INTERNATIONAL STANDARD

ISO 24610-2

First edition
2011-10-01

Language resource management — Feature structures —

Part 2: Feature system declaration

Gestion des ressources langagières — Structures de traits —

Partie 2: Déclaration de système de structures de traits

Reference number
ISO 24610-2:2011(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees 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 24610-2 was prepared by Technical Committee ISO/TC 37, *Terminology and other language and content resources*, Subcommittee SC 4, *Language resource management*.

ISO 24610 consists of the following parts, under the general title *Language resource management — Feature structures*:

— *Part 1: Feature structure representation*

— *Part 2: Feature system declaration*

Introduction

ISO 24610 is organized in two separate main parts.

— Part 1, *Feature structure representation*, is dedicated to the description of feature structures, providing an informal and yet explicit outline of their characteristics, as well as an XML-based structured way of representing feature structures in general and typed feature structures in particular. It is designed to lay a basis for constructing an XML-based reference format for exchanging (typed) feature structures between applications.

— Part 2, *Feature system declaration*, will provide an implementation standard for XML-based typed feature structures, first by defining a set of types and their hierarchy, then by formulating type constraints on a set of features and their respective admissible feature values and finally by introducing a set of validity conditions on feature structures for particular applications, especially related to the goal of language resource management.

A feature structure is a general-purpose data structure that identifies and groups together individual features by assigning a particular value to each. Because of the generality of feature structures, they can be used to represent many different kinds of information. Interrelations among various pieces of information and their instantiation in markup provide a meta-language for representing linguistic content. Moreover, this instantiation allows a specification of a set of features and values associated with specific types and their restrictions, by means of feature system declarations, or other XML mechanisms to be discussed in this part of ISO 24610.

Some of the statements here are copied from ISO 24610-1:2006 in order to make this part standalone without referring to part 1.

Language resource management — Feature structures —

Part 2:
Feature system declaration

1 Scope

This part of ISO 24610 provides a format to represent, store or exchange feature structures in natural language applications, for both annotation and production of linguistic data. It is ultimately designed to provide a computer format to define a type hierarchy and to declare the constraints that bear on a set of feature specifications and operations on feature structures, thus offering means to check the conformance of each feature structure with regards to a reference specification. Feature structures are an essential part of many linguistic formalisms as well as an underlying mechanism for representing the information consumed or produced by and for language engineering applications.

A feature system declaration (FSD) is an auxiliary file used in conjunction with a certain type of text that makes use of fs (that is, feature structure) elements. The FSD serves four purposes.

— It provides an encoding by which types and their subtyping and inheritance relationships can be introduced and defined, thus laying the basis for constructing a feature system.

— It provides a mechanism by which the encoder can list all of the feature names and feature values and give a prose description as to what each represents.

— It provides a mechanism by which type constraints can be declared, against which typed feature structures are validated relative to a given theory stated in typed feature logic. These constraints may involve constraints on the range of a feature's value, constraints on which features are permitted within certain types of feature structures, or constraints that prevent the co-occurrence of certain feature-value pairs. The source of these constraints is normally the empirical domain being modelled.

— It provides a mechanism by which the encoder can define the intended interpretation of underspecified feature structures. This involves defining default values (whether literal or computed) for missing features.

The scheme described in this part of ISO 24610 may be used to document any feature system, but is primarily intended for use with the typed feature structure representation defined in ISO 24610-1. The feature structure representations of ISO 24610-1 specify data structures that are subject to the typing conventions and constraints specified using ISO 24610-2. The feature structure representations of ISO 24610-1 are also used within some of the elements defined in ISO 24610-2.

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 24610-1:2006, *Language resource management — Feature structures — Part 1: Feature structure representation*

ISO/IEC 19757-2, *Information technology — Document Schema Definition Language (DSDL) — Part 2: Regular-grammar-based validation — RELAX NG*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 19757-2 and the following apply.

3.1
admissibility constraint
feature admissibility constraint
specification of a set of **admissible features** (3.2) and **admissible feature values** (3.3) associated with a specific **type** (3.24)

3.2
admissible feature
appropriate feature
feature which any **feature structure** (3.14) of a given **type** (3.24) may bear a **value** (3.17) for

NOTE This term is often interpreted elsewhere to mean obligatory, i.e. feature structures of the given type must bear a value for every admissible feature. This term does not imply that the feature is obligatory here.

