

Contents

Page

Foreword.....	iv
Introduction.....	v
1 Scope.....	1
2 Normative references.....	1
3 Terms and definitions.....	2
4 The role of the Document Schema Renaming Language.....	2
4.1 Namespace.....	2
5 Mapping user-defined names to schema-defined names.....	3
5.1 Reassigning element and attribute names.....	3
5.2 Remapping entity references.....	4
5.3 Renaming Processing Instruction Targets.....	6
6 Assigning default values.....	7
6.1 Default element content.....	7
6.2 Default attribute values.....	7
7 Defining a document fragment template.....	8
Annex A (normative) Validation of declarative document architectures.....	10
A.1 RELAX NG XML Schema for Validating DSRL.....	10
A.2 RELAX NG Compact Schema for Validating DSRL.....	10
A.3 Schematron Rules for Validating DSRL.....	12
Annex B (informative) Using DSRL and XSLT to Transform Schemas and Documents	13
B.1 Converting DSRL rules within schemas.....	13
B.2 Converting DSRL rules within document instances.....	13
B.3 Using XSLT to generate XForms.....	13
Bibliography.....	14

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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 Stylesheet 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 allow 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
- removing elements or attributes from specified locations within the document model
- defining named fragments of predefined 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 2396, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Standards Track Specification, August 1998, <http://www.ietf.org/rfc/rfc2396.txt>

XML, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 6 October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>

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 different 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.

In some circumstances you need to suppress the occurrence of specific elements within a particular document instance based on analysis of associated elements. DSRL provides a mechanism for identifying elements that must be removed from document instances if found in specified contexts.

NOTE 3: This functionality mimics that provided by exclusions in SGML DTDs.

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-specific information, or
- assign a `dsrl:map-names` attribute 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:name-map` element identifies reusable mappings between names used in the schema and those used in document instances that can be validated against it. The `dsrl:name-map` element can be included as a foreign element in the documentation of any element or attribute in the schema. The contents of the 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:name-map` element will typically appear as:

```
<define name="contents">
  <element name="body">
    <a:documentation>The readable contents of a document</a:documentation>
    <attribute name="cols">
      <dsrl: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 4: 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.

A `dsrl:element-name-map` or `dsrl:attribute-names-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 name token identifying the name assigned in the locally applicable schema to the element whose model this element instance is to be validated against. The value of the `dsrl:attribute-names-map` attribute is a pair 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 locally applicable schema to define the values that can be applied to the attribute. A typical example of a local redefinition of an attribute name 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 also 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:name-map` can be used in a separate declaration of name maps within a `dsdl8:name-maps` container. Within this element 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) to which this map is to be applied.

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

```
<dsrl:name-maps>
  <dsrl:name-map target="annexes">appendices</dsrl:name-map>
```

```
<dsrl:name-map target="body[@cols]">columns no-of-columns</dsrl:name-map>
...
</dsrl:name-maps>
```

To associated a name map with a schema a `dsrl:name-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.jtclsc34.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:name-maps="http://www.jtclsc34.org/examples/dsrl-map-example.xml">
```

The `dsrl:name-maps` attribute can also be used to identify the map 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 map. The following example shows how different maps can be applied at different levels in a document instance:

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

More than one `dsrl: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 the 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.

Name maps are inherited by descendants. Where all attributes of 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. A typical record will have the form:

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

5.2 Remapping 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 (`&`, `<`, `>`, `'` and `"`).

NOTE 8: 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-specific 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 consists 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 name maps, e.g.:

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

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.

NOTE 9: 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 10: 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, the initial `&` being replaced by `&` to ensure that the replaced name is not immediately expanded.

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

In some instances entity mapping may need 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 specified element and any children that do not have a value of `false` for the same attribute are inhibited.

5.3 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, or
- define mappings within an externally defined name map that can, optionally, be invoked as a parameter during processing.

NOTE 12: 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.

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 is 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 a property name applicable to processing instructions whose target is being redefined, and the second of which is an alternative name used for that property within the document instance.

NOTE 13: 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="encoding character-set version level">
```

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.

A `dsrl:default-content` element is used to define a default content for an element defined 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 document instance. A typical application might be:

```
<define name="RD-name">
  <element name="name">
    <a:documentation>Name of Research Department manager</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 14: 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}
```

6.2 Default attribute values

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

- create an attribute defaults declaration that defines the contents 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.

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. The element can be included as a foreign element in the documentation of the appropriate attribute in the schema. 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 document instance.

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 15: 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 content that is 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 replace any values found in the document instance.

NOTE 16: 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.

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 in as direct children of schemas.¹ The `dsrl:predefined-fragment` element may be used to identify document fragment templates within a schema. This element can be defined in two ways:

- 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` attribute identifies which fragment of the referenced resource is to be included, or
- as an element within an externally defined map.

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.

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 17: 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:

```
<define name="intro" xmlns:xi="http://www.w3.org/2001/XInclude">
  <element name="intro">
    <a:documentation>Introduction</a:documentation>
    <dsrl:document-fragment dsrl:name="template-1"
      dsrl:schema-source="http://us.org/schemas/introductions.dtd"/>
    <text><xi:include href="#template-1"/></text>
  </element>
</define>
```

How can we include a reference to a document fragment in a compact syntax definition?

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.

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` should 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 18: 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 19: 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 attributes are still to be provided with values when the document fragment template is included into a document instance.

```
<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>
    <dsrl:document-fragment 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>
    <text><xi:include href="#title-page-form"/></text>
  </element>
</define>
```

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:name-maps
## Allow maps to be imported from external sources
external-name-maps = attribute dsrl:name-maps {anyURI}
extrnal-map-use |= element * {external-name-maps}
## Identify target element to which map is to be applied
target = attribute target {pathValue}
pathValue = string
## Identify inheritance 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
name-maps = element dsrl:name-maps {(name-map | entity-map | property-names |
                                element-content)+}
name-map = element dsrl:name-map {target, inherit?, {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-template {any, target, attribute name {ID},
              attribute href {anyURI}?,
              attribute dsrl:schema-source {anyURI}?}

```

```
request-content |= element * {attribute dsrl:request-content ("true" | "false")?,
                             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?

The following RELAX NG compact syntax scheme 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-name-maps = attribute dsrl:name-maps {anyURI}
extrnal-map-use |= element * {external-name-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"}
name-map = element dsrl:name-map {inherit?, xsd:QName+}
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-template {any, attribute name {ID},
                attribute href {anyURI}?,
                attribute dsrl:schema-source {anyURI}?}
request-content |= element * {attribute dsrl:request-content ("true" | "false")?,
                             attribute dsrl:request-attributes {xsd:NAMES}?}
```

The following RELAX NG compact syntax scheme 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 {QName}
attribute-names = attribute dsrl:attribute-names-map {(xsd:QName, xsd:QName)+}
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 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.2 Converting DSRL rules within document instances

DSRL rules embedded within a document instances 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.3 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:

[5.3] Renaming Processing Instruction Targets

How could we indicated, 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?

[5.3] 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?

[7] Defining a document fragment template

How can we include a reference to a document fragment in a compact syntax definition?

[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?

[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?)