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 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national standards bodies for voting. Publication as an International Standard requires approval of at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements in this document 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 19757-8 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

ISO/IEC 19757 consists of the following parts, under the general title *Document Schema Definition Languages (DSDL)*:

– *Part 1: Overview*

– *Part 2: Regular-grammar-based validation – RELAX NG*

– *Part 3: Rule-based validation – Schematron*

– *Part 4: Namespace-based validation dispatching language – NVDL*

– *Part 5: Datatypes*

– *Part 6: Path-based integrity constraints*

– *Part 7: Character repertoire description language – CRDL*

– *Part 8: Document schema renaming language – DSRL*

– *Part 9: Datatype- and namespace-aware DTDs*

– *Part 10: Validation management*

Introduction

This International Standard defines a set of Document Schema Definition Languages (DSDL) that can be used to specify one or more validation processes performed against Extensible Markup Language (XML) or Standard Generalized Markup Language (SGML) documents. (XML is an application profile SGML ISO 8879:1986.)

A document model is an expression of the constraints to be placed on the structure and content of documents to be validated with the model. A number of technologies have been developed through various formal and informal consortia since the development of Document Type Definitions (DTDs) as part of ISO 8879, notably by the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS). A number of validation technologies are standardized in DSDL to complement those already available as standards or from industry.

To validate that a structured document conforms to specified constraints in structure and content relieves the potentially many applications acting on the document from having to duplicate the task of confirming that such requirements have been met. Historically, such tasks and expressions have been developed and utilized in isolation, without consideration for how the features and functionality available in other technologies might enhance validation objectives.

The main objective of this International Standard is to bring together different validation-related tasks and expressions to form a single extensible framework that allows technologies to work in series or in parallel to produce a single or a set of validation results. The extensibility of DSDL accommodates validation technologies not yet designed or specified.

In the past, different design and use criteria have led users to choose different validation technologies for different portions of their information. Bringing together information within a single XML document sometimes prevents existing document models from being used to validate sections of data. By providing an integrated suite of constraint description languages that can be applied to different subsets of a single XML document, this International Standard allows different validation technologies to be integrated under a well-defined validation policy.

This multi-part International Standard integrates constraint description technologies into a suite that:

– provides user control of names, order and repeatability of information objects (elements)

– allows users to identify restrictions on the co-concurrence of elements and element contents

– allows specific subsets of structured documents to be validated

– allows restrictions to be placed on the contents of specific elements, including restrictions based on the content of other elements in the same document

– allows the character set that can be used within specific elements to be managed, based on the application of the ISO/IEC 10646 Universal Multiple-Octet Coded Character Set (UCS)

– allows default values to be assigned to element contents and attribute values, and provides facilities for the incorporation of predefined fragments of structured data to be incorporated within documents

– allows SGML to be used to declare document structure constraints that extend DTDs to include functions such as namespace-controlled validation and datatypes.

Document Schema Definition Languages (DSDL) – Part 8: Document Schema Renaming Language – DSRL

1 Scope

The Document Schema Renaming Language (DSRL) provides a mechanism whereby users can assign locally meaningful names to XML elements, attributes and entities without having to completely rewrite the DTD or schema to which they are required to conform. It also provides a mechanism for defining templates that can be used to define the structure and/or content of predefined parts of document streams.

NOTE 1: Templates created using DSRL are similar in purpose to abstract classes.

DSRL also allows default values to be assigned to specific parts of a data stream. This includes mechanisms for defining standard sequences of data that can be incorporated into document instances by reference to an identifying name and the provision of default content for elements and attributes for which no value is provided.

This Part provides a syntax for:

– using XPath to identify elements and attributes whose name or contents are to be modified

– renaming elements, attributes, entities and processing instructions in specified locations within the document model, including the assignment of element or attribute names to different namespaces

– assigning a default value to the contents of a specific type of element or attribute

– the definition of replacement contents for specific elements or attributes

– defining named fragments of previously defined data elements that can included within a document instance.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 19757. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 19757 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

Each of the following documents has a unique identifier that is used to cite the document in the text. The unique identifier consists of the part of the reference up to the first comma.

URI, IETF RFC 3986, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Standards Track Specification, August 1998, http://www.ietf.org/rfc/rfc3986.txt

XML, *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation, 4 February 2004, http://www.w3.org/TR/2004/REC-xml-20040204

XML-Infoset, *XML Information Set*, W3C Recommendation, 24 October 2001, http://www.w3.org/TR/2001/REC-xml-infoset-20011024/

XML-Names, *Namespaces in XML*, W3C Recommendation, 14 January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/

XPath, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116

XML Schema, *XML Schema*, W3C Recommendation, 24 October 2001, http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

XSLT, *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, http://www.w3.org/TR/1999/REC-xslt-19991116

XInclude, *XML Inclusions (1.0)*, W3C Recommendation, 30 September 2004, http://www.w3.org/TR/xinclude/

XForms, *XML Forms (1.0)*, W3C Recommendation, 14 October 2003, http://www.w3.org/TR/2003/REC-xforms-20031014/

3 Terms and definitions

3.1 document architecture

set of rules that are used to map a document instance to a document model defined by one or more schemas

4 The role of the Document Schema Renaming Language

The Document Schema Renaming Language (DSRL) provides a mechanism whereby users can assign locally meaningful names to XML elements, attributes and entities without having to completely rewrite the DTD or schema to which they are required to conform.

In addition to forming an XML-based mechanism that mimics the functionality of SGML and XML entities, DSRL can form the basis of abstract design patterns. An element in a document instance can, either by being assigned a fixed attribute value in a schema or by inclusion of an attribute in the instance, be mapped to a named element in a specified schema, so that it can be validated using the declarations in that schema. If subelements have been renamed within the identified element these can be mapped to new names either by use of attributes associated with individual elements or by the use of a "name map" associated with any of their parent elements.

DSRL can also be used to remap entity references to alternative names, and to reassign entity maps as parts of declared processes. For example, at certain stages in processing it may be important to retain entity references without expansion. In such instances the entity replacement mechanism can be replaced by one that automatically maps each entity reference to an entity reference, without breaking the rules about recursive entity references.

NOTE 2: This functionality mimics that of SDATA entity definitions in SGML.

DSRL also provides a mechanism for creating templates that can be used to define the structure and/or content of specific fragments of a document instance within a schema or DTD. If they are complete, such fragments can be incorporated into document instances using XInclude. If they are only partially complete, and require further input from users to form a section that can be validly included in the document, they can be presented to users as an XML form (e.g. XForms compliant data requests) to ensure the capture of missing data.

4.1 Namespace

Elements and attributes that conform to this Part shall have a namespace whose associated URI is:

```
    http://purl.oclc.org/dsdl/dsrl
```

In this Part the prefix `dsrl:` is used to identify points at which this URI defines the namespace.

5 Mapping user-defined names to schema-defined names

5.1 Reassigning element and attribute names

To map an element to a differently named element in a schema definition users of this part of the standard can:

– declare the mappings required within a schema's application-specfic information, or

– assign `dsrl:element-name-map` and `dsrl:attribute-names-map` attributes to an element in a document instance, or

– define mappings within an externally defined name map.

When used within the application-specific information of a schema a `dsrl:element-name-map` element identifies reusable mappings between names used in to identify elements in the schema and those used in document instances that can be validated against that schema. The `dsrl:element-name-map` element can be included as a foreign element in the documentation of any element defined in a schema. To rename an attribute a `dsrl:attribute-name-map` element can be included as a foreign element in the documentation of any attribute. The contents of each element consists of a list of nametokens that identify alternative names that can be assigned to the element or attribute. Names may be qualified providing the relevant namespaces have been declared within the schema.

When embedded within a RELAX NG schema the `dsrl:element-name-map` element will typically appear as:

```
<define name="contents">
 <element name="body">
  <a:documentation>The readable contents of a document</a:documentation>
  <dsrl:element-name-map>main-text</dsrl:name-map>
  <attribute name="cols">
   <dsrl:attribute-name-map>columns no-of-columns</dsrl:name-map>
   <data type="integer"/>
  </attribute>
  <ref name="foreword"/>
  <ref name="chapters"/>
  <ref name="annexes"/>
 </element>
</define>
```

NOTE 3: The `dsrl:name-map` declaration tells the schema that if it sees an element or attribute with one of the names listed in its contents the parser should validate it as if it had the name of the current element or attribute.

NOTE 4: For W3C schemas the `dsrl:name-map` element will be embedded within an `xsd:appInfo` element. For RELAX NG schemas any foreign elements (i.e. any elements with a different namespace) are automatically presumed to contain application-specific information.

A `dsrl:element-name-map` or `dsrl:attribute-name-map` attribute can be used to within a document instance to rename a specific element or its attributes. The value of the `dsrl:element-name-map` attribute is a single nametoken identifying the name assigned in the applicable schema to the element whose model this element instance is to be validated against. The value of the `dsrl:attribute-name-map` attribute is a set of pairs of names, the first of which is an attribute name that occurs in the instance and the second of which is the attribute name used in the applicable schema to define the values that can be applied to the attribute. A typical example of a local redefinition of element and attribute names might be:

```
<main-text number-of-columns="3" dsrl:element-name-map="body"
 dsrl:attribute-names-map="number-of-columns cols">
```

NOTE 5: These attributes can be defined as fixed attributes in the local document declaration so that they only need to be defined once in the document, not on each instance of the element.

Alternatively the `dsrl:element-name-map` and `dsrl:attribute-name-map` elements can be used in a separate declaration of name maps within a `dsrl:maps` container. Within `dsrl:maps` elements all DSRL elements must be assigned a `target` attribute whose value is a set of one or more XPath statements that identify the element(s) or attribute(s) to which this map is to be applied. The content of the `dsrl:name-map` element is the name of the element or attribute to be used to validate the content of the identified element.

Should we identify the targetNamespace and/or target schema/DTD as part of the map? Should the mapping be made on the outer dsrl:maps element or the element that is to become the root of the output? If the target schema/DTD is not identified how do we validate the result of the mapping? Does tying a map to a schema/DTD detract from its reusabilty across versions of schemas/DTDs?