3.3
admissible feature value
admissible value
value restriction
range restriction
value (3.17) that the value of an **admissible feature** (3.2) must be subsumed by in **feature structures** (3.14) of a given **type** (3.24)

3.4
atomic type
user-defined **type** (3.24) with no **admissible features** (3.2) declared or inherited

3.5
bag
multiset
triple of an integer n, a set S and a function that maps the integers in the range, 1 to n, to elements of S

NOTE A bag is halfway between a set (in that its elements are unordered) and a list (in that particular elements can occur more than once).

3.6
built-in
non-user-defined element that may appear in place of a **feature structure** (3.14), for example, as a **feature value** (3.17)

NOTE Built-ins can be atomic or complex. The atomic built-ins are numeric, string, symbol and binary. The complex built-ins are **collections** (3.7) and applications of the operators, i.e. alternation, negation and merge (5.2.4).

3.7
collection
feature value (3.17) consisting of potentially many values, organized as a list, set or **bag** (3.5)

3.8
constraint
unit of specification that identifies some collection of **feature structures** (3.14) as invalid

NOTE 1 All constraints are implicational in their syntactic form, although some are distinguished as admissibility constraints. See **validity** (3.31) and 5.4. All feature structures not explicitly excluded as invalid are considered to be valid.

NOTE 2 A feature structure that has not been so identified by any of the constraints in a feature system is considered to be valid.

3.9
default value
value (3.17) otherwise assigned to a **feature** (3.12) when one is not specified

EXAMPLE Masculine is the default value of the grammatical gender in Dutch.

NOTE A feature structure may not bear a feature without a corresponding value.

3.10
empty feature structure
feature structure (3.14) that contains no information

NOTE An empty feature structure subsumes all other feature structures.

3.11
extension
converse of **subsumption** (3.21)

NOTE A feature structure F extends G if and only if G subsumes F.

3.12
feature
property or aspect of an entity that is formally represented as a function mapping the entity to a corresponding **value** (3.17)

3.13
feature specification
pairing of a **feature** (3.12) with a **value** (3.17) in a feature structure description

3.14
feature structure
record structure that associates one **value** (3.17) to each of a collection of features

NOTE 1 Each value is either a feature structure or a simpler **built-in** (3.6) such as a string.

NOTE 2 Feature structures are partially ordered. The minimal feature structures in this ordering are the empty feature structures.

3.15
feature system
type hierarchy (3.26) in which each **type** (3.24) has been associated with a collection of **admissibility constraints** (3.1) and **implicational constraints** (3.18)

NOTE cf. **type declaration** (3.25)

3.16
feature system declaration
FSD
specification of a particular **feature system** (3.15)

3.17
feature value
value
entity or aggregation of entities that characterize some property or aspect of another entity

3.18
implicational constraint
constraint of the form, "if G, then H," where G and H are **feature structures** (3.14)

NOTE This identifies any feature structure F as invalid for which G subsumes F, and yet F and H have no valid extension in common. See **subsumption** (3.21) and 8.5. Often used to refer to implicational constraints that are not also admissibility constraints.

3.19
interpretation
minimally informative (or equivalently, most general) **extension** (3.11) of a **feature structure** (3.14) that is consistent with a set of constraints declared by an **FSD** (3.16)

3.20
partial order
partially ordered set
set S equipped with a relation $\leq$ over $S \times S$ that is (1) reflexive (for all $s \in S$, $s \leq s$), (2) anti-symmetric (for all p, $q \in S$, if $p \leq q$ and $q \leq p$, then $p = q$), and (3) transitive (for all p, q, $r \in S$, if $p \leq q$ and $q \leq r$, then $p \leq r$)

NOTE The set of integers Z is partially ordered, but it has an additional property: for every p, $q \in Z$, either $p \leq q$ or $q \leq p$. Not all partial orders have this property. The taxonomical classification of organisms into phyla, genera and species, for example, is a partial order that does not. Type hierarchies may not necessarily. The typed feature structures of a feature system do not, unless (a) their type hierarchy does, and (b) either the type hierarchy has exactly one type, or every y type is constrained to have exactly one appropriate feature.

