TECHNICAL REPORT

ISO/IEC TR 22250-1

First edition
2002-02-15

Information technology — Document description and processing languages — Regular Language Description for XML (RELAX) —

Part 1: **RELAX Core**

Technologies de l'information — Description de documents et langages de traitement — Description de langage courant pour XML (RELAX) —

Partie 1: Noyau RELAX

Reference number
ISO/IEC TR 22250-1:2002(E)

Contents

Page

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 3.

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 national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC TR 22250 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 22250-1, which is a Technical Report of type 3, was prepared by JISC (as JIS/TR X 0029) and was adopted, under a special "fast-track procedure", by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

ISO/IEC TR 22250 consists of the following parts, under the general title *Information technology — Document description and processing languages — Regular Language Description for XML (RELAX)*:

— *Part 1: RELAX Core*

— *Part 2: RELAX Namespace*

Information technology — Document description and processing languages — Regular Language Description for XML (RELAX) —

Part 1: **RELAX Core**

1 Scope

This Technical Report gives mechanisms for formally specifying the syntax of XML-based languages. For example, the syntax of XHTML 1.0 can be specified in RELAX.

Compared with DTDs, RELAX provides the following advantages:

- Specification in RELAX uses XML instance (i.e., document) syntax,
- RELAX provides rich datatypes, and
- RELAX is namespace-aware.

The RELAX specification consists of two parts, *RELAX Core* and *RELAX Namespace*. This part of the Technical Report gives *RELAX Core,* which may be used to describe markup languages containing a single XML namespace. Part 2 of this Technical Report gives *RELAX Namespace*, which may be used to describe markup languages containing more than a single XML namespace, consisting of more than one *RELAX Core* document.

Given a sequence of elements, a software module called the *RELAX Core* processor compares it against a specification in *RELAX Core* and reports the result. The *RELAX Core* processor can be directly invoked by the user, and can also be invoked by another software module called the *RELAX Namespace* processor.

RELAX may be used in conjunction with DTDs. In particular, notations and entities declared by DTDs can be constrained by RELAX.

This part of the Technical Report also gives a subset of *RELAX Core*, which is restricted to DTD features plus datatypes. This subset is very easy to implement, and with the exception of datatype information, conversion between this subset and XML DTDs results in no information loss.

NOTE 1 Since XML is a subset of WebSGML (TC2 of ISO 8879), RELAX is applicable to SGML.

NOTE 2 A successor of RELAX Core is being developed at the RELAX NG TC of OASIS.

2 References

ISO 8879:1986, *Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*

ISO 8879:1986/Cor 2:1999, *Information processing – Text and office systems – Standard Generalized Markup Language (SGML) – Technical Corrigendum 2*

W3C (World Wide Web Consortium), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/REC-xml, 2000

W3C (World Wide Web Consortium), Name Spaces in XML, W3C Recommendation, http://www.w3.org/TR/REC-xml-names, 1999

W3C (World Wide Web Consortium), XML Information Set, W3C Proposed Recommendation, http://www.w3.org/TR/xml-infoset, 2001

W3C (World Wide Web Consortium), XML Schema Part 2, W3C Recommendation, http://www.w3.org/TR/xmlschema-2, 2001

IETF (Internet Engineering Task Force). RFC2396: Uniform Resource Identifiers (URI): Generic Syntax, 1998.

3 Terms and definitions

3.1 XML 1.0

For the purposes of this part of the Technical Report, the following terms and definitions given in XML 1.0 apply.

a) start tag

b) end tag

c) empty-element tag

d) attribute

e) attribute name

f) content

g) content model

h) attribute-list declaration

i) DTD

j) XML processor

k) validity

l) validating processor

m) non-validating processor

n) whitespace

o) child

p) parameter entity

q) match

NOTE 3 On top of those meanings given in XML 1.0, “match” has another meaning (see 5.8).

3.2 Name Spaces in XML

For the purposes of this part of the Technical Report, the following terms and definitions given in “Name Spaces in XML” apply.

a) namespace

b) namespace name

3.3 XML Schema Part 2

For the purposes of this part of the Technical Report, the following terms and definitions given in "XML Schema Part 2" apply.

a) lexical representation

b) facet

c) datatype

d) built-in datatype

3.4 XML Information Set

For the purposes of this part of the Technical Report, the following terms and definitions given in “XML Information Set” apply.

a) information set

b) document information item

c) element information item

d) property

e) core property

f) reference to skipped entity information item

g) entity information item

h) notation information item

3.5 Definitions specific to *RELAX Core*

3.5.1
tag name
names in start tags, end tags, and empty-element tags (generic identifiers in ISO 8879)

NOTE 4 This term is adopted from DOM.

3.5.2
hedge
ordered sequences of elements and character data

4 Notations

This part of the Technical Report uses DTD in order to specify the syntax of RELAX modules. However, since DTDs provide no support for XML namespaces, this part of the Technical Report only uses some of the constructs possible in DTDs.

To specify permissible contents of elements, this part of the Technical Report uses content models, which match the non-terminal symbol `contentspec` in XML 1.0.

EXAMPLE 1 The following content model specifies that an element is constrained to a sequence beginning with a **frontmatter** element followed by a **body** element, and finally an optional **backmatter** element.

```
(frontmatter, body, backmatter?)
```

To specify permissible attributes of elements, this part of the Technical Report uses fragments of attribute-list declarations, which match the non-terminal symbol `AttDef` of XML 1.0.

EXAMPLE 2 The following attribute-list fragment specifies that an element has an optional attribute **class** and that any character string can be used as the attribute value.

```
class CDATA #IMPLIED
```

Elements representing RELAX modules shall belong to the namespace "http://www.xml.gr.jp/xmlns/relaxCore". There shall be no constraints on subelements not belonging to this namespace and no constraints on qualified attributes.

5 Basic concepts

5.1 Design principles

The design principles of *RELAX Core* are:

a) *RELAX Core* shall be simple and powerful.

b) The design shall be prepared quickly.

c) The design shall be formal and concise.

d) It shall be possible to implement *RELAX Core* using existing XML document APIs (e.g., SAX and DOM).

e) *RELAX Core* shall be upward-compatible with DTDs.

f) *RELAX Core* shall have a subset such that conversion to and from DTDs loses no information except datatype information.

g) Datatypes of *RELAX Core* shall be compatible with those in XML Schema Part 2.

NOTE 5 “HOW TO RELAX” [3] is a tutorial of RELAX Core.

5.2 Instances, schemas, and meta schemas

5.2.1 Instances

A document information item is said to be an *instance*. When an instance satisfies conditions represented by a RELAX schema, the instance is said to *comply* or be *compliant with* the RELAX schema. If there is no possibility of confusion, the instance may be considered compliant without mention of the RELAX schema.

NOTE 6 A valid document as defined in XML 1.0 (to be precise, a document information item represented by this document) need not be compliant with a RELAX schema; an instance compliant with a RELAX schema (to be precise, documents representing this instance) need not be valid.

5.2.2 RELAX schema

A RELAX schema is a description of permissible elements, attributes, and their structural relationships.

5.2.3 RELAX meta schema

The *RELAX meta schema* is a RELAX schema specifying the syntax of RELAX. Any RELAX schema is compliant with the RELAX meta schema.