When embedded within a `dsrl:maps` element these elements will appear as:

```
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 <dsrl:element-name-map target="main-text">body</dsrl:name-map>
 <dsrl:attribute-name-map target="body[@number-of-columns]">cols</dsrl:name-map>
 ...
</dsrl:maps>
```

To associate a name map with a schema a `dsrl:maps` attribute can be associated with the root element of the schema's formal definition. The content of this element is a list of one or more space separated URIs that identify sources of the maps. For example, if the above map was stored at `http://www.jtc1sc34.org/examples/dsrl-map-example.xml` it could be associated with a RELAX NG schema by extending the root tag of the schema to read:

```
<grammar dsrl:maps="http://www.jtc1sc34.org/examples/dsrl-map-example.xml">
```

The `dsrl:maps` attribute can also be used to identify the maps that can be applied to a specific document instance, or to part of a document instance, by assigning it to the first element from which the map is to apply. Typically this will be the root element, but other elements can also be assigned maps. The following example shows how different maps can be applied at different levels in a document instance:

```
<html dsrl:maps="http://us.org/maps/html-metadata-map.xml">
 <metadata property-name="working-title" property-value="Mapping HTML Metadata"/>
 ...
 <body dsrl:maps="http://us.org/maps/html-body-map.xml">
  <heading1>Mapping HTML Metadata</heading1>
  ....
 </body
</html>
```

More than one `dsrl:element-name-map` or `dsrl:attribute-name-map` can be associated with a particular element or attribute definition. Nametokens from multiple declarations are concatenated. Maps stored in an external resource that have been included using a schema's normal inclusion mechanism can provide the initial definitions for maps. Definitions provided within the document instance will override declarations made within the schema or any name maps associated with it.

NOTE 6: Most schema languages require inclusions to be defined at the start of the schema. Where this is not the case included maps are added to the list in the order they are included.

Discussion is required to ascertain whether the inclusion of a map within a schema is implementable. How would the inclusion of a DSRL map be distinguished from the inclusion of a schema fragment? How would the system process such a map?

Name maps are inherited by descendants. Where all attributes with a specific name map to the same name in the relevant schema, definitions placed on the parent element will suffice for the children. But where mapping of an attribute name depends on its parent then the mapping must be associated with the definition of the element affected. If a definition is specific to an element, but other inherited mappings need to be assigned to descendants of the element a `dsrl:inherit-map-of-parent` attribute in the same declaration or element instance must be assigned a value of `true`.

NOTE 7: The default value of the `dsrl:inherit-map-of-parent` attribute is `false`.

When name mapping has been applied the XML information set will contain the name required by the schema to validate each element or attribute. Optionally the system may record the original names of the element and its attributes in a processing instruction that immediately follows the element's start-tag. The `PItarget` of the processing instruction shall be `dsrl`. The original name of the element shall be recorded in an `original-element-name` property. The names used for each attribute shall be recorded in an `original-attribute-names` property as a pair of values, the first of which records the name in the transformed document and the second the name used in the source document. A typical record will have the form:

```
<?dsrl original-element-name="main-text"
       original-attribute-names="cols number-of-columns"?>
```

5.2 Mapping attribute value tokens

Where an attribute is defined requiring to be a member of a defined name token group, or to be a valid name, a mapping can be declared between names in a source document and names in a schema using one of the following methods:

– declare the mappings required within a schema's application-specfic information, or

– assign a `dsrl:map-attribute-values` attribute to an element in a document instance, or

– define mappings within an externally defined attribute value map.

When used within the application-specific information of a schema a `dsrl:attribute-values-map` element identifies reusable mappings between names used in the schema and those used in document instances that can be validated against that schema. The `dsrl:attribute-values-map` element can be included as a foreign element in the documentation of any attribute in the schema that has a list of permitted values.The contents of the element consists of a list of nametoken pairs that identify alternative values that can be used to validate attribute values. The first name of each pair of values must be a value that appears in the source file; the second name of each pair must be the value to be used when validating the entered value against the schema. The first name in each pair must be unique in the list of first names; the second name in each pair must be a name that is defined in in the associated schema but need not be unique as more than one alternative name may be permitted for each name used in the schema

When embedded within a RELAX NG schema the `dsrl:attribute-values-map` element will typically appear as:

```
...
<define name="contents">
 <element name="address">
  <a:documentation>Address for correspondence purposes</a:documentation>
   <dsrl:element-name-map>adresse</dsrl:element-name-map>
   <dsrl:element-name-map>dirección</dsrl:element-name-map>
  <attribute name="type">
   <dsrl:attribute-name-map>sorte</dsrl:attribute-name-map>
   <dsrl:attribute-name-map>tipo</dsrl:attribute-name-map>
   <choice>
     <value>home</value>
     <value>office</value>
   </choice>
   <dsrl:attribute-values-map>
     maison   home
     bureau   office
     casa     home
     oficina  office
   </dsrl:attribute-values-map>
  </attribute>
  <ref name="street"/>
  <ref name="locality"/>
  <ref name="city"/>
  <ref name="postcode"/>
  <ref name="country"/>
 </element>
</define>
...
```

Within document instances specific elements can have their attribute values mapped using the `dsrl:attribute-values-map` attribute. The value of the `dsrl:attribute-values-map` attribute is a set of pairs of names, the first of which is an attribute name that occurs in the instance and the second of which is the attribute name used in the applicable schema. A typical example of a local redefinition of an attribute name might be:

```
<address sorte="bureau" dsrl:attribute-names-map="sorte type"
 dsrl:attribute-values-map="bureau office maison home">
```

NOTE 8: Attribute value maps can be defined as fixed attributes in the local document declaration so that they only need to be defined once in the document, not on each instance of the element.

Alternatively the `dsrl:attribute-values-map` element can be used in a separate declaration within a `dsrl:maps` container. Within `dsrl:maps` elements all DSRL elements must be assigned a `target` attribute whose value is a set of one or more XPath statements that identify the element(s) or attribute(s) to which this map is to be applied. The content of the `dsrl:attribute-values-map` element is a list of nametoken pairs that identify alternative values that can be used to validate attribute values. The first name of each pair of values must be a value that appears in the source file; the second name of each pair must be the value to be used when validating the entered value against the schema. The first name in each pair must be unique in the list of first names; the second name in each pair must be a name that is defined in in the associated schema but need not be unique as more than one alternative name may be permitted for each name used in the schema

When embedded within a `dsrl:maps` element the `dsrl:attribute-values-map` element will appear as:

```
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 ...
 <dsrl:attribute-values-map target="adresse[@sorte]">
     maison   home
     bureau   office
 </dsrl:attribute-values-map>
 ...
</dsrl:maps>
```

NOTE 9: The form of the target address is expressed as `adresse[@sorte]` rather than `adresse/@sorte` because the templates generated by these declarations have to be qualified by the relevant value, e.g. `match="addresse[@sorte='maison']`.

Is this a valid reason for having a non-standard form for the target? Would it be just as simple to convert a `adresse/@sorte` input to `match="addresse[@sorte='maison']`?

NOTE 10: Within `dsrl:maps` any `dsrl:attribute-value-map` elements should immediately follow any `dsrl:attribute-names-map` element with the same value for the `target` attribute.

5.3 Mapping entity references

Neither Part 2 of this International Standard, the RELAX NG regular-grammar-based validation language, nor W3C XML schemas provide a mechanism for defining XML entities that can be referenced within document instances. Only XML DTDs can be used to specify general entities other than the five specified as default entities within the XML specification (`&amp;`, `&lt;`, `&ge;`, `&apos;` and `&quote;`).

NOTE 11: An alternative mechanism for defining the equivalent of general entities within W3C schemas or RELAX NG is provided within Clause 7 of this standard.

Often the names assigned to entity references, including the default ones defined for XML, are difficult for users to understand or remember, especially when they are specified using a language which is not the native language of a particular user community. The facilities in this clause allow locally-significant names to be mapped to those used to define entities in a referenced entity set.

To map an entity reference to a differently named entity in an entity definition users of this Part of the standard can:

– declare the mappings required within a schema's application-specfic information, or

– assign a `dsrl:map-entities` attribute to an element in a document instance, or

– define mappings within an externally defined name map.

A `dsrl:entity-name-map` element is used to identify reusable mappings between names used in entity definitions and those used in entity references. Such mappings may be defined using a foreign element in a schema that can be defined at any level in the model at which entities may be defined. The contents of the element consist of matched pairs of entity names where the first name is that assigned to the entity in its definition and the second is an alternative name for the entity used in the instance. The same entity name can be used as the first member of more than one entry. The second name in each pair must be unique within the list. More than one `dsrl:entity-name-map` element can occur in the same schema if different maps are required at different levels of the model (e.g. because different parts of the document are created by users from different language communities). Entries in such maps shall be concatenated. The same name shall not be associated with more than one entity definition in the concatenated list.

Typically an `dsrl:entity-name-map` will be applied to the root element of a schema, either using a local definition of the form:

```
<define name="root">
 <element name="book">
  <a:documentation>A book</a:documentation>
  <dsrl:entity-name-map>lt    less-that     gt   greater-than
                        lt    open-tag      gt   close-tag
                        amp   start-entity
                        apos  apostrophe    apos single-quote
                        quote double-quote</dsrl:entity-name-map>
  <attribute name="ISBN">
   <data type="ISBN"/>
  </attribute>
  <ref name="prelims"/>
  <ref name="body"/>
  <ref name="index"/>
 </element>
</define>
```

or by adding an entry to an externally defined set of maps, e.g.:

```
<dsrl:maps>
 ...
 <dsrl:entity-name-map target="book">lt less-that gt greater-than
  lt open-tag gt close-tag amp start-entity apos single-quote quote double-quote
 </dsrl:entity-name-map>
 ...
</dsrl:maps>
```

Jirka states "I don't think externally defined dsrl:entity-name-map should have a target attribute. Entity declarations are always valid for the whole document instance." I would argue, however, that mappings may need to be scoped, because within one element they may need to be mapped but in another they may not need to be mapped. This needs discussion in committee.

Mappings that are specific to a given instance can be specified using a `dsrl:map-entities` attribute on the outermost element the map is to apply to. The contents of the attribute consists of matched pairs of entity names where the first name is that assigned to the entity in its definition within the DTD and the second is an alternative name for the entity used in the instance. The same entity name can be used as the first member of more than one pair. The second name in each pair must be unique within the list. The following example shows how a different map can be applied at different levels:

```
<p dsrl:map-entities="lt less-than gt greater-than">
If x&less-than;0 then the expression must clearly marked up as being a negative,
giving it the form: <code dsrl:map-entities="lt open-tag gt close-tag">
&open-tag;expression negative="true"&close-tag;</code>.</p>
```

NOTE 12: It is assumed that such attributes will normally be defined as fixed attributes in the local document declaration so that they only need to be defined once in the document, not on each instance of the element.

NOTE 13: Multiple names can be assigned to any entity declared in the DTD associated with the document entity. Declarations that duplicate an existing map entry shall be ignored.