3.21
subsumption
property that holds between two feature structures, G and F, such that G is said to subsume F if and only if F carries all of the information with it that G does

NOTE A formal definition is provided in 5.6.

3.22
subtype
type (3.24) to which another type confers its constraints and appropriate features

3.23
supertype
base type
type (3.24) from which another type inherits constraints and appropriate features

NOTE s is a subtype of t iff t is a supertype of s. Every type is a subtype and supertype of itself.

3.24
semantic type
type
referring expression that distinguishes a collection of **feature structures** (3.14) as an identifiable and conceptually significant class

NOTE As implied by the name *semantic type*, types in this part of ISO 24610 do not serve to distinguish feature structures or their specifications syntactically.

3.25
type declaration
structure that declares the **supertypes** (3.23), **admissible features** (3.2), **admissible feature values** (3.3), **admissibility constraints** (3.1) and **implicational constraints** (3.18) for a given **type** (3.24)

NOTE The constraints on a type in the resulting feature system are those that have been declared in its declaration, in addition to those that it has inherited from its supertypes.

3.26
type hierarchy
partial order (3.20) over a set of **types** (3.24)

NOTE See ISO 24610-1:2006, Annex C, *Type inheritance hierarchies*.

3.27
typed feature structure
TFS
feature structure (3.14) that bears a **type** (3.24)

3.28
typing
assignment of a semantic **type** (3.24) to a **built-in** (3.6) or **feature structure** (3.14), either atomic or complex

NOTE Semantic types in feature systems are partially ordered, with multiple inheritance.

3.29
underspecification
provision of partial information about a **value** (3.17)

NOTE An underspecification generally subsumes one of a range of candidate values that could be resolved to a single value through subsequent constraint resolution. See **subsumption** (3.21).

3.30
well-formedness
syntactic conformity of a **feature structure** (3.14) representation to ISO 24610-1

3.31
validity
conformity of a **typed feature structure** (3.27) to the **constraints** (3.8) of a particular **feature system** (3.15)

NOTE See Clause 6.

4 Overall structure

The main part of the document consists of four clauses: Clauses 5, 6, 7 and 8.

— Clause 5, *Basic concepts*, reviews the definition of typed feature structures and the notions of atomic and complex types, collections and other operators that may appear in feature values. It then describes the notions of type inheritance hierarchies, type constraints, default values and underspecification that are essential to the construction of feature systems.

— Clause 6, *Defining well-formedness versus validity*, discusses the conditions of well-formedness and validity.

— Clause 7, *A feature system for a grammar*, illustrates how to define types with a type hierarchy and type constraints which declare what features and values are admissible for specific types.

— Finally, Clause 8, *Declaration of a feature system*, discusses how a feature system can be declared and developed into a validator.

The main part of the document is followed by two annexes: Annex A contains the XML schema for this part of ISO 24610; Annex B contains a complete example.

5 Basic concepts

5.1 Typed feature structures reviewed

Typed feature structures (TFSs) are introduced as basic records for language resource management.

For more information, refer to ISO 24610-1:2006, 4.7, *Typed feature structure,* and Annex C, *Type inheritance hierarchies*.

Here, a TFS is formally defined as a tuple over a finite set **Feat** of features, a collection X of non-feature-structure elements, and a type hierarchy **Type**, ≤, where **Type** is a finite set of types and ≤ is a subtyping relation over **Type**.

A feature structure is a tuple <Q, γ, θ, δ>, in which

a) Q is a set of nodes,

b) γ ∈ Q is the root node of the feature structure,

c) θ : Q → **Type** is a partial typing function, and

d) δ : **Feat** × Q → Q ∪ X is a partial feature value function,

such that, for all $q \in Q$, there exists a path of features $F_1, ..., F_n$ such that $\delta[F_n, ... \delta(F_1, \gamma) ...] = q$.

<fs> elements denote nodes. This definition deviates from the standard one used in linguistics and theoretical computer science in that (1) typing is partial, not total, i.e. not all feature structures have types, and (2) feature values might not be feature structures, but instead be drawn from a collection denoted by other XML elements such as string, numeric, symbol, and binary (the X above). Note that nodes are typed, but features themselves are not.

The following XML representation of a feature structure is considered well-formed, where the attribute type is assigned to each of the two <fs> elements.