5.3 Modules and frameworks

A document information item that conforms to *RELAX Core* is said to be a *RELAX module*. A RELAX module addresses elements in a single namespace as well as their attributes and contents.

A document information item that conforms to *RELAX Namespace* is said to be a *RELAX framework.* A RELAX framework addresses multiple namespaces by specifying a RELAX module per each namespace.

A single-namespace RELAX schema consists of a framework and a single module. Since the framework does not reference to other modules, the module provides the complete schema definition.

A multiple-namespace RELAX schema consists of a framework and modules referenced from the framework.

5.4 Islands and instances

A multi-namespace instance is compared against a RELAX schema comprising multiple modules. Such an instance is first decomposed into multiple *islands*, each of which is a single-namespace hedge. Each island is then compared against a single RELAX module (Figure 1).

$$\begin{tikzcd}
\text{module} \arrow[d] & \text{island} \arrow[l, "\textit{compared against}"'] \arrow[d] \\
\text{framework} & \text{instance} \arrow[l, "\textit{compared against}"']
\end{tikzcd}$$

Figure 1 — The relationship between modules/frameworks and islands/instances

A single-namespace instance is already an island, and thus need not be further decomposed.

5.5 Behaviour of the *RELAX Core* processor

The *RELAX Core* processor is a software module that, given an island and a RELAX module, compares the island against the RELAX module in order to determine if the island is compliant with the RELAX module.

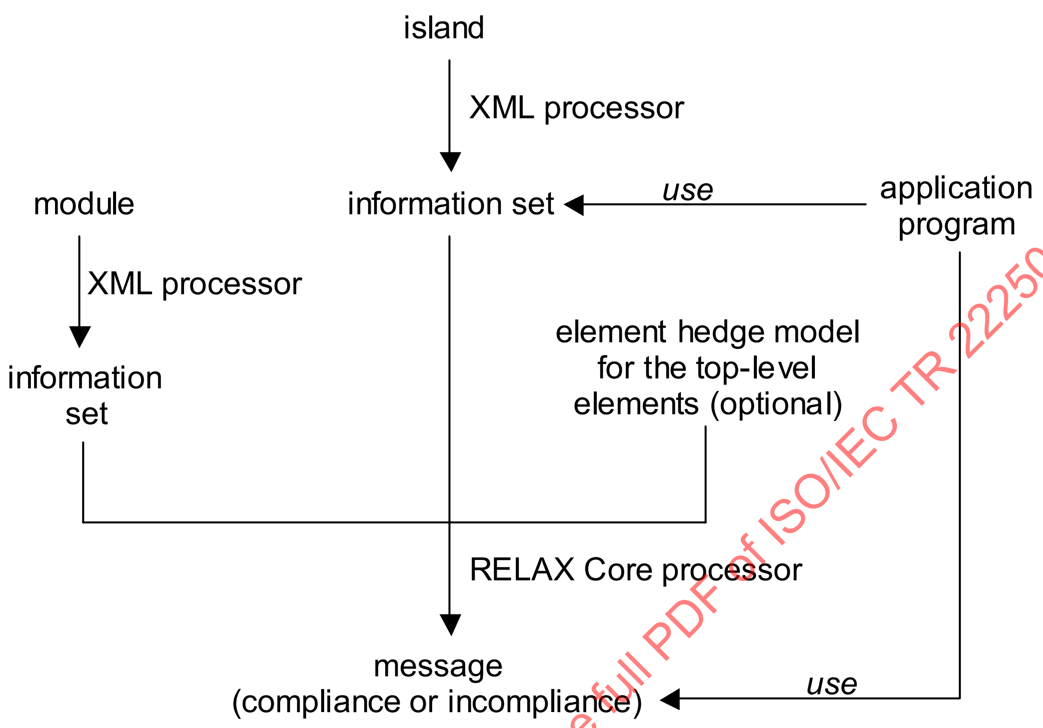

Figure 2 — The *RELAX Core* processor, the XML processor, and application programs

The *RELAX Core* processor shall receive islands and RELAX modules as information sets from the XML processor. The *RELAX Core* processor shall use core properties of information items in the information sets and shall not use other properties.

NOTE 7 Implementations of the *RELAX Core* processor receive information sets via APIs such as SAX or DOM.

The *RELAX Core* processor may also receive a hedge model that constrains the top-level elements of islands.

After comparison, the *RELAX Core* processor shall output a message that the island is compliant or a message that it is not. The *RELAX Core* processor may output other messages.

Other than such messages, the *RELAX Core* processor shall have no outputs. Application programs shall receive information sets from the XML processor and may receive messages from the *RELAX Core* processor.

Both validating processors and non-validating processors may be used by the *RELAX Core* processor.

When the *RELAX Core* processor receives references to skipped entity information items, it shall output a message, at user option, and may stop normal processing.

5.6 Datatypes

RELAX Core uses the built-in datatypes of XML Schema Part 2. Datatypes can be used as conditions on attributes or used as hedge models.

Datatypes in *RELAX Core* represent sets of strings. Given a string and a datatype, it is possible to determine if the string is contained by the set of strings represented by that datatype.

EXAMPLE 3 The datatype named **integer** defines a set of strings representing integers. It is possible to determine whether or not a string represents an integer.

References to datatypes may have additional conditions called *facets*.

EXAMPLE 4 A reference to the **integer** datatype may have facets, which may specify that the value should be equal to or greater than 10, and that it should be equal to or less than 20.

The *RELAX Core* processor does not convert character strings to data (e.g., conversion of the string "1" to the integer 1). Such conversion is left to application programs.

NOTE 8 Typically, application programs merely invoke conversion libraries.

5.7 Roles and clauses

In *RELAX Core*, conditions on tag names and attributes are captured by *roles* and *clauses*. A role is a name, and is described by a clause. A clause does not have a name.

A clause is either **tag** or **attPool**. **tag** specifies permissible tag name, while **attPool** does not. When **tag** or **attPool** specify a permissible attribute, this attribute is said to be *declared* by the **tag** or **attPool**.

There shall be at most one clause per role.

A clause may reference to another clause via a role. A clause shall not directly or indirectly reference to itself. A referenced clause shall be described by **attPool**; it shall not be described by **tag**.

A clause shall not directly or indirectly (via other clause) reference to another clause more than once.

A clause shall not directly or indirectly (via other clause) declare an attribute more than once.

5.8 Production rules, labels, and hedge models

5.8.1 General

In *RELAX Core*, conditions on element structures are captured by *labels* and *production rules*. A label is a name, and is described by a production rule. A production rule shall not have a name.

A production rule is either **elementRule** or **hedgeRule**. **elementRule** is a triplet of a label, role, and hedge model. **hedgeRule** is a pair, consisting of a label and a hedge model.

Roles referenced by **elementRule** shall be described by **tag** clauses. **elementRule** shall not reference roles described by **attPool** clauses.

More than one **elementRule** may share a label, and more than one **hedgeRule** may share a label. However, **elementRule** and **hedgeRule** shall not share a label.

More than one **elementRule** may share a role.