If both names in a matched pair of entity names are identical the entity reference must be replaced by itself. For example, to preserve the values of the default entities within a `code` element the following definitions would be applied:

```
<define name="Code">
 <element name="code">
  <dsrl:entity-name-map>lt lt gt gt amp amp apos apos quote quote</dsrl:entity-name-map>
  <text/>
 </element>
</define>
```

Entity maps are inherited by descendants. Where all entities of a specific map to the same name in the relevant document instance a map placed on the root element will suffice for all children. But where mapping of an entity name depends on its location in the document instance then the mapping must be associated with the definition of the element affected. If a mapping is specific to a particular element type, but other inherited mappings need to be assigned to descendants of the element, a `dsrl:inherit-entity-map-of-parent` attribute in the element instance containing the revised mapping definition must be assigned a value of `true`.

NOTE 14: The default value of the `dsrl:inherit-entity-map-of-parent` attribute is `false`.

In some instances entity mapping may need to to be disabled within a specific element within a document instance. When an element includes a `dsrl:disable-entity-mapping` attribute whose value is `true` entity reference replacement within the contents of the specificed element and any children that do not have a value of `false` for the same attribute are inhibited.

5.4 Renaming processing instruction targets

Where the names and properties of processing instructions have not been defined in terms understandable to user-communities, users of this Part of the standard can create a mapping rule that declares the alternative names to be used by adding an empty `dsrl:map-pi-target` element as a foreign element within an application-specific information declaration that is a child of the root node of a schema. The element must be assigned values for two compulsory attributes:

– `dsrl:target-name`, which contains the name to be used as the `PITarget` value for mapped processing instructions

– `dsrl:alternative-names`, which contains one or more nametokens that can be used as alternative names for the target name.

Alternatively an application can define mechanisms that allow users to define mappings, using a `dsrl:map-pi-target` element within an externally defined map, encoded as a `dsrl:maps` element, which can be invoked as a parameter during processing by reference to the map's filename.

NOTE 15: It is only possible to apply PI mappings to processing instructions that precede a root element if the mapping file is invoked before the first character of the input stream is read.

Ways in which maps can be invoked before processing of a schema need to be discussed further. There may be a requirement to be able to invoke the process using techniques described in Part 10 of this standard, but at present pipelining lanagues do not seem to generally offer such functionality.

Where properties of the target namespace have been defined using the commonly adopted convention of `property-name=property-value`, as is used to invoke XML properties such as character encoding, etc, in XML declarations, it also possible to assign locally significant names for properties using a `dsrl:property-names` attribute. The contents of this attribute is a set of matched pairs of nametokens, the first member of the pair being the name used for that property within the document instance and the second being the name that needs to be applied when the processing instructions are processed by the target validator.

This version changes the order of the two entries in a mapping pair. Is this order "natural"?

NOTE 16: Multiple assignments of alternative names to the same target property shall not be considered to be an error.

How could we indicate, using RNC or Schematron, that the `dsrl:property-names` attribute can occur in the PI itself? If we did this would it be possible to have a `dsrl:PI-name` attribute as well or would this be too late as the PI name has already been processed by the time the name map is encountered?

A typical application of processing instruction maps might have the form:

```
<dsrl:map-pi-targets dsrl:target-name="XML" dsrl:alternative-names="SGML OurML"
 dsrl:property-names="character-set encoding level version">
```

Should people be allowed to apply PI maps to PIs whose target is XML? Would it be acceptable to allow property-names to be mapped but discourage the use of alternatives to the XML target-name?

6 Assigning default values

6.1 Default element content

To assign a default value to an element if no content is provided in the instance users can:

– create a content declaration that defines the contents required within a schema's application-specific information, or

– assign a `dsrl:default-content` attribute to an element in a document instance, or

– define default element content within an externally defined map recorded in a `dsrl:maps` element.

A `dsrl:default-content` element is used to define a default content for an element defined within a schema or an externally defined map. Within a schema the element can be included as a foreign element in the documentation of the appropriate element in the schema. The contents of the `dsrl:default-content` element consists of the contents to be assigned to the element if no contents are provided. If the optional `dsrl:force-default` attribute is set to `true` the default content will automatically replace any content found in the doucment instance. A typical application might be:

```
<define name="RD-name">
 <element name="name">
  <a:documentation>Name of Researcher</a:documentation>
  <dsrl:default-content dsrl:force-default="true">Martin Bryan</dsrl:default-content>
  <text/>
 </element>
</define>
```

Default content that is specific to a given instance can be specified using a `dsrl:default-content` attribute on the element the content is to apply to. The contents of the attribute consists of the string to be assigned to the content.

NOTE 17: When declared using attributes the default contents may not include embedded elements. It is assumed that such attributes will normally be defined as fixed attributes in the local document declaration so that they only need to be defined once in the document, not on each instance of the element. A typical example would be:

```
myname = element name {attribute dsrl:default-content {"Martin Bryan"}, text}
```

Jirka claims that we need to add `[a:defaultValue="Martin Bryan"]` immediately after the first { to "define a fixed attribute". Jirka states this is needed for RELAX NG DTD Compatibility. Is this essential? Would users understand the reason for it if we added it without comment?

6.2 Default attribute values

To assign a defualt value to an attribute if no content or value is defined in the instance users can:

– create an attribute defaults declaration that defines the values required as part of a schema's application-specific information, or

– assign a `dsrl:default-attribute-values` attribute to an element in a document instance, or

– define default attribute values within an externally defined map recorded in a `dsrl:maps` element.

A `dsrl:default-attribute-values` element is used to define a default values for one or more attributes associated with an element defined in a schema or an externally defined map. Within a schema the element can be included as a foreign element in the documentation of the appropriate attribute. The contents of the element consist of space separated pairs of attribute names and values. If a default attribute needs to include spaces the value must be quoted. If the optional `force-defaults` attribute is set to `true` the default values will automatically replaced any values found in the doucment instance. A typical application might be:

```
<define name="RD-name">
 <element name="name">
  <a:documentation>Name of Researcher</a:documentation>
  <dsrl:default-attribute-values dsrl:force-default="true">
    type researcher team 'Research and Development'
  </dsrl:default-attribute-values>
  <text/>
 </element>
</define>
```

More than one `dsrl:default-attribute-values` element can be associated with a particular element or attribute definition. Default values from multiple declarations are concatenated. Where the same attribute is assigned more than one default value the last value to be assigned shall be used. Default values stored in an external resource that have been included using the schema's normal inclusion mechanism will provide the initial definitions for values.

NOTE 18: Most schema languages require inclusions to be defined at the start of the schema. Where this is not the case included maps are added to the list in the order they are included.

Default values that are specific to a given instance can be specified using a `dsrl:default-attribute-values` attribute on the element the content is to apply to. The contents of the attribute consist of space separated pairs of attribute names and values. If a default attribute needs to include spaces the value must be quoted. If the optional `force-defaults` attribute is set to `true` the default values will automatically replaced any values found in the doucment instance.

NOTE 19: It is assumed that such attributes will normally be defined as fixed attributes in the local document declaration so that they only need to be defined once in the document, not on each instance of the element.

Most schema languages already contain mechanisms for assigning default values to attributes. Whilst the approach suggested here allows `dsrl:maps` to contain all of the relevant default values, is this enough of an advantage to justify duplication of existing functionality?

7 Defining a document fragment template

A document fragment template defines a well-formed element, with or without predefined contents for embedded elements and attribute values. Document fragment templates can only be defined as direct children of schemas.[1]

Should it be possible to define fragments at element level rather than schema level?

[1] Document fragment templates cannot be defined as part of an element or attribute definition or within a rule or pattern defined by other parts of this International Standard. They can only be embedded within a schema's application-specific information.

The `dsrl:predefined-fragment` element may be used to identify document fragment templates within a schema or an externally defined map. This element can be defined using the following methods:

– as an element whose contents contain the required fragment, or

– as an empty element whose `href` attribute contains a URI that identifies where a copy of the template can be obtained from and, optionally, whose `xpointer` qualifier identifies which fragment of the referenced resource is to be included, or

– as an element within an externally defined map recorded in a `dsrl:maps` element.

The name of the fragment to be included is defined by the required `name` attribute. It must provide a unique identifier for the element within the schema or external map.

The namespace of the root element of a document fragment must be specifically declared using an attribute defined according to the specification for XML-Names. The optional `dsrl:schema-source` attribute can be used to enter a URI that identifies where a copy of a schema that can be used to validate the fragment can be obtained.

NOTE 20: Strictly speaking the namespace definition should be sufficient to identify the schema required. The `dsrl:schema-source` attribute can be used to provide a locally significant mapping of the namespace name to a specific copy of the validation schema.

A typical example of a reference to an externally stored document fragment would have the form:

```
<grammar xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 <dsrl:document-fragment dsrl:name="template-1"
  dsrl:schema-source="http://us.org/schemas/introductions.dtd"
  href="http://us.org/templates.xml#intro"/>
 ...
 <define name="intro" xmlns:xi="http://www.w3.org/2001/XInclude">
  <element name="intro">
   <a:documentation>Introduction (uses standardized text)</a:documentation>
   <text><xi:include href="#template-1"/></text>
  </element>
 </define>
</grammar>
```

When defined using the RELAX NG Compact Syntax this definition would take the form:

```
## Introduction (uses standardized text)
intro = element intro {[xi:include [href="#template-1"] ] text}
```

Is my use of braces to surround the spec supplied by Jirka correct? Is there any restriction on the use of spaces betweeen the two closing square brackets? Is the above valid? Where would the `xi` namespace need to be defined for use in the compact syntax?

Elements defining templates that require further input before the fragment can be embedded in a document instance must be flagged with a `dsrl:request-content` attribute whose value is `true`. Where entry of a field is optional the value assigned to the `dsrl:request-content` shall be `optional`. Where attribute values need to be specified before the fragment is complete the `dsrl:request-attributes` attribute may be added. The value of this attribute is a set of nametokens identifying the qualified name of all attributes whose value needs to be captured.

NOTE 21: A document fragment template must be a well-formed XML document that can be included in an XML document once missing content and attribute values have been defined.

NOTE 22: This standard does not specify how attribute value and element content should be supplied to complete the elements and attributes identified as requiring completion before inclusion, only that they do require completion. Missing values can be supplied by parameters passed to XSLT transformations, by use of XForms that request the relevant information, or any other mechanism deemed suitable by the application for capturing the required information.

Validators must report an error if one or more elements or attributues are still to be provided with values when the document fragment template is included into a document instance.

```
<grammar xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 <dsrl:document-fragment target="prelims" dsrl:name="title-page-form"
  dsrl:schema-source="http://us.org/schemas/books.xsd">
  <us:title-page>
   <us:title dsrl:request-content="true"/>
   <us:subtitle dsrl:request-content="optional"/>
   <us:author dsrl:request-content="true"
            dsrl:request-attributes="title initial nickname"/>
  </us:title-page>
 </dsrl:document-fragment>
 ...
 <define name="intro" xmlns:xi="http://www.w3.org/2001/XInclude"
                      xmlns:us="http://www.us.org/schemas">
  <element name="prelims">
   <a:documentation>Capture of data for title page</a:documentation>
   <text><xi:include href="#title-page-form"/></text>
  </element>
 </define>
</grammar>
```

How can we change this into a compact syntax definition?

