(*as*) ...
and ... IsProfileAttributeSetPart2$^{2.63}$(*as*) ...

and ... IsBindingsValueExpression$^{2.64}$(*v*) ...
and ... IsStringExpression$^{2.65}$(*v*) ...
and ... IsAtomicStringExpression$^{2.66}$(*v*) ...
and ... IsStringFunction$^{2.67}$(*v*) ...
and ... IsNumericExpression$^{2.68}$(*v*) ...
and ... IsNumericFunction$^{2.69}$(*v*) ...
and ... IsObjectOrObjectClassIdExpression$^{2.70}$(*v*) ...
and ... IsObjectOrObjectClassSelectionFunction$^{2.71}$(*v*) ...
and ... IsBindingReferenceExpression$^{2.72}$(*v*) ...
and ... IsBindingReference$^{2.73}$(*v*) ...
and ... IsBindingName$^{2.74}$(*v*) ...
and ... IsBindingSelectionFunction$^{2.75}$(*v*) ...
and ... IsCurrentInstanceFunction$^{2.76}$(*v*) ...
and ... IsObjectTypeValue$^{2.77}$(*v*) ...
and ... IsLogicalObjectId$^{2.78}$(*v*) ...
and ... IsLayoutObjectId$^{2.79}$(*v*) ...
and ... IsObjectId$^{2.80}$(*v*) ...
and ... IsLogicalObjectClassId$^{2.81}$(*v*) ...
and ... IsLayoutObjectClassId$^{2.82}$(*v*) ...
and ... IsObjectClassId$^{2.83}$(*v*) ...
and ... IsSeqOfObjectClassId$^{2.84}$(*v*) ...
and ... IsConstructionExpression$^{2.85}$(*v*) ...
and ... IsConstructionType$^{2.86}$(*v*) ...

and ... (*slog*)IsContainedIn$^{2.141}$(*slay*) ...
and ... (*id*)IsClassIdImSubIn$^{2.142}$(*gco*) ...
and ... (*id*)IsIdImSubIn$^{2.143}$(*sco*) ...
and ... (*id*)IsIdContIn$^{2.144}$(*b*) ...
and ... (*idq*)IsSeqClassIdImSubIn$^{2.145}$(*gco*) ...
and ... (*idq*)CoveredBy$^{2.146}$(*r*) ...
and ... (*id*)OccursIn$^{2.147}$(*m*) ...
and ... (*go*)IsInitialGenericIn$^{2.148}$(*gos*) ...
and ... (*so*)IsInitialSpecificIn$^{2.149}$(*sos*) ...
and ... (*lgo*)DescribesClassImSubOf$^{2.150}$(*hgo*) ...
and ... (*lgo*)DescribesClassSubOf$^{2.151}$(*hgo*)In(*g*) ...
and ... (*lso*)DescribesImSubOf$^{2.152}$(*hso*) ...
and ... (*cont*)DescribesContPortOf$^{2.153}$(*b*)

NOTE — Other predicates or operators which are used here, but are defined in clause 6, are not listed here.

In principle a given interchange set may be verified to conform to ISO 8613 by establishing

$\mathcal{AS}$ *impl* IsInterchangeSet$^{2.1}$(-.**this-term**.-)

where $\mathcal{AS}$ stands for the conjugation of all axioms and definitions and the argument of IsInterchangeSet stands for a formal expression (a term) which denotes the particular interchange set in question. If the extended overall formula yields True, the document description conforms to ISO 8613, if False, it does not. An undefined result is impossible.

Semiformal Description 2.3

Predicate "is a document description" (2.3)

An entity *doc* is a document description if it is a document profile *prof* (5) or if it is an entity which is processable, formatted processable or formatted (5–6), according to the value of the document profile attribute `'document architecture class'` (7–13), with a `'resource document'` specified in the profile if any `'resource'` is specified in the document (14–15).

Definition 2.3

```
1   $\forall\, doc$
2   $(_0\, \mathrm{IsDocumentDescription}(doc)\ \underline{iff}$
3     $\exists\, prof$
4     $(_1\, \mathrm{IsDocumentProfilePart2}^{2.20}(prof)\ \underline{and}$
5       $(_2\, doc = \texttt{[}prof\texttt{]}\ \underline{or}\ \mathrm{IsProcessable}^{2.4}(doc)\ \underline{or}$
6         $\mathrm{IsFormattedProcessable}^{2.5}(doc)\ \underline{or}\ \mathrm{IsFormatted}^{2.6}(doc)\ {}_2)\ \underline{and}$
7       $doc \neq \texttt{[}prof\texttt{]}\ \underline{impl}$
8       $(_3\, (_4\, \mathrm{C}\ \hat{}prof \bullet \texttt{'document architecture class'} = \texttt{'processable'}\ \underline{iff}$
9           $\mathrm{IsProcessable}^{2.4}(doc)_4)\ \underline{and}$
10        $(_5\, \mathrm{C}\ \hat{}prof \bullet \texttt{'document architecture class'} = \texttt{'formatted processable'}\ \underline{iff}$
11          $\mathrm{IsFormattedProcessable}^{2.5}(doc)_5)\ \underline{and}$
12        $(_6\, \mathrm{C}\ \hat{}prof \bullet \texttt{'document architecture class'} = \texttt{'formatted'}\ \underline{iff}$
13          $\mathrm{IsFormatted}^{2.6}(doc)_6)\ \underline{and}$
14        $\forall\, cst \in doc$
15        $(_7\, \texttt{'resource'} \in \mathrm{NAMS}^{1.18}(cst)\ \underline{impl}\ \texttt{'resource document'} \in \mathrm{NAMS}^{1.18}(prof)\ {}_7)_3)_1)_0)$
```