NOTE 9 Regular grammars comprise production rules and generate sets of strings. The left-hand side of a production rule is a non-terminal symbol, and the right-hand side is either a terminal symbol, a terminal symbol followed by a non-terminal symbol, or a non-terminal symbol. *RELAX Core* is an extension of regular grammars such that sets of logical structures are generated. A label corresponds to a non-terminal symbol in the left-hand side, and a role corresponds to a terminal symbol. Hedge models extend non-terminal symbols in the right-hand side so that **elementRule** addresses tree structures rather than strings.

Permissible contents of elements are described by *hedge models*. A hedge model shall be either an element hedge model, mixed hedge model, or datatype reference.

When more than one **elementRule** shares a label and role together, one of the following conditions hold:

a) all of them have element hedge models,

b) all of them have mixed hedge models, or

c) all of them reference the same datatype, possibly having different facets.

5.8.2 Element hedge models

An element hedge model generates a regular set of label sequences. Any label sequence in this set, possibly prepended, interspersed with, or followed by whitespace characters, is said to *match* this element hedge model.

5.8.3 Mixed hedge models

A mixed hedge model is an element hedge model with the **mixed** wrapper. Any label sequence in the set generated by the wrapped element hedge model, possibly prepended, interspersed with, or followed by arbitrary characters, is said to *match* this mixed hedge model. Note that not only whitespace characters but also non-whitespace characters are permitted.

5.8.4 Datatype references

A datatype reference specifies a datatype name. It may further specify additional conditions called facets. A sequence of characters *matches* a datatype reference if the character sequence belongs to the referenced datatype and satisifies the accompanying facets, if any.

5.9 Taxonomy and occurrences of names

Five types of names are used in *RELAX Core*. They are datatype names, tag names, attribute names, roles, and labels. Names of different types do not collide. For example, although there is a datatype called **integer**, one may use "integer" as a tag name or label.

The following table shows which types of names appear in instances and where in RELAX modules they appear.

Table 1 — Types of names and their occurrences in instances and RELAX modules

Types of names	In Instances	In RELAX Modules
datatype names	do not occur	occur as conditions on attributes or hedge models (datatype references)
tag names	occur	occur as part of clauses
attribute names	occur	occur as part of clauses
roles	do not occur	occur in clauses (description of roles) occur in clauses (references to roles)
labels	do not occur	occur in production rules (description of labels) occur in production rules (reference to labels)

6 Module Constructs

6.1 module

module represents an entire module. This element provides management information about the module.

module has the **moduleVersion** attribute the ***relaxCore*Version** attribute, and the **targetNamespace** attribute.

```
moduleVersion CDATA #IMPLIED
relaxCoreVersion CDATA #REQUIRED
targetNamespace CDATA #IMPLIED
```

The version of this module is indicated by the **moduleVersion** attribute.

The version of *RELAX Core* is indicated by the **relaxCoreVersion** attribute. The version number "1.0" shall be used to indicate conformance to Version 1.0 of *RELAX Core*; it is an error for a document to use the value "1.0" if it does not conform to Version 1.0 of *RELAX Core*.

NOTE 10 It is the intent of the editing committee to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of *RELAX Core*, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary.

The *RELAX Core* processor may signal an error if it receives documents labelled with versions that it does not support. It may continue or abort normal processing.

The **targetNamespace** attribute specifies the target namespace to which elements described by this module belong. When this attribute is specified and this module is included by another module, the value of this attribute shall be identical to the target namespace of that module in effect. When this attribute is not specified and this module is included by another module, the target namespace of that module in effect shall be used as the target namespace. When this attribute is not specified but this module is not referenced by a RELAX framework or module, “” is used as the target namespace

Given that all elements in the module itself belong to the namespace "http://www.xml.gr.jp/xmlns/relaxCore", the **module** element shall declare this namespace.

Every element in a module shall belong to the namespace "http://www.xml.gr.jp/xmlns/relaxCore". Thus, the module element shall declare this namespace.

The following content model describes the permissible content of **module** elements:

```
(annotation?, interface?,
  (tag | attPool | elementRule | hedgeRule | div | include )*)
```

EXAMPLE 5 An example **module** element is shown below. Child elements are omitted for clarity.

```
<module
    moduleVersion="1.2"
    RELAX CoreVersion="1.0"
    xmlns="http://www.xml.gr.jp/xmlns/relaxCore">
    ...
</module>
```

6.2 interface

interface provides interface information between the module and RELAX frameworks. If a RELAX framework has only a single namespace, **interface** provides information about the permissible root element of instances.

interface has no attributes.

The following content model describes the permissible content of **interface** elements:

```
(annotation?, (export | div)*)
```

6.3 export

export indicates which information can be referenced from RELAX frameworks.

export has the **label** attribute.

```
label NMTOKEN #REQUIRED
```

The **label** attribute exposes a label described by **elementRule** to RELAX frameworks. If a RELAX framework has only a single namespace, this attribute shows that the root of instances may have a specified label.

The following content model describes the permissible content of **export** elements:

```
(annotation?)
```

EXAMPLE 6 An example of **export** is shown below.

```
<export label="doc"/>
```

6.4 tag

tag specifies the condition that elements play a specified role, by combining a condition on tag names, conditions on attribute values, and references to other roles.

tag has the role attribute and the name attribute.

```
role NMTOKEN #IMPLIED
name NMTOKEN #IMPLIED
```

The **name** attribute specifies the tag name. The **role** attribute specifies which role is described by the **tag**.

When the **tag** is not a child element of **elementRule**, the **name** attribute shall be specified but the **role** attribute need not be specified. If the **role** attribute is not specified, its value is assumed to be the same as the **name** attribute.

When the **tag** is a child element of **elementRule**, the **name** attribute need not be specified and the **role** attribute shall not be specified. An appropriate role which does not collide with other roles is generated as the value of the **role** attribute by the *RELAX Core* processor. If the **name** attribute is not specified, it is assumed to be the same as the **label** attribute of the parent **elementRule**.

The following content model describes the permissible content of **tag** elements. Subordinate **ref** shall specify the **role** attribute.

```
(annotation?, (ref | attribute)*)
```

An element *e* play the **role** specified by the **role** attribute if the following three conditions hold:

a) The tag name of *e* matches the value of the **name** attribute.

b) Each of the conditions (expressed by the subordinate **atttribute** elements) on attributes is satisfied by some attribute of *e*.

c) *e* plays all roles referenced by the subordinate **ref** elements.

Observe that *e* satisfies these conditions even if it has attributes not declared by the **tag.**

EXAMPLE 7 An example of **tag** is shown below. The **bar1** role is referenced by a subordinate **ref** element.

```
<tag name="foo" role="bar">
  <ref role="bar1"/>
</tag>
```

6.5 attPool

attPool specifies the condition that elements play a specified role, by combining conditions on attribute values, and references to other roles.

attPool has the **role** attribute.

```
role NMTOKEN #REQUIRED
```

The **role** attribute specifies which role is described by the **attPool**.

The following content model describes the permissible content of **attPool** elements. Subordinate **ref** shall specify the **role** attribute.

```
(annotation?, (ref | attribute)*)
```

An element *e* plays the role specified by the **role** attribute if the following two conditions hold:

a) Each of the conditions (expressed by the subordinate **atttribute** elements) on attributes is satisfied by some attribute of *e*.

b) *e* plays all roles specified by the subordinate **ref** elements.