Should we provide an attribute that allows users to specify the wording to be used for prompting for specific pieces of data? If so how would this work for attributes? Alternatively should we provide a pointer to a form that contains the relevant prompts?

Annex A
(normative)

Validation of declarative document architectures

The normative schemas defined in this annex provide formal definitions for the elements and attributes used to declare document architectures. The elements defined by these schemas will normally be used as foreign elements within schemas or as foreign attributes within document instances.

A.1 RELAX NG XML Schema for Validating DSRL

To be completed

A.2 RELAX NG Compact Schema for Validating DSRL

When names maps are stored externally the following RELAX NG compact syntax schema can be used to validate the map:

```
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"
namespace xsd  = "http://www.w3.org/2001/XMLSchema-datatypes"
## For consistency with the importable version of the schema all element names
## in this schema are qualified
start = dsrl:maps
## Allow maps to be imported from external sources
external-maps = attribute dsrl:maps {anyURI}
external-map-use |= element * {external-maps}
## Identify target element to which map is to be applied
target = attribute target {pathValue}
pathValue = string
## Identify inheritence rules to be applied to maps
inherit = attribute dsrl:inherit-map-of-parent {true | false}
inherit-entities = attribute dsrl:inherit-entity-map-of-parent {true | false}
disable-entities = attribute dsrl:disable-entity-mapping {true | false}
## Maps for converting instance-specific names into schema-specific names
maps = element dsrl:maps {(element-name-map |
                           (attribute-name-map, attribute-values-map?) |
                           entity-map | property-names | element-content)+}
element-name-map = element dsrl:element-name-map {target, inherit?, {xsd:Name+}}
attribute-name-map = element dsrl:attribute-name-map {target, inherit?, {xsd:Name+}}
attribute-value-map = element dsrl:attribute-values-map
                              {inherit?, {(xsd:Name, xsd:Name)+}}
entity-map = element dsrl:entity-name-map
             {target, (inherit-entities|disable-entities)?, {(xsd:Name, xsd:Name)+}
property-names = element dsrl:map-pi-target {empty,
                 attribute dsrl:target-name {xsd:Name},
                 attribute dsrl:alternative-names {xsd:Name+},
                 attribute dsrl:property-names {(xsd:Name, xsd:Name)+}?}
element-content = element dsrl:default-content {text, target?,
                  attribute dsrl:force-default {"true" | "false"}?}
attribute-values = element dsrl:default-attribute-values {value-pairs, target?,
                   attribute dsrl:force-default {"true" | "false"}?}
value-pairs = (attname, attvalue)+
attname = xsd:Name
attvalue = (quotedString|noSpaceString)
quotedString = (("'", text, "'")|('"', text, '"'))
```

```
noSpaceString = text - [ ]
template = element dsrl:predefined-fragment {any, target, attribute dsrl:name {ID},
                                             attribute href {anyURI}?,
                                             attribute drsl:schema-source {anyURI}?}
request-content |= element * {attribute dsrl:request-content ("true" | "optional")?,
                              attribute dsrl:request-attributes {xsd:NAMES}?}
```

Is the definition of noSpaceString valid? If not, how can we show that the string must not contain a space? Can \s be used within a character set definition?

The following RELAX NG compact syntax schame can be imported into schemas that contain DSRL name map components as annotations at points at which `foreign` elements are permitted:

Do foreign elements exclude elements defined in a different namespace within the schema?

Does xsd:QName allow you to have both namespace qualified and unqualified names? The RNC CName construct requires a prefix. What construct should RNC use to indicate that both qualified and unqualified names can occur at a particular point in a construct?)

```
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"
namespace xsd ="http://www.w3.org/2001/XMLSchema-datatypes"
external-maps = attribute dsrl:maps {anyURI}
external-map-use |= element * {external-maps}
inherit = attribute dsrl:inherit-map-of-parent {"true" | "false"}
inherit-entities = attribute dsrl:inherit-entity-map-of-parent {"true" | "false"}
disable-entities = attribute dsrl:disable-entity-mapping {"true" | "false"}
element-name-map = element dsrl:element-name-map {inherit?, xsd:QName+}
attribute-name-map = element dsrl:attribute-name-map {inherit?, xsd:QName+}
attribute-values-map = element dsrl:attribute-values-map {(xsd:Name, xsd:Name)+}
entity-map = element dsrl:entity-name-map
             {(inherit-entities|disable-entities)?, {(xsd:Name, xsd:Name)+}
property-names = element dsrl:map-pi-target {empty,
                  attribute dsrl:target-name {xsd:Name},
                  attribute dsrl:alternative-names {xsd:Name+},
                  attribute dsrl:property-names {(xsd:Name, xsd:Name)+}?}
element-content = element dsrl:default-content {text,
                   attribute dsrl:force-default {"true" | "false"}?}
attribute-values = element dsrl:default-attribute-values {value-pairs,
                     attribute dsrl:force-default {"true" | "false"}?}
value-pairs = (attname, attvalue)+
attname = xsd:Name
attvalue = (quotedString|noSpaceString)
quotedString = (("'", text, "'")|('"', text, '"'))
noSpaceString = text - [ ]
template = element dsrl:predefined-fragment {any, attribute dsrl:name {ID},
                                             attribute href {anyURI}?,
                                             attribute drsl:schema-source {anyURI}?}
request-content |= element * {attribute dsrl:request-content ("true" | "false")?,
                              attribute dsrl:request-attributes {xsd:NAMES}?}
```

The following RELAX NG compact syntax schame can be imported into schemas that will be used to validate DSRL attributes within document instances:

```
namespace dsrl = "http://purl.oclc.org/dsdl/dsrl"
namespace xsd  = "http://www.w3.org/2001/XMLSchema-datatypes"
local-maps |= element * {inherit?, element-name?, attribute-names?,
                        entity-map?, (inherit-entities|disable-entities)?}
disable-entities = attribute dsrl:disable-entity-mapping {true | false}
```

```
element-name = attribute dsrl:element-name-map {xsd:QName}
attribute-names = attribute dsrl:attribute-names-map {(xsd:QName, xsd:QName)+}
attribute-values-map = attribute dsrl:attribute-value-map {(xsd:Name, xsd:Name)+}
entity-map = attribute dsrl:map-entities {(xsd:Name, xsd:Name)+}
inherit = attribute dsrl:inherit-map-of-parent {"true" | "false"}
inherit-entities = attribute dsrl:inherit-entity-map-of-parent {"true" | "false"}
element-content = attribute dsrl:default-content {string}
attribute-values = attribute dsrl:default attribute values {(attname, attvalue)+}
attname = xsd:Name
attvalue = (quotedString|noSpaceString)
quotedString = (("'", text, "'")|('"', text, '"'))
noSpaceString = text - [ ]
## How can we show that attribute dsrl:property-names {(Name, Name)+}?} should be
## applied to processing instructions?
```

A.3 Schematron Rules for Validating DSRL

To be completed

Annex B
(informative)

Using DSRL and XSLT to Transform Schemas and Documents

B.1 Converting free-standing DSRL rules

DSRL rules that are defined in a separate mapping file can be converted into XSLT transformation rules that can be used to convert a document instance into a form that can be validated by the relevant validation schema by use of the following XSLT 2.0 transform:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xs="http://www.w3.org/1999/XSL/TransformAlias"
 xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xi="http://www.w3.org/2001/XInclude">
 <!-- Example of how XSLT 2.0 can be used to transform a DSRL map to create
      an XSL styleseet, called DSRLtranform.xsl, that can be used to transform
      XML instances into validatable documents.

      Also creates a file, DSRLentities.ent that redefines mapped entities in format
      required to allow transformation using DSRLtransform.xsl.

      © ISO SC34/WG1, 2006
 -->
 <xsl:output name="transforms" method="xml" indent="yes"/>
 <xsl:output name="entity-definitions" method="text"/>
 <xsl:output name="fragment-definitions" method="xml" indent="yes"/>

 <xsl:namespace-alias stylesheet-prefix="xs" result-prefix="xsl"/>

   <!-- Variable for storing information of local working directory:
        Needs to be customized for local environment -->
 <xsl:param name="output-directory">/DSDL/DSDL8%20Examples/</xsl:param>
 <xsl:param name="entity-redefinition">DSRLentities.ent</xsl:param>
 <xsl:param name="MappedEntities">MappedEntities.ent</xsl:param>
 <xsl:param name="fragment-store">DSRLfragments.ent</xsl:param>

 <xsl:template match="dsrl:maps">
  <xsl:result-document href="{$output-directory}DSRLtransform.xsl"
   format="transforms">
 <xsl:text disable-output-escaping="yes">
&#60;!DOCTYPE xsl:stylesheet [
&#60;!ENTITY % MappedEntities SYSTEM "</xsl:text>
<xsl:value-of select="$MappedEntities"/>
<xsl:text disable-output-escaping="yes">"&#62; %MappedEntities;
]&#62;
</xsl:text>
   <xs:stylesheet version="1.0" xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
    <xsl:namespace name="">
     <xsl:value-of select="*/@targetNamespace"/>
    </xsl:namespace>
    <xs:output method="xml" indent="yes"/>
    <xsl:apply-templates/>
    <xs:template match="*|@*">
     <xs:copy>
      <xs:apply-templates select="@*"/>
      <xs:apply-templates select="*|text()|comment()|processing-instruction()"/>
     </xs:copy>
    </xs:template>
   </xs:stylesheet>
  </xsl:result-document>
  <xsl:call-template name="fragments"/>
 </xsl:template>

 <xsl:template match="dsrl:element-name-map">
  <xsl:variable name="target"><xsl:value-of select="@target"/></xsl:variable>
  <xsl:variable name="new-name"><xsl:value-of select="."/></xsl:variable>
  <xsl:if test="not(//dsrl:default-content[@target=$target])">
  <xsl:if test="not(//dsrl:default-attribute-values[@target=$target])">
   <xs:template match="{@target}">
   <xs:element name="{.}">
    <xsl:if test="@targetSchemaLocation">
     <xs:attribute name="schemaLocation"
      namespace="http://www.w3.org/2001/XMLSchema-instance">
      <xsl:value-of select="./@targetNamespace"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="@targetSchemaLocation"/>
     </xs:attribute>
    </xsl:if>
    <xsl:if test="//dsrl:default-attribute-values[@target='{name()}']">
     <xsl:call-template name="default-attribute-values"/>
    </xsl:if>
    <xsl:if test="not(//dsrl:default-attribute-values[@target='{name()}'])">
     <xs:apply-templates select="@*"/>
    </xsl:if>
    <xs:apply-templates/>
   </xs:element>
  </xs:template>
  </xsl:if>
  </xsl:if>
 </xsl:template>