EXAMPLE Typed feature structure:

```
<fs type="word">
  <f name="orth">
    <string>had</string>
  </f>
  <f name="morphoSyntax">
    <fs type="verb">
      <f name="tense">
        <symbol value="past"/>
      </f>
      <f name="auxiliary">
        <binary value="false"/>
      </f>
    </fs>
  </f>
</fs>
```

The feature name ORTH above stands for orthography, the conventional written form of a word or phrase.

This XML representation shows how the morpho-syntactic features of an English word “had” are specified as a past-tensed and non-auxiliary verb.

In the alternative, “matrix” or “AVM” notation, type names are conventionally in the lower-case, sometimes italicized or in the text type font, feature names in the upper-case, and strings in quotes. Binary values are

indicated with + or –. These conventions are followed in this document, too. The above feature structure would be depicted in matrix notation as shown in Figure 1.

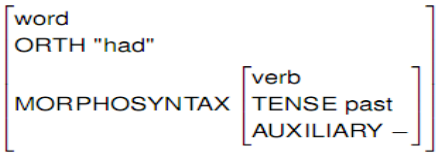

Figure 1 — Matrix notation

5.2 Types

5.2.1 Atomic types

Alongside the built-ins (<symbol>, <string>, <numeric> and <binary>), it is possible for a feature structure to have a type but no features. These are called simple or atomic feature structures, and types that allow for no features in their feature system declaration (FSD) are called atomic types.

There is, as a result, always the possibility of declaring new atomic types and using these instead of the above-mentioned built-ins to specify simple values. The above feature structure, for example, could have instead been rendered as follows, assuming the extra types had, past and false were declared in an FSD.

EXAMPLE Typed feature structure: alternative formulation

```
<fs type="word">
 <f name="orth">
  <fs type="had"/>
 </f>
 <f name="morphoSyntax">
  <fs type="verb">
   <f name="tense">
    <fs type="past"/>
   </f>
   <f name="auxiliary">
    <binary value="false"/>
   </f>
  </fs>
 </f>
</fs>
```


There is a difference also noticed between the two classes of built-ins: <string> on the one hand, and <symbol>, <binary> and <numeric> on the other. Any kind of string is permissible as the content of the <string> element, whereas a very restricted set of values is permissible in <symbol>, <binary> and <numeric> elements. To reflect this difference, members of the latter class specify their values using the attribute *value*. The type <binary>, for instance, is associated with four values: true, false, plus (equivalent to true) and minus (equivalent to false).

NOTE ISO 24610-1:2006 introduced the type *binary*, but the W3C's XML schema (2001) names it *boolean*.

It is the duty of the encoder to choose between atomic-type encodings and built-in encodings consistently. This part of ISO 24610 does not regard one as identical or even consistent with the other.

5.2.2 Complex types

Types that are not atomic are called complex. These include all of the types declared by the encoder in an FSD that declare or inherit admissible features. A feature is only admissible to a type if feature structures of that type are permitted by the FSD to have values for that feature. This does not mean that well-formed feature structures cannot arbitrarily associate types with feature structures regardless of their featural content – they can. But only those feature structures that use only admissible features to their type, as specified by

some FSD, could be validated against that FSD. The distinction between validity and well-formedness is further elaborated upon in Clause 6.

All user-declared types, no matter whether they are atomic or complex, are semantic, i.e. syntactically, they look no different from each other, apart from the value of their type attribute. It is the role of a validator to interpret the real significance of these types through enforcing restrictions on admissibility, restrictions on the possible values that admissible features can have (<vRange>), and other constraints that take the form of logical implications. All of these are specified, for each type, in an FSD.

The built-ins defined by the ISO 24610-1:2006 feature structure representations (FSRs) standard are purely syntactic. They can be used without declaration in an FSD, and they cannot be declared in an FSD. They can appear in value range restrictions, or in implicational constraints, but they cannot have such restrictions (since they have no admissible features) or constraints of their own.

5.2.3 Collections

Not all built-ins are as simple as those mentioned above, however. Some grammatical features such as specifiers (SPR), complements (COMPS) and arguments (ARGS) are considered as having a list of grammatical values, especially in Head-driven Phrase Structure Grammars (Pollard and Sag 1994[10]; Sag, Wasow, Bender 2003[12]). For languages other than English, some of these features may take other kinds of collections, namely sets or multisets, as their value. In a language (e.g. German, Korean or Japanese) that allows a relatively free word order, the feature COMPS may be analysed as taking a set or multiset, instead of a list, of complements. For more general applications, ISO 24610-1:2006 thus introduces sets and multisets as well as lists as built-in ways of assembling complex feature values.