EXAMPLE 8 An example of **attPool** is shown below. The **bar1** role is referenced by a subordinate ref element.

```
<attPool role="bar">
  <ref role="bar1"/>
</attPool>
```

6.6 ref with the role attribute

ref with the **role** attribute references a role described by **attPool**.

ref with the **role** attribute does not have other attributes.

```
role NMTOKEN #REQUIRED
```

The following content model describes the permissible content of **ref** elements having the **role** attribute:

```
EMPTY
```

Examples of **ref** with the **role** attribute are also contained in the examples of **tag** and **attPool**.

6.7 attribute

attribute describes conditions on attribute names and values. It further indicates whether the attribute is optional.

attribute has the **name** attribute, the **required** attribute, and the **type** attribute. The **name** attribute is mandatory.

```
name NMTOKEN #REQUIRED
required (true) #IMPLIED
type NMTOKEN #IMPLIED
```

The **name** attribute specifies the attribute name. The **required** attribute shows whether this attribute is optional. If "**true**" is specified, this attribute is mandatory.

Multiple **attribute** elements in a **tag** element or the **attPool** elements directly or indirectly referenced by this **tag** shall not specify names matching each other.

The **type** attribute specifies a datatype name. If the **type** attribute is omitted, the built-in datatype **string** is assumed.

The following content model describes the permissible content of **attribute** elements. It is assumed that all element types representing facets are connected by "|" and declared as the value of the facet parameter entity.

```
(annotation?, (%facet;)*)
```

The datatype name specified by the **type** attribute and the facets specified by the child elements collectively form a datatype reference. The value of the attribute specified by the **name** attribute is required to match this datatype reference.

EXAMPLE 9 An example of **attribute** is shown below. An **attribute** element is used as a child element of **tag**.

```
<tag name="a" >
  <attribute name="href" type="anyURI"/>
</tag>
```

NOTE 11 In XML 1.0, element types without attributes do not require attribute-list declarations. However, in *RELAX Core*, tag names occurring in instances always require **tag** elements.

6.8 elementRule

elementRule represents a production rule which consists of a triplet of a label, role, and hedge model.

elementRule has the role attribute, the **label** attribute and the **type** attribute. When **elementRule** does not have a subordinate **tag**, the **role** attribute shall be specified and the **label** attribute need not be specified. When the **label** attribute is omitted, it is assumed to have the same value as the **role** attribute. When **elementRule** has a subordinate **tag**, the **label** attribute shall be specified and the **role** attribute shall not be specified.

```
role NMTOKEN #IMPLIED
label NMTOKEN #IMPLIED
type NMTOKEN #IMPLIED
```

The **role** attribute specifies a role. The **label** attribute specifies which label is described by this **elementRule**. The **type** attribute references to a datatype. When this **elementRule** has an element hedge model or mixed hedge model, the **type** attribute shall not be specified.

The following content model describes the permissible content of **elementRule** elements. It is assumed that all element types representing facets are connected by "|" and declared as the value of the facet parameter entity. Subordinate **ref** shall specify the **label** attribute.

```
(annotation?, tag?,
  ((ref | hedgeRef | choice | sequence | element | none | empty | mixed)
   |
   (%facet;)*))
```

Handling of subordinate **tag** elements is described in 8.5.

elementRule is said to have an element hedge model if either **ref**, **hedgeRef**, **choice**, **sequence**, **element**, **none** or **empty** is specified as the child element.

elementRule is said to have a mixed hedge model if **mixed** is specified as the child element.

If **elementRule** does not have an element hedge model or mixed hedge model, a datatype reference shall be specified by the **type** attribute. When **elementRule** has a datatype reference, facets may be specified as the content of the **elementRule**. Datatypes and facets are described in Clause 7.

6.9 hedgeRule

hedgeRule represents a production rule which consists of a pair of a label and hedge model.

hedgeRule has the **label** attribute.

```
label NMTOKEN #REQUIRED
```

The **label** attribute specifies which label is described by the **hedgeRule**.

The following content model describes the permissible content of **hedgeRule** elements. Subordinate **ref** shall specify the label attribute.

```
(annotation?,
  (ref | hedgeRef | choice | sequence | element | none | empty))
```

6.10 ref with the label attribute

ref with the **label** attribute represents an element hedge model which references to a label not described by **hedgeRule**.

ref elements with the **label** attribute have the **occurs** attribute.

```
label NMTOKEN #REQUIRED
occurs CDATA #IMPLIED
```

The value of the **occurs** attribute shall be either "*", "+", or "?".

The following content model describes the permissible content of **ref** elements with the **label** attribute:

```
EMPTY
```

Let *l* be the label referenced by the **label** attribute of **ref**. When the **occurs** attribute is not specified, this **ref** shall generate a label sequence made up from one occurrence of *l* only. When the **occurs** attribute is specified, "*" shall repeat the sequence zero or more times, "+" shall repeat the sequence one or more times, and "?" shall repeat the sequence zero or one time.

6.11 hedgeRef

hedgeRef represents an element hedge model which references to a label described by some **hedgeRule** element.

hedgeRef has the occurs attribute.

```
label NMTOKEN #REQUIRED
occurs CDATA #IMPLIED
```

The value of the **occurs** attribute shall be either "*", "+", or "?".

The following content model describes the permissible content of **hedgeRef** elements:

```
EMPTY
```

hedgeRef is replaced by element hedge models of those **hedgeRule** elements which describe the label referenced by this **hedgeRef** (more about this, see 8.4).

6.12 sequence

sequence represents an element hedge model that concatenates element hedge models.

sequence has the **occurs** attribute. Permissible values and semantics of the **occurs** attribute are the same as in **ref** with the **label** attribute.

```
occurs CDATA #IMPLIED
```

The following content model describes the permissible content of **sequence** elements:

```
(ref | hedgeRef | choice | sequence | element | none | empty)*
```

Suppose that the child elements of **sequence** are $c_1, c_2, \ldots, c_m$. Further suppose that c_1 generates a label sequence $l_{11}, l_{12}, \ldots, l_{1i;}$ c_2 generates a label sequence $l_{21}, l_{22}, \ldots, l_{2j}$; c_3 and the following child elements also generates similar sequences, and c_m generates a label sequence $l_{m1}, l_{m2}, \ldots, l_{mj}$. When the **occurs** attribute is not specified, this **sequence** shall generate $l_{11}, l_{12}, \ldots, l_{1i}, l_{21}, l_{22}, \ldots, l_{2j}, \ldots, l_{m1}, l_{m2}, \ldots, l_{mk}$. When the **occurs** attribute is specified, "*" shall repeat the sequence zero or more times, "+" shall repeat the sequence one or more times, and "?" shall repeat the sequence zero or one time.