 <xsl:template match="dsrl:attribute-name-map">
  <xsl:variable name="element">
   <xsl:value-of select="preceding-sibling::dsrl:element-name-map[position()=1]/@target"/>
  </xsl:variable>
  <xsl:comment><xsl:value-of select="$element"/></xsl:comment>
  <xsl:if test="not(following-sibling::dsrl:attribute-values-map[position()=1])">
   <xs:template match="{@target}">
    <xs:element name="{$element}">
     <xs:attribute name="{text()}">
      <xs:value-of>
       <xsl:attribute name="select">
        <xsl:value-of select="@target"/>
        </xsl:attribute>
         </xs:value-of>
     </xs:attribute>
     <xsl:if test="parent::dsrl:element-name-map/@targetSchemaLocation!=''">
      <xs:attribute name="schemaLocation"
       namespace="http://www.w3.org/2001/XMLSchema-instance">
       <xsl:value-of select="parent::dsrl:element-name-map/@targetNamespace"/>
       <xsl:text> </xsl:text>
       <xsl:value-of select="parent::dsrl:element-name-map/@targetSchemaLocation"/>
      </xs:attribute>
     </xsl:if>
    </xs:element>
   </xs:template>
  </xsl:if>
 </xsl:template>

 <xsl:template match="dsrl:attribute-values-map">
  <xsl:variable name="target">
   <xsl:value-of select="@target"/>
  </xsl:variable>
  <xsl:variable name="element">
   <xsl:value-of select="preceding-sibling::dsrl:element-name-map[position()=1]/@target"/>
  </xsl:variable>
  <xsl:variable name="namespace">
   <xsl:value-of select="preceding-sibling::dsrl:element-name-map[position()=1]/
                         @targetNamespace"/>
  </xsl:variable>
  <xsl:variable name="schema">
   <xsl:value-of select="preceding-sibling::dsrl:element-name-map[position()=1]/
                         @targetSchemaLocation"/>
  </xsl:variable>
  <xsl:choose>
   <xsl:when test="preceding-sibling::dsrl:attribute-name-map[position()=1]">
    <xsl:variable name="new-name">
     <xsl:value-of select="preceding-sibling::dsrl:attribute-name-map[position()=1]"/>
    </xsl:variable>
    <xsl:analyze-string select="." regex="(\w+)\s+(\w+)\s*">
     <xsl:matching-substring>
      <xs:template match="{substring-before($target, ']')}='{regex-group(1)}']">
       <xs:element name="{$element}">
        <xs:attribute>
         <xsl:attribute name="name"><xsl:value-of select="$new-name"/></xsl:attribute>
         <xsl:value-of select="regex-group(2)"/>
        </xs:attribute>
        <xsl:if test="$schema!=''">
         <xs:attribute name="schemaLocation"
          namespace="http://www.w3.org/2001/XMLSchema-instance">
          <xsl:value-of select="$namespace"/>
          <xsl:text> </xsl:text>
          <xsl:value-of select="$schema"/>
         </xs:attribute>
        </xsl:if>
        <xs:apply-templates/>
       </xs:element>
      </xs:template>
     </xsl:matching-substring>
    </xsl:analyze-string>
   </xsl:when>
   <xsl:otherwise>
    <xsl:variable name="new-name">
     <xsl:value-of select="substring-before(substring-after(@target, '@') , ']')"/>
    </xsl:variable>
    <xs:template match="{$target}" priority="3">
     <xsl:analyze-string select="." regex="(\w+)\s+(\w+)\s*">
      <xsl:matching-substring>
       <xs:if>
        <xsl:attribute name="test">
         <xsl:value-of select="$target"/>=<xsl:value-of select="regex-group(1)"/>
        </xsl:attribute>
        <xs:attribute>
        <xsl:attribute name="name"><xsl:value-of select="$new-name"/></xsl:attribute>
         <xsl:value-of select="regex-group(2)"/>
        </xs:attribute>
       </xs:if>
      </xsl:matching-substring>
     </xsl:analyze-string>
     </xs:template>
    </xsl:otherwise>
   </xsl:choose>
  </xsl:template>

  <xsl:template match="dsrl:entity-name-map">
   <xs:template match="text()">
    <xs:call-template name="find-entity-ref">
     <xs:with-param name="t" select="."/>
    </xs:call-template>
   </xs:template>

   <xs:template name="find-entity-ref">
    <!-- Converts entity references to a validatable form.
         Presumes that all entities mapped to are defined in the relevant document type.
         Remember that when XML schemas are being used for validation only the five
         predefined entities, amp, lt, gt, quot and apos are defined; any other name
         specified for validation will generate an error.
    -->
    <xs:param name="t"/>
    <!--Try wrapping choose in entity definition-->
    <xsl:variable name="entities">
     <xsl:analyze-string select="." regex="([\-\w]+)\s+([\-\w]+)\s*">
      <xsl:matching-substring>
       <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
        <xsl:value-of select="regex-group(2)"/>
        "[[entity::<xsl:value-of select="regex-group(2)"/>]]"
       <xsl:text disable-output-escaping="yes">&#62;</xsl:text>
      </xsl:matching-substring>
     </xsl:analyze-string>
    </xsl:variable>
    <xsl:result-document href="{$output-directory}{$entity-redefinition}"
     format="entity-definitions">
    <!-- This file should be used to update the entity definitions of all
          mapped entities by adding it to the end of the input DOCTYPE defintion.
          Typically this will involve the addition of a parameter entity with the
          following generalized form:
          <!ENTITY % mapped-entities SYSTEM "DSRLentities.ent"> %mapped-entities;
       -->
     <xsl:value-of select="$entities"/>
    </xsl:result-document>
    <xsl:variable name="mapped-entities">
     <xsl:analyze-string select="." regex="([\-\w]+)\s+([\-\w]+)\s*">
      <xsl:matching-substring>
      <xsl:if test="not(regex-group(1)='amp') and not(regex-group(1)='lt') and
       not(regex-group(1)='gt') and not(regex-group(1)='apos') and
       not(regex-group(1)='quot')">
       <xsl:variable name="entity-definition1">
        <xsl:text disable-output-escaping="yes">&#38;</xsl:text>
        <xsl:value-of select="regex-group(2)"/><xsl:text>;</xsl:text></xsl:variable>
        <xsl:variable name="entity-definition2">[[Need definition for
         <xsl:value-of select="regex-group(1)"/> here]]</xsl:variable>
          <xsl:text disable-output-escaping="yes">&#60;</xsl:text>!ENTITY
           <xsl:value-of select="regex-group(1)"/> "
           <xsl:value-of select="$entity-definition2"/>"
           <xsl:text disable-output-escaping="yes">&#62;</xsl:text>
         </xsl:if>
      </xsl:matching-substring>
     </xsl:analyze-string>
    </xsl:variable>
    <xsl:result-document href="{$output-directory}{$MappedEntities}"
     format="entity-definitions">
     <!-- This file should be used to update the entity definitions of all mapped
          entities by adding it to the end of the input DOCTYPE defintion.
          Typically this will involve the addition of a parameter entity with the
          following generalized form:
          <!ENTITY % mapped-entities SYSTEM "MappedEntities.ent"> %mapped-entities;
     -->
      <xsl:value-of select="$mapped-entities"/>
    </xsl:result-document>
    <xs:choose>
     <xsl:analyze-string select="." regex="([\-\w]+)\s+([\-\w]+)\s*">
      <xsl:matching-substring>
       <xsl:variable name="entity-to-be-validated">
        <xsl:value-of select="regex-group(1)"/>
       </xsl:variable>
       <xsl:variable name="entity-name-entered">
        <xsl:value-of select="regex-group(2)"/>
       </xsl:variable>
       <xs:when test="contains($t, '[[entity::{$entity-name-entered}]]')">
        <xs:variable name="starts" select="substring-before($t,
         '[[entity::{$entity-name-entered}]]')"/>
        <xs:call-template name="find-entity-ref">
         <xs:with-param name="t" select="$starts"/>
        </xs:call-template>
        <xs:value-of>
         <xsl:text disable-output-escaping="yes">&#38;</xsl:text>
         <xsl:value-of select="$entity-to-be-validated"/>;</xs:value-of>
        <xs:variable name="ends" select="substring-after($t,
         '[[entity::{$entity-name-entered}]]')"/>
        <xs:call-template name="find-entity-ref">
         <xs:with-param name="t" select="$ends"/>
        </xs:call-template>
       </xs:when>
      </xsl:matching-substring>
     </xsl:analyze-string>
     <xs:otherwise>
      <xs:value-of select="$t"/>
     </xs:otherwise>
    </xs:choose>
   </xs:template>
  </xsl:template>

 <xsl:template match="dsrl:map-pi-target">
  <xsl:variable name="target"><xsl:value-of select="@target-name"/></xsl:variable>
  <xsl:analyze-string select="@alternative-names" regex="([\-\w]+)\s*">
   <xsl:matching-substring>
    <xs:template match="processing-instruction({regex-group(1)})">
     <xs:processing-instruction name="{$target}">
      <xs:value-of select="."/>
     </xs:processing-instruction>
    </xs:template>
   </xsl:matching-substring>
  </xsl:analyze-string>
```