Collections (<vColl>; ISO 24610-1:2006, 5.8, *Collections as complex feature values*) take the organization (org) attribute, with the values “list”, “set” and “bag”. In lists, order and multiplicity of elements matter. In bags, only multiplicity matters (these are often called multisets). In sets, neither order nor multiplicity matter.

For example, the feature ARGS of verbs can be represented by specifying the organization of <vColl> as a list of values, each of which is of type *phrase*.

EXAMPLE List value

```
<fs type="word">
 <f name="orth">
  <string>put</string>
 </f>
 <f name="args">
  <vColl org="list">
   <fs type="phrase">
    <vLabel name="L1"/>
    <f name="nominal">
     <binary value="plus"/>
    </f>
   </fs>
   <fs type="phrase">
    <vLabel name="L2"/>
    <f name="nominal">
     <binary value="plus"/>
    </f>
   </fs>
   <fs type="phrase">
    <vLabel name="L3"/>
    <f name="prepositional">
     <binary value="plus"/>
    </f>
   </fs>
  </vColl>
 </f>
</fs>
```

Some would call the type of this collection *list (phrase)*, but polymorphic lists are not yet supported in this part of ISO 24610. This is equivalent to the following AVM notation, where NP stands for a feature structure of the type *phrase* with a positive NOMINAL feature, namely a noun phrase, and PP, a feature structure of the type *phrase* with a positive PREPOSITIONAL feature, namely a prepositional phrase. The boxed integers are the labels for marking structure sharing as shown in Figue 2.

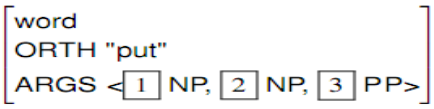

Figure 2 — Marking structure sharing

5.2.4 Operators

The other class of built-ins are operators that take one or more built-ins or feature structures as arguments, but instead of constructing a collection from them, denote a value that is in some other way derived from them.

Alternations (<vAlt>; ISO 24610-1:2006, 5.9.2, *Alternations*) denote one of their arguments' values. A feature structure containing an alternation does not denote multiple feature structures, however. An alternation is a single value that underspecifies which of several possible alternatives it is. Alternations can be regarded as the joins of their arguments in the partial order induced by subsumption (see 5.6).

Negations (<vNeg>; ISO 24610-1:2006, 5.9.3, *Negation*) take a single argument, and denote a value which is not its argument. A negation is equivalent to an alternation among all values that are inconsistent with its argument. A negation is actually not a logical negation of a value, but rather the complement of that value in the full Boolean lattice that contains the partial order induced by subsumption.

A **merge** (<vMerge>; ISO 24610-1:2006, 5.9.4, *Collection of values*) denotes the concatenation or union of several values and/or collections of values, according to how its *org* attribute is set. *org* takes the same values, with the same meanings, as in <vColl>.

5.3 Type inheritance hierarchies

The type hierarchy <Type, ≤> is discussed in great length in ISO 24610-1:2006, Annex C *Type inheritance hierarchies*. This structure is normally depicted as a directed acyclic graph with a unique top node. The label of this top node is often top, and represents the most general type, the type that is consistent with all typed feature structures. Subtypes are connected to, and appear below, their supertypes. The most specific types appear at the bottom of the graph. These are mutually incompatible with each other, which is generally understood implicitly, or, on occasion, depicted by another special type, bottom as the unique bottom-most element. Bottom is not used in this part of ISO 24610.

Figure 3 provides an example that depicts a part of the natural world:

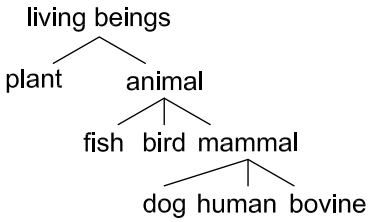

Figure 3 — Type hierarchy for living beings

According to this picture, living beings consist of plants and animals. Animals are subclassified into fish, birds and mammals. Dogs, humans and bovines (oxen, cows, bulls) belong to the class of mammals.