6.13 choice

choice represents an element hedge model that is a selection from element hedge models.

choice has the **occurs** attribute. Permissible values and semantics of the **occurs** attribute are the same as in **ref** with the **label** attribute.

```
occurs CDATA #IMPLIED
```

The following content model describes the permissible content of **choice** elements:

```
(ref | hedgeRef | choice | sequence | element | none | empty)*
```

Suppose that the child elements of **choice** are $c_1, c_2, \ldots, c_m$. Further suppose that c_1 generates a label sequence $l_{11}, l_{12}, \ldots, l_{1i;}$ c_2 generates a label sequence $l_{21}, l_{22}, \ldots, l_{2j}$; c_3 and the following child elements also generate similar sequences, and c_m generates a label sequence $l_{m1}, l_{m2}, \ldots, l_{mk}$. When the **occurs** attribute is not specified, this **choice** shall generate any of the label sequence $l_{11}, l_{12}, \ldots, l_{1i}$, the label sequence $l_{21}, l_{22}, \ldots, l_{2j}$, ..., or the label sequence $l_{m1}, l_{m2}, \ldots, l_{mk}$. When the **occurs** attribute is specified, "*" shall repeat the sequence zero or more times, "+" shall repeat the sequence one or more times, and "?" shall repeat the sequence zero or one time.

6.14 empty

empty represents an element hedge model that matches the empty sequence of labels.

empty has no attributes.

The following content model describes the permissible content of **empty** elements:

```
EMPTY
```

6.15 none

none represents an element hedge model that matches no label sequences.

none has no attributes.

The following content model describes the permissible content of **none** elements:

```
EMPTY
```

6.16 mixed

mixed provides a mixed hedge model.

mixed has no attributes.

The following content model describes the permissible content of **mixed** elements:

```
(ref | hedgeRef | choice | sequence | element | none | empty)
```

Suppose that the element hedge model which is the child of a mixed hedge model generates some label sequence. This label sequence, possibly prepended, intervened, or followed by arbitrary characters, matches the mixed hedge model.

6.17 element

element provides a convenient shorthand which is expanded to **ref**, **tag**, and **elementRule**.

element has the **name** attribute, the **type** attribute, and the **occurs** attribute. The **name** attribute and the **type** attribute shall be specified. Permissible values and semantics of the **occurs** attribute are the same as in **ref** with the **label** attribute.

```
name   NMTOKEN   #REQUIRED
type   NMTOKEN   #REQUIRED
occurs CDATA #IMPLIED
```

The following content model describes the permissible content of **element**. In this content model, it is assumed that all element types representing facets are connected by "|" and declared as the value of the facet parameter entity.

```
(annotation?, (%facet;)*)
```

element is expanded to **ref**, **tag**, and **elementRule** according to the following rule.

a) A **ref** element shall be created, and the **element** shall be replaced by this **ref**. An appropriate label which does not collide with other labels shall be created as the value of the **label** attribute of this **ref**. The **occurs** attribute of the **element**, if any, shall be copied to the **ref**.

b) An **elementRule** element shall be created, and shall be added to this module. An appropriate role which does not collide with other roles shall be created as the value of the **role** attribute of the **elementRule**. The value of the **label** attribute shall be the label created in a). The hedge model of the **elementRule** shall be a datatype reference. The name of the referenced datatype shall be the value of the **type** attribute of the **element**. The content of the **elementRule** shall be the content of the **element**.

c) A **tag** element shall be created, and shall be added to this module. The value of the **role** attribute of this **tag** shall be the role created in b). The value of the **name** attribute of this **tag** shall be the value of the **name** attribute of the original **element**.

6.18 include

include provides a mechanism for referencing other modules.

include has the **moduleLocation** attribute.

```
moduleLocation   CDATA   #REQUIRED
```

The **moduleLocation** attribute references another module via a URI reference (IETF RFC 2396). The URI reference shall not contain a fragment identifier.

The following content model describes the permissible content of **include** elements:

```
(annotation?)
```

The module which contains **include** is said to be a *referencing module*, and a module referenced by the **moduleLocation** attribute is said to be a *referenced module.* The referenced module and the referencing module are required to specify the same value by the **targetNamespace** attribute.

6.19 div

div is introduced as a mechanism for grouping clauses, production rules, and **include** elements, and grouping of **export** elements in **interface** elements.

div has no attributes.

The following content model describes the permissible content of **div** elements. **div** may have subordinate **export** only when **interface** is an ancestor of the **div**. When a **div** element does not have an **interface** ancestor, it may contain subordinate clauses, production rules, and **include** elements.

```
(annotation?, div*,
  (((elementRule | hedgeRule | tag | attPool | include),
     (elementRule | hedgeRule | tag | attPool | include | div)*)
   |
    (export, (export | div)*))?)
```

6.20 annotation

annotation is introduced as a mechanism for embedding comments in modules.

annotation has no attributes.

The following content model describes the permissible content of **annotation** elements:

```
(appinfo | documentation)*
```

6.21 documentation

documentation is introduced as a note for human users to read.

documentation has the **source** attribute.

```
source   CDATA   #IMPLIED
```

When the **source** attribute is specified, the value shall be a URI reference (see IETF RFC 2396).

The following content model describes the permissible content of **documentation** elements:

```
(#PCDATA)
```

A **documentation** element specifying the **source** attribute shall be an empty element. Non-empty **documentation** elements shall not specify the **source** attribute.

6.22 appinfo

appinfo is a mechanism for embedding information for user programs that handle modules.

appinfo has the **source** attribute.

```
source   CDATA   #IMPLIED
```

When the **source** attribute is specified, the value shall be a URI reference (see IETF RFC 2396).

The following content model describes the permissible content of **appinfo** elements:

```
(#PCDATA)
```

An **appinfo** element specifying the **source** attribute shall be an empty element. Non-empty **appinfo** elements shall not specify the **source** attribute.

7 Datatypes

7.1 General

Datatypes in this part of the Technical Report shall be either built-in datatypes of XML Schema Part 2 or datatypes specific to *RELAX Core*.

NOTE 12 At present, users are not allowed to define new datatypes.

7.2 Built-in datatypes of XML Schema Part 2

The built-in datatypes of XML Schema Part 2 are available. Implementation of the following datatypes are strongly recommended.

a) string

b) boolean

c) float

d) double

e) anyURI

f) normalizedString

g) token

h) language

i) NMTOKEN

j) NMTOKENS

k) Name

l) NCName

m) ID

n) IDREF

o) long

p) int

q) short

r) byte

s) unsingedLong

t) unsingedInt

u) unsingedShort, and

v) unsingedByte

The **ID**, **IDREF**, **IDREFS**, **ENTITY**, **ENTITIES**, **NOTATION**, **NMTOKEN**, **NMTOKENS** datatypes can be referenced by the **type** attribute of **attribute**, but cannot be referenced by the **type** attribute of **elementRule** or **element**.

A **tag** element and those **attPool** elements referenced from this **tag** directly or indirectly shall not declare more than one attribute as **ID**.

If multiple **tag** elements sharing the same tag name declare **ID** attributes directly or indirectly, these attributes shall be declared by some **attribute** element of a single **attPool** element, which shall be referenced by each of these **tag** elements directly or indirectly. If multiple **tag** elements sharing the same tag name declare **IDREF** attributes directly or indirectly, these attributes shall be declared by some **attribute** element of a single **attPool** element, each of which shall be referenced by each of these **tag** elements directly or indirectly. If multiple **tag** elements sharing the same tag name declare **IDREFS** attributes directly or indirectly, these attributes shall be declared by **attPool** elements, each of which shall be referenced by each of these **tag** elements directly or indirectly.