```
</xsl:template>

<xsl:template match="dsrl:default-content">
 <xsl:param name="mapped-attributes">
  <xsl:value-of select="dsrl:default-attribute-values"/>
 </xsl:param>
 <xsl:param name="attribute-default-forced">
  <xsl:value-of select="dsrl:default-attribute-values/@force-default"/>
 </xsl:param>
 <xsl:variable name="element-name">
  <xsl:if test="@map-to-element-name">
   <xsl:value-of select="@map-to-element-name"/>
  </xsl:if>
  <xsl:if test="not(@map-to-element-name)"><xsl:value-of select="@target"/></xsl:if>
 </xsl:variable>
 <xsl:variable name="parent"><xsl:value-of select="@parent"/></xsl:variable>
  <xs:template match="{@parent}/{@target}">
  <xsl:if test="@force-default='true'">
   <xs:element name="{$element-name}">
   <xs:for-each select="@*">
   <xsl:call-template name="check-for-defaulted-attributes">
    <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
    <xsl:with-param name="attribute-default-forced"
     select="$attribute-default-forced"/>
   </xsl:call-template>
   </xs:for-each>
   <xsl:call-template name="add-missing-defaulted-attributes">
    <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
    <xsl:with-param name="attribute-default-forced"
     select="$attribute-default-forced"/>
   </xsl:call-template>
    <xs:value-of><xsl:value-of select="."/></xs:value-of>
   </xs:element>
  </xsl:if>
  <xsl:if test="not(@force-default='true')">
   <xs:element name="{$element-name}">
   <xs:for-each select="@*">
   <xsl:call-template name="check-for-defaulted-attributes">
    <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
    <xsl:with-param name="attribute-default-forced"
     select="$attribute-default-forced"/>
   </xsl:call-template>
   </xs:for-each>
   <xsl:call-template name="add-missing-defaulted-attributes">
    <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
    <xsl:with-param name="attribute-default-forced"
     select="$attribute-default-forced"/>
   </xsl:call-template>
   <xs:value-of select="."/>
   </xs:element>
   </xsl:if>
 </xs:template>
 <xsl:if test="not(@force-default='true')">
 <xsl:variable name="parent"><xsl:value-of select="@parent"/></xsl:variable>
   <xsl:variable name="new-element-name">
  <xsl:if test="//dsrl:element-name-map/@target=$parent">
   <xsl:value-of select="//dsrl:element-name-map[@target=$parent]"/>
  </xsl:if>
  <xsl:if test="not(//dsrl:element-name-map/@target=$parent)">
   <xsl:value-of select="@parent"/>
  </xsl:if>
  </xsl:variable>
  <xs:template match="{@parent}" priority="1">
  <xs:element name="{$new-element-name}">
   <xsl:if test="//dsrl:element-name-map[@target=$parent]/@targetNamespace">
   <xs:attribute name="targetNamespace">
    <xsl:value-of select="//dsrl:element-name-map[@target=$parent]/@targetNamespace"/>
   </xs:attribute>
   </xsl:if>
   <xsl:if test="//dsrl:element-name-map[@target=$parent]/@targetSchemaLocation">
   <xs:attribute name="targetSchemaLocation">
   <xsl:value-of select="//dsrl:element-name-map[@target=$parent]/
    @targetSchemaLocation"/>
   </xs:attribute>
  </xsl:if>
   <xsl:for-each select="//dsrl:attribute-values-map[@target[starts-with(.,$parent)]]">
   <xsl:variable name="attribute-names">
     <xsl:for-each select="//dsrl:attribute-values-map[@target[starts-with(.,$parent)]]">
       <xsl:value-of select="substring-before(substring-after(@target, '@'), ']')"/>
       <xsl:text> </xsl:text>
     </xsl:for-each>
   </xsl:variable>
   <xsl:variable name="old-name"><xsl:value-of select="@target"/></xsl:variable>
   <xsl:variable name="new-name" select="//dsrl:attribute-name-map[@target=$old-name]"/>
   <xsl:variable name="map"><xsl:value-of select="."/></xsl:variable>
   <xsl:variable name="attribute-name">
    <xsl:value-of select="substring-before(substring-after(@target, '['), ']')"/>
   </xsl:variable>
   <xsl:analyze-string select="$map" regex="(\w+)\s+(\w+)\s*">
    <xsl:matching-substring>
     <xs:if test="{$attribute-name}='{regex-group(1)}'">
      <xs:attribute name="{$new-name}">
       <xsl:value-of select="regex-group(2)"/>
      </xs:attribute>
     </xs:if>
    </xsl:matching-substring>
   </xsl:analyze-string>
   <xs:if test="{$attribute-name}">
    <xs:attribute name="{$new-name}">
     <xs:value-of select="{$attribute-name}"/>
    </xs:attribute>
   </xs:if>
   <xs:for-each select="@*">
    <xs:if test="not(contains('{$attribute-names}', name()))">
     <xs:copy/>
    </xs:if>
   </xs:for-each>
  </xsl:for-each>
  <xsl:if test="not(//dsrl:attribute-values-map/@target[starts-with(.,$parent)])">
   <xs:value-of select="{//dsrl:attribute-name-map/@target[starts-with(.,$parent)]
    [substring-after(.,'@')]}"/>
   </xsl:if>
   <xsl:if test="not(//dsrl:attribute-name-map[starts-with(@target,$parent)])">
   <xs:copy select="@*"/>
   </xsl:if>
 <xs:apply-templates select="*|comment()|processing-instruction()"/>
  <xs:if test="not({@target})">
  <xs:element name="{$element-name}">
  <xs:for-each select="@*">
  <xsl:call-template name="check-for-defaulted-attributes">
```

```
   <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
   <xsl:with-param name="attribute-default-forced"
    select="$attribute-default-forced"/>
  </xsl:call-template>
  </xs:for-each>
   <xsl:call-template name="add-missing-defaulted-attributes">
    <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
    <xsl:with-param name="attribute-default-forced"
     select="$attribute-default-forced"/>
   </xsl:call-template>
   <xsl:variable name="text"><xsl:value-of select="text()"/></xsl:variable>
   <xs:value-of><xsl:value-of select="normalize-space($text)"/></xs:value-of>
  </xs:element>
  </xs:if>
  </xs:element>
 </xs:template>
 </xsl:if>
</xsl:template>

<xsl:template name="default-attribute-values" match="dsrl:default-attribute-values">
 <xsl:param name="mapped-attributes">
  <xsl:value-of select="."/>
 </xsl:param>
 <xsl:param name="attribute-default-forced">
  <xsl:value-of select="@force-default"/>
 </xsl:param>
 <xsl:param name="target"><xsl:value-of select="@target"/></xsl:param>
 <xs:template match="{@target}">
 <xs:element>
  <xsl:attribute name="name">
   <xsl:choose>
   <xsl:when test="preceding-sibling::dsrl:element-name-map[@target=$target]">
   <xsl:value-of select="preceding-sibling::dsrl:element-name-map[@target=$target]"/>
   </xsl:when>
   <xsl:when test="following-sibling::dsrl:element-name-map[@target=$target]">
   <xsl:value-of select="following-sibling::dsrl:element-name-map[@target=$target]"/>
   </xsl:when>
   <xsl:otherwise>
   <xsl:value-of select="$target"/></xsl:otherwise>
  </xsl:choose>
  </xsl:attribute>
  <xs:for-each select="@*">
  <xsl:call-template name="check-for-defaulted-attributes">
  <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
  <xsl:with-param name="attribute-default-forced" select="$attribute-default-forced"/>
  </xsl:call-template>
  </xs:for-each>
  <xsl:call-template name="add-missing-defaulted-attributes">
  <xsl:with-param name="mapped-attributes" select="$mapped-attributes"/>
  <xsl:with-param name="attribute-default-forced" select="$attribute-default-forced"/>
   </xsl:call-template>
  <xs:apply-templates/>
  </xs:element>
 </xs:template>
</xsl:template>

 <xsl:template name="check-for-defaulted-attributes">
  <xsl:param name="mapped-attributes"/>
  <xsl:param name="attribute-default-forced"/>
  <xsl:choose>
  <xsl:when test="$attribute-default-forced='true'">
   <xs:choose>
   <xsl:analyze-string select="$mapped-attributes" regex="([\-\w]+)\s+([\-\w]+)\s*">
   <xsl:matching-substring>
    <xs:when test="name()='{regex-group(1)}'">
    <xs:attribute name="{regex-group(1)}">
     <xsl:value-of select="regex-group(2)"/>
    </xs:attribute>
    </xs:when>
   </xsl:matching-substring>
   </xsl:analyze-string>
   <xs:otherwise>
   <xs:attribute name="{{name()}}"><xs:value-of select="."/></xs:attribute>
   </xs:otherwise>
   </xs:choose>
  </xsl:when>
  <xsl:when test="not($attribute-default-forced='true')">
   <xs:apply-templates/>
  </xsl:when>
  </xsl:choose>
 </xsl:template>

<xsl:template name="add-missing-defaulted-attributes">
  <xsl:param name="mapped-attributes"/>
  <xsl:param name="attribute-default-forced"/>
  <xsl:if test="$attribute-default-forced='true'">
   <xsl:analyze-string select="$mapped-attributes" regex="([\-\w]+)\s+([\-\w]+)\s*">
   <xsl:matching-substring>
    <xs:if test="not(@*[name()='{regex-group(1)}'])">
    <xs:attribute name="{regex-group(1)}">
     <xsl:value-of select="regex-group(2)"/>
    </xs:attribute>
    </xs:if>
   </xsl:matching-substring>
  </xsl:analyze-string>
  </xsl:if>
 </xsl:template>

 <!-- For off-line processing gragments are stored in a separate file, whose
      name defautls to DSRLfragments.xml, but this can be changed using a
      system supplied parameter -->
 <xsl:template name="fragments">
  <xsl:param name="fragment-store">DSRLfragments.xml</xsl:param>
  <xsl:param name="fragment-name"><xsl:value-of select="@name"/></xsl:param>
  <xsl:param name="fragment-schema">
    <xsl:value-of select="@schema-source"/>
  </xsl:param>
  <xsl:result-document href="{$output-directory}{$fragment-store}"
   format="fragment-definitions">
  <xsl:element name="dsrl:fragments">
  <xsl:for-each select="dsrl:document-fragment">
   <xsl:element name="dsrl:fragment">
    <xsl:attribute name="id" select="@name"/>
    <xsl:copy-of select="child::*"/>
   </xsl:element>
  </xsl:for-each>
  </xsl:element>
 </xsl:result-document>
 </xsl:template>
 <!-- It is presumed that fragments will be accessed using XInclude statements
      that reference the entry stored in the system's fragment store.-->
 <xsl:template match="dsrl:document-fragment">
  <xs:template match="xi:include">
   <xs:variable name="file">
    <xs:value-of select="substring-before(@href, '#')"/>
   </xs:variable>
   <xs:variable name="id">
    <xs:value-of select="substring-after(@href, '#')"/>
   </xs:variable>
   <xs:apply-templates select="document($file)//dsrl:fragment[@id=$id]" />
  </xs:template>
  <xs:template match="dsrl:fragment"><xs:apply-templates/></xs:template>
 </xsl:template>

</xsl:stylesheet>
```

NOTE 23: For off-line working fragments are not able to include `dsrl:request-content` elements.

To demonstrate how this transformation can be applied, we will show how a file consisting of a number of French addresses, marked up using French element and attribute names, and referencing entities defined with French names, can be mapped to a schema for European addresses which uses English for its markup names. The example document to be transformed has the following format:

```
<doc xmlns:xi="http://www.w3.org/2001/XInclude">
 <adresse sorte="maison">
  <numero>29</numero>
  <rue>Rue Bricot</rue>
  <ville>Monmartre</ville>
  <?PInameAsInput Embedded PI?>
  <cité>Paris</cité>
  <département>Île de France</département>
  <code-postal>95010</code-postal>
  <pays>France &open-tag;this&and;that&close-tag;</pays>
 </adresse>
 <adresse sorte="bureau">
  <numero>2</numero>
  <rue>Avenue Charles de Gaulle</rue>
  <?MyPI Another mapped PI?>
  <cité>Toulon</cité>
  <département>Aud&e; &et; Dauphine</département>
  <code-postal>12345</code-postal>
  <pays>France</pays>
 </adresse>
 <xi:include href="DSRLfragments.xml#title-page-form"/>
</doc>
```

NOTE 24: The final `xi:include` element is somewhat illogical at this point, but is included so that you can see how fragments could be incorporated into a file.

a simplified DTD that could be used to validate this document instance might have the form:

```
<!ELEMENT doc (adresse+, xi:include*)>
<!ATTLIST doc xmlns:xi CDATA "http://www.w3.org/2001/XInclude" >
<!ELEMENT adresse (numero, rue, ville?, cité, département, code-postal, pays?)>
<!ATTLIST adresse sorte CDATA #REQUIRED >
<!ELEMENT cité (#PCDATA)>
<!ELEMENT code-postal (#PCDATA)>
<!ELEMENT département (#PCDATA)>
<!ELEMENT numero (#PCDATA)>
```

```
<!ELEMENT pays (#PCDATA)>
<!ELEMENT rue (#PCDATA)>
<!ELEMENT ville (#PCDATA)>
<!ELEMENT xi:include ANY>
<!ATTLIST xi:include href CDATA #REQUIRED>
<!ENTITY % DSRLentities SYSTEM "DSRLentities.ent">
%DSRLentities;
```

The document will be validated against the following W3C XML Schema:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns="http://csw.co.uk/addresses" targetNamespace="http://csw.co.uk/addresses"
 elementFormDefault="qualified">
  <xs:element name="address">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="building-identifier"/>
        <xs:element ref="road"/>
        <xs:element ref="locality" minOccurs="0"/>
        <xs:element ref="postal-town"/>
        <xs:element ref="county" minOccurs="0"/>
        <xs:element ref="postcode"/>
        <xs:element ref="country" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="home"/>
            <xs:enumeration value="office"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="building-identifier">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="country">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="county">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="locality">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="postal-town">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="postcode">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="road">
    <xs:simpleType>
      <xs:restriction base="xs:string"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="address" maxOccurs="unbounded"/>
        <xs:any namespace="xi"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The map used to transform an instance marked up using the DTD has the form:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="TransformDSRLmaps.xsl"?>
<dsrl:maps xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 <!--Mapping of element and attribute names and attribute values-->
 <dsrl:element-name-map target="adresse" targetNamespace="http://csw.co.uk/addresses"
  targetSchemaLocation="EuropeanAddress.xsd">address</dsrl:element-name-map>
 <dsrl:attribute-name-map target="adresse[@sorte]">type</dsrl:attribute-name-map>
 <dsrl:attribute-values-map target="adresse[@sorte]">
   maison home
   bureau office
  </dsrl:attribute-values-map>
 <dsrl:attribute-name-map target="adresse[@torte]">type2</dsrl:attribute-name-map>
 <dsrl:attribute-values-map target="adresse[@torte]">
   maison home
   bureau office
  </dsrl:attribute-values-map>

 <dsrl:element-name-map target="numero">building-identifier</dsrl:element-name-map>
 <dsrl:element-name-map target="rue">road</dsrl:element-name-map>
 <dsrl:element-name-map target="ville">locality</dsrl:element-name-map>
 <dsrl:element-name-map target="cité">postal-town</dsrl:element-name-map>
 <dsrl:element-name-map target="département">county</dsrl:element-name-map>
 <dsrl:element-name-map target="code-postal">postcode</dsrl:element-name-map>
 <dsrl:element-name-map target="pays">country</dsrl:element-name-map>
 <!--dsrl:attribute-name-map target="direccione[@tipo]">type2</dsrl:attribute-name-map>
 <dsrl:attribute-values-map target="direccione[@tipo]">
   maison   home
   bureau   office
  </dsrl:attribute-values-map-->
<!--Assigning default values-->
<dsrl:entity-name-map>
eacute e
amp     et
amp     and
lt          open-tag
gt        close-tag
</dsrl:entity-name-map>
<dsrl:map-pi-target target-name="PIname" alternative-names="PInameAsInput
```

```
 AlternativePIname"/>

<dsrl:map-pi-target target-name="ProcessThis" alternative-names="MyPI"/>

<dsrl:default-content target="cité" parent="adresse" map-to-element-name="postal-town"
 force-default="true">Bordeaux</dsrl:default-content>

<dsrl:default-content target="ville" parent="adresse" map-to-element-name="locality">
<dsrl:default-attribute-values force-default="false">required false
</dsrl:default-attribute-values>
Downtown
</dsrl:default-content>

<dsrl:default-attribute-values target="pays" force-default="true">
 code-system iso3166</dsrl:default-attribute-values>

 <dsrl:document-fragment name="title-page-form"  target="prelims"
   schema-source="http://us.org/schemas/books.xsd" xmlns:us="http://we.are.us">
   <us:title-page>
    <us:title dsrl:request-content="true"/>
    <us:subtitle dsrl:request-content="optional"/>
    <us:author dsrl:request-content="true"
               dsrl:request-attributes="title initial nickname"/>
   </us:title-page>
   <us:publisher>Get A Life Publishing (Barbados) plc</us:publisher>
  </dsrl:document-fragment>
</dsrl:maps>
```

NOTE 25: This map contains some conversions, such as those for processing instructions, whose only real purpose is to illustrate the application of each of the features of DSRL. In practice maps will notnormally include all of the DSRL constructs, as this test map does.

When transformed using the XSLT 2.0 transformation shown at the start of this clause, the following tranformation map is produced:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY % MappedEntities SYSTEM "MappedEntities.ent"> %MappedEntities;
]>
<xsl:stylesheet xmlns:xi="http://www.w3.org/2001/XInclude"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl"
 xmlns="http://csw.co.uk/addresses" version="1.0">
  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="adresse">
   <xsl:element name="address">
     <xsl:attribute name="schemaLocation"
      namespace="http://www.w3.org/2001/XMLSchema-instance">
      http://csw.co.uk/addresses EuropeanAddress.xsd</xsl:attribute>
     <xsl:apply-templates select="@*"/>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
 <!--adresse-->
  <xsl:template match="adresse[@sorte='maison']">
   <xsl:element name="adresse">
     <xsl:attribute name="type">home</xsl:attribute>
     <xsl:attribute name="schemaLocation"
```

```
      namespace="http://www.w3.org/2001/XMLSchema-instance">
      http://csw.co.uk/addresses EuropeanAddress.xsd</xsl:attribute>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
  <xsl:template match="adresse[@sorte='bureau']">
   <xsl:element name="adresse">
     <xsl:attribute name="type">office</xsl:attribute>
     <xsl:attribute name="schemaLocation"
       namespace="http://www.w3.org/2001/XMLSchema-instance">
       http://csw.co.uk/addresses EuropeanAddress.xsd</xsl:attribute>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
 <!--adresse-->
  <xsl:template match="adresse[@torte='maison']">
   <xsl:element name="adresse">
     <xsl:attribute name="type2">home</xsl:attribute>
     <xsl:attribute name="schemaLocation"
      namespace="http://www.w3.org/2001/XMLSchema-instance">
      http://csw.co.uk/addresses EuropeanAddress.xsd</xsl:attribute>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
  <xsl:template match="adresse[@torte='bureau']">
   <xsl:element name="adresse">
     <xsl:attribute name="type2">office</xsl:attribute>
     <xsl:attribute name="schemaLocation"
      namespace="http://www.w3.org/2001/XMLSchema-instance">
      http://csw.co.uk/addresses EuropeanAddress.xsd</xsl:attribute>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>

  <xsl:template match="numero">
   <xsl:element name="building-identifier">
     <xsl:apply-templates select="@*"/>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
  <xsl:template match="rue">
   <xsl:element name="road">
     <xsl:apply-templates select="@*"/>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>


  <xsl:template match="département">
   <xsl:element name="county">
     <xsl:apply-templates select="@*"/>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>
  <xsl:template match="code-postal">
   <xsl:element name="postcode">
     <xsl:apply-templates select="@*"/>
     <xsl:apply-templates/>
     </xsl:element>
   </xsl:template>
```

```
<xsl:template match="text()">
  <xsl:call-template name="find-entity-ref">
    <xsl:with-param name="t" select="."/>
  </xsl:call-template>
 </xsl:template>
 <xsl:template name="find-entity-ref">
  <xsl:param name="t"/>
  <xsl:choose>
    <xsl:when test="contains($t, '[[entity::e]]')">
     <xsl:variable name="starts" select="substring-before($t, '[[entity::e]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$starts"/>
     </xsl:call-template>
     <xsl:value-of>&eacute;</xsl:value-of>
     <xsl:variable name="ends" select="substring-after($t, '[[entity::e]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$ends"/>
     </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($t, '[[entity::et]]')">
     <xsl:variable name="starts" select="substring-before($t, '[[entity::et]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$starts"/>
     </xsl:call-template>
    <xsl:value-of>&amp;</xsl:value-of>
     <xsl:variable name="ends" select="substring-after($t, '[[entity::et]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$ends"/>
     </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($t, '[[entity::and]]')">
     <xsl:variable name="starts" select="substring-before($t, '[[entity::and]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$starts"/>
     </xsl:call-template>
     <xsl:value-of>&amp;</xsl:value-of>
     <xsl:variable name="ends" select="substring-after($t, '[[entity::and]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$ends"/>
     </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($t, '[[entity::open-tag]]')">
     <xsl:variable name="starts" select="substring-before($t, '[[entity::open-tag]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$starts"/>
     </xsl:call-template>
     <xsl:value-of>&lt;</xsl:value-of>
     <xsl:variable name="ends" select="substring-after($t, '[[entity::open-tag]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$ends"/>
     </xsl:call-template>
    </xsl:when>
    <xsl:when test="contains($t, '[[entity::close-tag]]')">
     <xsl:variable name="starts" select="substring-before($t, '[[entity::close-tag]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$starts"/>
     </xsl:call-template>
     <xsl:value-of>&gt;</xsl:value-of>
     <xsl:variable name="ends" select="substring-after($t, '[[entity::close-tag]]')"/>
     <xsl:call-template name="find-entity-ref">
       <xsl:with-param name="t" select="$ends"/>
     </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
     <xsl:value-of select="$t"/>
    </xsl:otherwise>
  </xsl:choose>
 </xsl:template>
 <xsl:template match="processing-instruction(PInameAsInput)">
  <xsl:processing-instruction name="PIname">
    <xsl:value-of select="."/>
  </xsl:processing-instruction>
 </xsl:template>
 <xsl:template match="processing-instruction(AlternativePIname)">
  <xsl:processing-instruction name="PIname">
    <xsl:value-of select="."/>
  </xsl:processing-instruction>
 </xsl:template>