In a document information item, more than one element information item shall not specify the same name as values of **ID** attributes. In other words, the value of an **ID** attribute shall uniquely identify an element information item. The value of an **IDREF** attribute shall match the value of some **ID** attribute in the document information item. The value of an **IDREFS** attribute shall consist of names each of which shall match the value of some **ID** attribute in the document information item.

The value of an **ENTITY** attribute shall match the name of some entity information item in the document information item. The value of an **ENTITIES** attribute shall consist of names each of which shall match the name of some entity information item in the document information item.

The value of a **NOTATION** attribute shall match the name of some notation information item in the document information item.

7.3 Datatypes Specific to RELAX

7.3.1 none

This datatype represents an empty set of strings. No character strings belong to this datatype. This datatype has no applicable facets.

7.3.2 emptyString

This datatype represents a singleton set containing the empty string. This datatype has no applicable facets.

7.4 Facets

The facets of the built-in datatypes of XML Schema Part 2 are available. Implementation of the following facets are strongly recommended.

a) length

b) minLength

c) maxLength

d) enumeration

e) maxInclusive

f) maxExclusive

g) minInclusive, and

h) minExclusive

8 Reference model

8.1 General

This part of the Technical Report specifies the reference model in order to clarify the criteria for determining whether an island is compliant with a RELAX module and an optional element hedge model.

NOTE 13 This reference model does not constrain implementations.

8.2 Creation of element hedge models

In the absence of an element hedge model for the top-level elements of an island, an element hedge model shall be created from the **label** attribute of the **export** elements in the given module.

First, for each label specified by the **label** attribute of these **export** elements, a **ref** element that reference to this label shall be created. Next, an element hedge model shall be created by wrapping these **ref** elements with a **choice** element. The **ref** elements and **choice** shall not have the **occurs** attribute.

8.3 Expansion of include

Each **include** element shall be expanded to the referenced module. When a referenced module further references to another module, **include** elements in the referenced module shall be expanded in advance.

8.4 Expansion of element

element elements in element hedge models shall be expanded to **ref**, **elementRule** and **tag**.

8.5 Expansion of module

Each **hedgeRef** element shall be replaced by element hedge models of the **hedgeRule** elements referenced by the **hedgeRef**. Details are as below:

a) Locate all **hedgeRule** elements that describe the label referenced by this **hedgeRef** element.

b) Wrap the hedge models of these **hedgeRule** elements by a **choice** element.

c) Copy the **occurs** attribute of the **hedgeRef** element to this **choice** element.

d) Replace the **hedgeRef** element with this **choice** element.

If other **hedgeRef** elements appear in the **choice** element, they shall be recursively expanded.

8.6 Expansion of tag embedded in elementRule

Each **tag** element embedded in **elementRule** shall be moved so that it becomes a sibling element of this **elementRule**. A role which does not collide with other roles shall be created, and shall be specified as the value of the **role** attribute of the **tag** and that of the **elementRule**. If the **tag** does not have the **name** attribute, it shall have the value of the **label** attribute of the **elementRule**.

8.7 Interpretation

An *interpretation* of a hedge is a mapping from each element in this hedge to a role and a label.

A hedge is compliant if it has at least one sound interpretation. An interpretation is sound if the following conditions hold:

a) Each element plays the associated role.

b) Derivation at each element is correct.

c) The sequence of labels associated with the top-level elements match the given element hedge model.

Consider an element *e* and its children $e_1, e_2, \ldots, e_n$. Let t_0 be the character sequence preceding e_1 in *e*, let t_i be the character sequence occurring between e_i and e_{i+1} in *e*, and let t_n be the character sequence following e_n in *e*. By definition, $t_0, e_1, t_1, e_2, t_2, \ldots, e_n, t_n$ provides the content of *e*.

Let *l* be the label associated with *e*, and let $l_1, l_2, \ldots, l_n$ be the labels associated with $e_1, e_2, \ldots, e_n$, respectively. Derivation at *e* is correct if there exists some **elementRule** such that the value of its label attribute is *l*, the value of its **role** attribute is the role associated with *e*, and $t_0, l_1, t_1, l_2, t_2, \ldots, l_n, t_n$ matches its hedge model.

At user option, the *RELAX Core* processor shall output message, when some of the attributes of *e* is not directly or indirectly (via some **attPool**) declared by the **tag** element describing the role associated with *e*.

Labels l_1 and l_2 not described by **hedgeRule** are said to be *contextually indistinguishable* when the following conditions hold:

a) The hedge model of some **elementRule** p_1 references to l_1.

b) The hedge model of some **elementRule** p_2 references to l_2.

c) Either p_1 and p_2 are identical, or their **role** and **label** attributes specify the same role and label, respectively.

NOTE 14 Every label is contextually indistinguishable from itself, if it is described by **elementRule** and referenced by some hedge model.

If it is possible to construct an element that plays both r_1 and r_2, they are said to be *coexistent*.

NOTE 15 Every role is coexistent with itself.

If the **role** and **label** attributes of some **elementRule** specify role *r* and label *l*, respectively, *r* is said to *lead to l*,.

The *RELAX Core* processor should but need not continue normal processing, when a RELAX module does not satisfy the following uniqueness condition.

a) If labels l_1 and l_2 are contextually indistinguishable, roles r_1 and r_2 are coexistent, and r_1 and r_2 lead to l_1 and l_2, respectively, then l_1 and l_2 are identical and r_1 and r_2 are identical.

The *RELAX Core* processor may output message and stop normal processing, when it receives a RELAX module not satisfying this uniqueness condition. The *RELAX Core* processor should output warning message when the uniqueness condition does not hold.

9 Conformance

9.1 General

This part of the Technical Report defines conformance levels of RELAX modules and conformance levels of the *RELAX Core* processor.

9.2 Conformance levels of RELAX modules

This part of the technical report defines two conformance levels of RELAX modules: "classic" and "fully relaxed".

The conformance level "classic" shall have restrictions as below:

a) **elementRule** shall not have the **label** attribute.

b) more than one **elementRule** shall not specify the same role.

c) more than one **hedgeRule** shall not specify the same label.

d) **tag** shall not specify the **role** attribute.

e) more than one **tag** shall not specify the same tag name.

f) **element** shall not exist

g) **tag** shall not exist as a child element of **elementRule**.

h) The child element of **mixed** shall be either **ref** with the **label** attribute, **hedgeRef**, or **choice**. They shall specify "*" as the value of the **occurs** attribute. When **choice** is the child element, item I) shall apply. When **hedgeRef** is the child element, item J) shall apply.

i) Child elements of those **choice** elements shown in item H) shall be either **ref** with the **label** attribute or **hedgeRef**. These child elements shall not specify the **occurs** attribute. When **hedgeRef** is a child element, item J) shall apply.

j) **hedgeRule** referenced by those **hedgeRef** elements shown in items H) and I) shall have either **ref** with the **label** attribute, **hedgeRef**, or **choice** as a hedge model. They shall not specify the **occurs** attribute. When **hedgeRef** is the hedge model, this item shall apply recursively.

k) Datatypes shall be restricted to **string**, **boolean**, **float**, **double**, **long**, **int**, **short**, **byte**, **ID**, **IDREF**, **ENTITY**, **NOTATION**, **IDREFS**, **ENTITIES**, **NMTOKEN**, and **NMTOKENS**.

l) Facets shall be restricted to **enumeration**, **maxInclusive**, **maxExclusive**, **minInclusive**, and **minExclusive**.