 <xsl:template match="processing-instruction(MyPI)">
  <xsl:processing-instruction name="ProcessThis">
    <xsl:value-of select="."/>
  </xsl:processing-instruction>
 </xsl:template>

 <xsl:template match="adresse/cité">
  <xsl:element name="postal-town">
    <xsl:for-each select="@*">
     <xsl:apply-templates/>
    </xsl:for-each>
    <xsl:value-of>Bordeaux</xsl:value-of>
  </xsl:element>
 </xsl:template>

 <xsl:template match="adresse/ville">
  <xsl:element name="locality">
    <xsl:for-each select="@*">
     <xsl:apply-templates/>
    </xsl:for-each>
    <xsl:value-of select="."/>
  </xsl:element>
 </xsl:template>
 <xsl:template match="adresse" priority="1">
  <xsl:element name="address">
    <xsl:attribute name="targetNamespace">http://csw.co.uk/addresses</xsl:attribute>
    <xsl:attribute name="targetSchemaLocation">EuropeanAddress.xsd</xsl:attribute>
    <xsl:if test="@sorte='maison'">
     <xsl:attribute name="type">home</xsl:attribute>
    </xsl:if>
    <xsl:if test="@sorte='bureau'">
     <xsl:attribute name="type">office</xsl:attribute>
    </xsl:if>
    <xsl:if test="@sorte">
     <xsl:attribute name="type">
       <xsl:value-of select="@sorte"/>
     </xsl:attribute>
    </xsl:if>
    <xsl:for-each select="@*">
     <xsl:if test="not(contains('sorte torte ', name()))">
```

```
        <xsl:copy/>
      </xsl:if>
     </xsl:for-each>
     <xsl:if test="@torte='maison'">
      <xsl:attribute name="type2">home</xsl:attribute>
     </xsl:if>
     <xsl:if test="@torte='bureau'">
      <xsl:attribute name="type2">office</xsl:attribute>
     </xsl:if>
     <xsl:if test="@torte">
      <xsl:attribute name="type2">
        <xsl:value-of select="@torte"/>
      </xsl:attribute>
     </xsl:if>
     <xsl:for-each select="@*">
     <xsl:if test="not(contains('sorte torte ', name()))">
    <xsl:copy/>
      </xsl:if>
     </xsl:for-each>
     <xsl:apply-templates select="*|comment()|processing-instruction()"/>
     <xsl:if test="not(ville)">
      <xsl:element name="locality">
       <xsl:for-each select="@*">
        <xsl:apply-templates/>
        </xsl:for-each>
        <xsl:value-of>Downtown</xsl:value-of>
      </xsl:element>
     </xsl:if>
   </xsl:element>
  </xsl:template>

  <xsl:template match="pays">
   <xsl:element name="country">
     <xsl:for-each select="@*">
      <xsl:choose>
        <xsl:when test="name()='code-system'">
          <xsl:attribute name="code-system">iso3166</xsl:attribute>
         </xsl:when>
        <xsl:otherwise>
         <xsl:attribute name="{name()}">
           <xsl:value-of select="."/>
         </xsl:attribute>
        </xsl:otherwise>
      </xsl:choose>
     </xsl:for-each>
     <xsl:if test="not(@*[name()='code-system'])">
      <xsl:attribute name="code-system">iso3166</xsl:attribute>
     </xsl:if>
     <xsl:apply-templates/>
   </xsl:element>
  </xsl:template>

  <xsl:template match="xi:include">
   <xsl:variable name="file">
     <xsl:value-of select="substring-before(@href, '#')"/>
   </xsl:variable>
   <xsl:variable name="id">
     <xsl:value-of select="substring-after(@href, '#')"/>
   </xsl:variable>
   <xsl:apply-templates select="document($file)//dsrl:fragment[@id=$id]"/>
  </xsl:template>
   <xsl:template match="dsrl:fragment">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="*|@*">
   <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates select="*|text()|comment()|processing-instruction()"/>
   </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

In addition the following fragment is added to the local fragment store:

```
<?xml version="1.0" encoding="UTF-8"?>
<dsrl:fragments xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
 <dsrl:fragment id="title-page-form">
  <us:title-page xmlns:us="http://we.are.us">
   <us:title dsrl:request-content="true"/>
   <us:subtitle dsrl:request-content="optional"/>
   <us:author dsrl:request-content="true"
    dsrl:request-attributes="title initial nickname"/>
     </us:title-page>
   <us:publisher xmlns:us="http://we.are.us">Get A Life Publishing plc</us:publisher>
  </dsrl:fragment>
</dsrl:fragments>
```

When this map is used to transform the French document instance the following European-style address is produced:

```
<?xml version="1.0" encoding="UTF-8"?><?PIname PI proceeds root?>
<doc xmlns:xi="http://www.w3.org/2001/XInclude">
    <address xmlns="http://csw.co.uk/addresses"
     targetNamespace="http://csw.co.uk/addresses"
     targetSchemaLocation="EuropeanAddress.xsd" type="maison">
      <building-identifier>29</building-identifier>
      <road>Rue Bricot</road>
      <locality>Monmartre</locality><?PIname Embedded PI?>
      <postal-town>Bordeaux</postal-town>
      <county>Île de France</county>
      <postcode>95010</postcode>
      <country code-system="iso3166">France &lt;this&amp;that&gt;</country>
   </address>
     <address xmlns="http://csw.co.uk/addresses"
      targetNamespace="http://csw.co.uk/addresses"
      targetSchemaLocation="EuropeanAddress.xsd" type="bureau">
      <building-identifier>2</building-identifier>
      <road>Avenue Charles de Gaulle</road><?ProcessThis Another mapped PI?>
      <postal-town>Bordeaux</postal-town>
      <county>Audé &amp; Dauphine</county>
      <postcode>12345</postcode>
      <country code-system="iso3166">France</country>
      <locality>Downtown</locality>
   </address>
   <us:title-page xmlns:us="http://we.are.us"
    xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">
    <us:title dsrl:request-content="true"/>
    <us:subtitle dsrl:request-content="optional"/>
    <us:author dsrl:request-content="true"
     dsrl:request-attributes="title initial nickname"/>
    </us:title-page>
   <us:publisher xmlns:us="http://we.are.us"
    xmlns:dsrl="http://purl.oclc.org/dsdl/dsrl">Get A Life Publishing plc</us:publisher>
</doc><?PIname PI follows root?>
```

B.2 Converting DSRL rules within schemas

DSRL rules embedded within a schema can be converted into XSLT transformation rules that can be used to convert a schema into a multi-name validation schema by use of the following XSLT transform:

```
To be defined
```

B.3 Converting DSRL rules within document instances

DSRL rules embedded within a document intances can be converted into XSLT transformation rules that can be used to transform the document instance into a form that can be used to validate the document instance by use of the following XSLT transform:

```
To be defined
```

B.4 Using XSLT to generate XForms

This annex contains an XSL transformation that will convert any incomplete document fragment into an XForm whose result, after completion of all fields, will be a complete document fragment that can be incorporated into a document instance using XInclude.

```
To be defined
```

Bibliography

[1] *XSL Transformations (XSLT) Version 1.0*, http://www.w3.org/TR/xslt

Summary of editorial comments:

[5.1] Reassigning element and attribute names

Should we identify the targetNamespace and/or target schema/DTD as part of the map? Should the mapping be made on the outer dsrl:maps element or the element that is to become the root of the output? If the target schema/DTD is not identified how do we validate the result of the mapping? Does tying a map to a schema/DTD detract from its reusabilty across versions of schemas/DTDs?

[5.1] Reassigning element and attribute names

Discussion is required to ascertain whether the inclusion of a map within a schema is implementable. How would the inclusion of a DSRL map be distinguished from the inclusion of a schema fragment? How would the system process such a map?

[5.2] Mapping attribute value tokens

Is this a valid reason for having a non-standard form for the target? Would it be just as simple to convert a `adresse/@sorte` input to `match="addresse[@sorte='maison']`?

[5.3] Mapping entity references

Jirka states "I don't think externally defined dsrl:entity-name-map should have a target attribute. Entity declarations are always valid for the whole document instance." I would argue, however, that mappings may need to be scoped, because within one element they may need to be mapped but in another they may not need to be mapped. This needs discussion in committee.

[5.4] Renaming processing instruction targets

Ways in which maps can be invoked before processing of a schema need to be discussed further. There may be a requirement to be able to invoke the process using techniques described in Part 10 of this standard, but at present pipelining lanagues do not seem to generally offer such functionality.

[5.4] Renaming processing instruction targets

This version changes the order of the two entries in a mapping pair. Is this order "natural"?

[5.4] Renaming processing instruction targets

How could we indicate, using RNC or Schematron, that the `dsrl:property-names` attribute can occur in the PI itself? If we did this would it be possible to have a `dsrl:PI-name` attribute as well or would this be too late as the PI name has already been processed by the time the name map is encountered?

[5.4] Renaming processing instruction targets

Should people be allowed to apply PI maps to PIs whose target is XML? Would it be acceptable to allow property-names to be mapped but discourage the use of alternatives to the XML target-name?

[6.1] Default element content

Jirka claims that we need to add `[a:defaultValue="Martin Bryan"]` immediately after the first { to "define a fixed attribute". Jirka states this is needed for RELAX NG DTD Compatibility. Is this essential? Would users understand the reason for it if we added it without comment?

[6.2] Default attribute values

Most schema languages already contain mechanisms for assigning default values to attributes. Whilst the approach suggested here allows `dsrl:maps` to contain all of the relevant default values, is this enough of an advantage to justify duplication of existing functionality?

[7] Defining a document fragment template

Should it be possible to define fragments at element level rather than schema level?

[7] Defining a document fragment template

Is my use of braces to surround the spec supplied by Jirka correct? Is there any restriction on the use of spaces betweeen the two closing square brackets? Is the above valid? Where would the `xi` namespace need to be defined for use in the compact syntax?

[7] Defining a document fragment template

How can we change this into a compact syntax definition?

[7] Defining a document fragment template

Should we provide an attribute that allows users to specify the wording to be used for prompting for specific pieces of data? If so how would this work for attributes? Alternatively should we provide a pointer to a form that contains the relevant prompts?

[2] RELAX NG Compact Schema for Validating DSRL

Is the definition of noSpaceString valid? If not, how can we show that the string must not contain a space? Can \s be used within a character set definition?

[2] RELAX NG Compact Schema for Validating DSRL

Do foreign elements exclude elements defined in a different namespace within the schema?

Does xsd:QName allow you to have both namespace qualified and unqualified names? The RNC CName construct requires a prefix. What construct should RNC use to indicate that both qualified and unqualified names can occur at a particular point in a construct?)