The conformance level "fully relaxed" shall not have any of the above restrictions.

9.3 Conformance levels of the *RELAX Core* processor

This part of the Technical Report defines two conformance levels for the *RELAX Core* processor: "classic" and "fully relaxed".

The *RELAX Core* processor conforms to the conformance level "classic" if it can handle modules conforming to the conformance level "classic" correctly. Given a module containing features beyond the conformance level "classic", the *RELAX Core* processor may stop normal processing, and, at user option, provide appropriate message.

The *RELAX Core* processor conforms to the conformance level "fully relaxed" if it handles any RELAX module correctly.

When a RELAX module has syntactical errors (i.e., it is not a document information item or does not meet conditions specified in this part of the Technical Report), further processing shall not occur. At user option, the *RELAX Core* processor shall report such syntactical errors.

Annex A

DTD for *RELAX Core*

A.1 The kernel of *RELAX Core*

```
<?xml version="1.0" encoding="utf-8"?>
<!--
DTD for RELAX Core (Ver 1.0)
-->
<!--*******************************************************-->
<!--                                                       -->
<!--         Parameter entities for qualified names        -->
<!--                                                       -->
<!--*******************************************************-->

<!ENTITY % corePrefix "">
<!ENTITY % interface "%corePrefix;interface">
<!ENTITY % export "%corePrefix;export">
<!ENTITY % div "%corePrefix;div">
<!ENTITY % tag "%corePrefix;tag">
<!ENTITY % elementRule "%corePrefix;elementRule">
<!ENTITY % module "%corePrefix;module">
<!ENTITY % include "%corePrefix;include">
<!ENTITY % attPool "%corePrefix;attPool">
<!ENTITY % hedgeRule "%corePrefix;hedgeRule">
<!ENTITY % ref "%corePrefix;ref">
<!ENTITY % hedgeRef "%corePrefix;hedgeRef">
<!ENTITY % choice "%corePrefix;choice">
<!ENTITY % sequence "%corePrefix;sequence">
<!ENTITY % element "%corePrefix;element">
<!ENTITY % none "%corePrefix;none">
<!ENTITY % empty "%corePrefix;empty">
<!ENTITY % anyOtherElement "%corePrefix;anyOtherElement">
<!ENTITY % anyOtherAttribute "%corePrefix;anyOtherAttribute">
<!ENTITY % mixed "%corePrefix;mixed">
<!ENTITY % attribute "%corePrefix;attribute">
<!ENTITY % annotationCore "%corePrefix;annotation">
<!ENTITY % appinfoCore "%corePrefix;appinfo">
<!ENTITY % documentationCore "%corePrefix;documentation">


<!--*******************************************************-->
<!--                                                       -->
<!--            References to datatypes.mod               -->
<!--                                                       -->
<!--*******************************************************-->

<!ENTITY % p "">
<!ENTITY % restriction1 "not_supported">
<!ENTITY % attrDecls "not_supported">
<!ENTITY % annotation "%annotationCore;">
<!ENTITY % datatype-definitions SYSTEM "datatypes.dtd">
%datatype-definitions;

<!--*******************************************************-->
<!--                                                       -->
<!--         The overall structure of RELAX modules.       -->
<!--                                                       -->
<!--*******************************************************-->

<!ELEMENT %interface; ((%annotation;)?, (%export; | %div;)*)>

<!ENTITY % clause "%tag;|%attPool;">

<!ENTITY % rule "%elementRule;|%hedgeRule;">
```

```
<!ELEMENT %module; ((%annotationCore;)?, (%interface;)?,
                       (%clause; | %rule; | %div; | %include; )*)>

<!ATTLIST %module;
                 moduleVersion     CDATA      #IMPLIED
                 relaxCoreVersion  CDATA      #REQUIRED
                 targetNamespace   CDATA      #IMPLIED
                 xmlns             CDATA      #FIXED
                                        "http://www.xml.gr.jp/xmlns/relaxCore"
>

<!--*******************************************************-->
<!--                                                       -->
<!--                          div                          -->
<!--                                                       -->
<!--*******************************************************-->

<!ELEMENT %div; ((%annotationCore;)?,
               (%div;)*,
               (((%rule; |%clause; | %include; ),
                 (%rule; |%clause; | %include; | %div;)*)
                |
                (%export;, (%export; | %div;)*))?)>

<!--

(%rule; |%clause; | %include; | %div;)* is used when a div appears in
a module body, while (%export; | %div;)* is used when it appears in an
interface element.

 -->

<!--*******************************************************-->
<!--                                                       -->
<!--                       Interface                       -->
<!--                                                       -->
<!--*******************************************************-->

<!ELEMENT %export; ((%annotationCore;)?)>
<!ATTLIST %export; label NMTOKEN #REQUIRED>

<!--*******************************************************-->
<!--                                                       -->
<!--                       Include                         -->
<!--                                                       -->
<!--*******************************************************-->

<!ELEMENT %include; ((%annotationCore;)?)>
<!ATTLIST %include; moduleLocation CDATA #REQUIRED>

<!--*******************************************************-->
<!--                                                       -->
<!--                    Hedge Models                       -->
<!--                                                       -->
<!--*******************************************************-->

<!-- The parameter entity "particle" is used to describe element hedge
models.  It is also used as subordinates of <sequence>, <choice>,
and <mixed>. -->

<!ENTITY % particle "%ref; | %hedgeRef; | %choice; | %sequence; | %element;
           | %none; | %empty;">


<!ENTITY % hedgeModel
  "(%particle;  | %mixed;)">

<!-- The parameter entity "repeatable" is used to specify the "occurs"
 attribute, which is shared by several elements.  Permissible values
 are either "?", "+", or "*".  -->
```

```
<!ENTITY % repeatable
                 "occurs        CDATA       #IMPLIED">


<!ELEMENT %hedgeRef; EMPTY >
<!ATTLIST %hedgeRef;
                 label      NMTOKEN      #REQUIRED
                 %repeatable;
>

<!ELEMENT %ref; EMPTY >
<!ATTLIST %ref;
                 label      NMTOKEN      #IMPLIED
                 role       NMTOKEN      #IMPLIED
                 %repeatable;
>

<!ELEMENT %empty; EMPTY >

<!ELEMENT %choice; (%particle;)* >
<!ATTLIST %choice;
                 %repeatable;
>

<!ELEMENT %sequence; (%particle;)* >
<!ATTLIST %sequence;
                 %repeatable;
>

<!ELEMENT %none; EMPTY>

<!ELEMENT %mixed; (%particle;) >

<!ELEMENT %element; ((%annotationCore;)?, (%facet;)*)>
<!ATTLIST %element;
                 name     NMTOKEN #REQUIRED
                 type     NMTOKEN #REQUIRED
                 %repeatable;
>


<!--*******************************************************-->
<!--                                                     -->
<!--                        Rules                        -->
<!--                                                     -->
<!--*******************************************************-->

<!ELEMENT %elementRule; ((%annotationCore;)?, (%tag;)?,
                         ((%hedgeModel;) | (%facet;)*))>

<!ATTLIST %elementRule;
     role        NMTOKEN #IMPLIED
     label       NMTOKEN #IMPLIED
                type        NMTOKEN #IMPLIED
>

<!ELEMENT %hedgeRule;  ((%annotationCore;)?, (%particle;)) >
<!ATTLIST %hedgeRule;
                label       NMTOKEN #REQUIRED

>

<!--*******************************************************-->
<!--                                                     -->
<!--                       Clauses                       -->
<!--                                                     -->
<!--*******************************************************-->

<!ENTITY % clauseBody "(%annotationCore;)?, (%ref; | %attribute;)*" >

<!ELEMENT %tag;  (%clauseBody;)>
<!ATTLIST %tag;
```

```
                 role      NMTOKEN      #IMPLIED
                 name      NMTOKEN      #IMPLIED
>

<!ELEMENT %attPool; (%clauseBody;)>
<!ATTLIST %attPool;
                 role   NMTOKEN      #REQUIRED
>

<!ELEMENT %attribute; ((%annotationCore;)?, (%facet;)*) >
<!ATTLIST %attribute;
                 name       NMTOKEN      #REQUIRED
                 required    (true)      #IMPLIED
                 type       NMTOKEN      #IMPLIED
>
<!--**********************************************************-->
<!--                                                          -->
<!--                        Annotations                       -->
<!--                                                          -->
<!--**********************************************************-->

<!ELEMENT %annotationCore; (%appinfoCore; | %documentationCore;)*>

<!ELEMENT %appinfoCore; ANY>   <!-- too restrictive -->
<!ATTLIST %appinfoCore;
          source      CDATA      #IMPLIED>

<!ELEMENT %documentationCore; ANY>   <!-- too restrictive -->
<!ATTLIST %documentationCore;
          source      CDATA    #IMPLIED>
```

A.2 Datatypes

RELAX Core uses “datatypes.dtd” in Appendix B of XML Schema part 2, which defines the syntax for referencing to datatypes and facets. The URL of this DTD is "http://www.w3.org/2001/datatypes.dtd".

Annex B

RELAX Module for *RELAX Core*

B.1 The kernel of *RELAX Core*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE module SYSTEM "relaxCore.dtd">
<!--
Module for RELAX Core (Ver 1.0)

-->

<module
  moduleVersion="1.0"
  relaxCoreVersion="1.0"
  targetNamespace="http://www.xml.gr.jp/xmlns/relaxCore"
  xmlns="http://www.xml.gr.jp/xmlns/relaxCore">

  <interface>
    <export label="module"/>
  </interface>

  <include moduleLocation="datatypes.rxm"/>

  <div>

    <annotation>
      <documentation>The overall structure of RELAX modules</documentation>
    </annotation>

    <elementRule role="module">
      <sequence>
        <ref label="annotation" occurs="?"/>
        <ref label="interface" occurs="?"/>
        <choice occurs="*">
          <hedgeRef label="clause"/>  <!-- forward references are fine -->
          <hedgeRef label="rule"/>    <!-- forward references are fine -->
          <ref label="divInModule"/>
          <ref label="include"/>
        </choice>
      </sequence>
    </elementRule>

    <tag name="module">
      <attribute name="moduleVersion" type="string"/>
      <attribute name="relaxCoreVersion" type="string" required="true">
        <enumeration value="1.0"/>
      </attribute>
      <attribute name="targetNamespace" type="anyURI"/>
    </tag>

    <elementRule role="interface">
      <sequence>
        <ref label="annotation" occurs="?"/>
        <choice occurs="*">
          <ref label="export"/>
          <ref label="divInInterface"/>
        </choice>
      </sequence>
    </elementRule>

    <tag name="interface"/>

    <hedgeRule label="clause">
      <choice>
```

```
        <ref label="tag"/>
        <ref label="attPool"/>
      </choice>
    </hedgeRule>

    <hedgeRule label="rule">
      <choice>
        <ref label="elementRule"/>
        <ref label="hedgeRule"/>
      </choice>
    </hedgeRule>


    <elementRule label="divInModule">
      <annotation>
        <documentation>div elements in modules</documentation>
      </annotation>
      <tag name="div"/>
      <sequence>
        <ref label="annotation" occurs="?"/>
        <choice occurs="*">
          <hedgeRef label="rule"/>
          <hedgeRef label="clause"/>
          <ref label="divInModule"/>
          <ref label="include"/>
        </choice>
      </sequence>
    </elementRule>
  </div>

  <div>
    <annotation>
      <documentation>Interface</documentation>
    </annotation>

    <elementRule role="export">
      <ref label="annotation" occurs="?"/>
    </elementRule>

    <tag name="export">
      <attribute name="label" required="true" type="NCName"/>
    </tag>

    <elementRule label="divInInterface">
      <annotation>
        <documentation>div elements in interfaces</documentation>
      </annotation>
      <tag name="div"/>
      <sequence>
        <ref label="annotation" occurs="?"/>
         <choice occurs="*">
          <ref label="export"/>
          <ref label="divInInterface"/>
        </choice>
      </sequence>
    </elementRule>

  </div>

  <div>
    <annotation>
      <documentation>Include</documentation>
    </annotation>

    <elementRule role="include">
      <ref label="annotation" occurs="?"/>
    </elementRule>

    <tag name="include">
      <attribute name="moduleLocation" type="anyURI" required="true"/>
    </tag>
  </div>
```

```
<div>
  <annotation>
    <documentation>Hedge Models</documentation>
  </annotation>

  <hedgeRule label="particle">
    <annotation>
      <documentation>This is used to describe element hedge models.
It is also used as subordinates of sequence,
choice, and mixed.
</documentation>
    </annotation>
    <choice>
      <ref label="refWithLabel"/>
      <ref label="hedgeRef"/>
      <ref label="choice"/>
      <ref label="sequence"/>
      <ref label="element"/>
      <ref label="none"/>
      <ref label="empty"/>
    </choice>
  </hedgeRule>

  <hedgeRule label="hedgeModel">
    <choice>
      <hedgeRef label="particle"/>
      <ref label="mixed"/>
    </choice>
  </hedgeRule>

  <attPool role="repeatable">
    <annotation>
      <documentation>This is used to specify the "occurs" attribute,
        which is shared by several elements.</documentation>
    </annotation>
    <attribute name="occurs" type="string">
      <enumeration value="?"/>
      <enumeration value="*"/>
      <enumeration value="+"/>
    </attribute>
  </attPool>

  <elementRule role="hedgeRef" type="emptyString"/>

  <tag name="hedgeRef">
    <ref role="repeatable"/>
    <attribute name="label" required="true" type="NCName"/>
  </tag>

  <elementRule label="refWithLabel" type="emptyString">
    <annotation>
      <documentation>ref elements with the label attribute</documentation>
    </annotation>
    <tag name="ref">
      <ref role="repeatable"/>
      <attribute name="label" required="true" type="NCName"/>
      <attribute name="role" type="none"/>
    </tag>
  </elementRule>

  <elementRule role="empty" type="emptyString"/>

  <tag name="empty"/>

  <elementRule role="choice">
    <hedgeRef label="particle" occurs="*"/>
  </elementRule>

  <tag name="choice">
    <ref role="repeatable"/>
  </tag>
```