Contents

Page

Foreword

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

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

ISO/IEC 19757-9 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 34, Document Description and Processing Languages.

— *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: Datatype Library Language — DTLL*

— *Part 6: Path-based integrity constraints*

— *Part 7: Character Reportoire Description Language — CRDL*

— *Part 8: Document Schema Renaming Language — DSRL*

— *Part 9: Namespace and datatype declaration in DTDs*

— *Part 10: Validation Management*

Introduction

The language of Document Type Definitions (DTDs) was the original schema language defined by W3C XML and was closely based upon the DTD language defined by ISO 8879, SGML. For a variety of reasons, both technical and economic, many users of XML for document-centric applications, especially among those who were previously (and in some cases continue to be) users of SGML, still favour the use of DTD language for grammar-based schema definition in such applications.

It is important to provide users that have made a significant investment in DTDs with a migration path that will enable them to adopt DSDL without having to translate all their existing DTDs to a different schema language, especially as this would oblige them to replace all systems that only work with DTDs, with all the expense and organizational upheavals thereby entailed. A sensible migration path should enable such users to continue to use DTDs for as much of the document validation process as can reasonably be managed, but also enable them to reap the benefits of using those parts of DSDL that most obviously complement and extend the use of DTDs for validation purposes.

It is equally important that the migration path should enable users to continue to use legacy systems that are incapable of using any kind of extension to the DTD language, while at the same time introducing new systems that are equipped to use such extensions. The method of extension must therefore be such that DTDs with extended functionality are valid XML DTDs in accordance with W3C XML. It should remain possible to validate an instance *that does not make any use of the extended functionality* against a DTD that contains the extended functionality, using legacy system tools, and achieve the same result as would be achieved if the DTD did not contain the extended functionality.

The most significant validation tasks that cannot be performed using a DTD alone are:

— validation of names with respect to namespaces

— validation of data content and attribute values with respect to datatypes

— rules-based validation.

Of these three, the last is made possible by implementation of Part 3 of DSDL. The first two tasks are currently only possible if Part 2 of DSDL is implemented, but existing DTD users are unlikely to wish to maintain parallel RELAX NG and DTD schemas for each application simply as a means of supporting the use of namespaces and datatypes.

For these reasons this Part of DSDL addresses the extension of DTDs to support the declaration of namespaces and datatypes.

Document Schema Definition Languages (DSDL) — Part 9: Namespace and datatype declaration in DTDs

1 Scope

This Part of DSDL defines a language that is designed to extend the declarative functionality of an XML DTD to include:

— declaring one or more namespaces to which some or all of the element and attribute names in a DTD belong

— declaring constraints on the content of elements with content model ANY to contain elements whose names belong to one or more specified namespaces

— declaring datatypes for elements that contain data content only and for attribute values.

Two alternative syntax bindings for this language are defined. The first syntax binding uses XML processing instructions and is designed to enable declarations in this language to be embedded within an XML DTD without invalidating the DTD or altering its interpretation so far as legacy DTD parsers are concerned. This first syntax also provides a means of associating a DTD with an external subset containing declarations in either syntax. This syntax is defined in Clause 4 using the modified BNF syntax notation used in W3C XML.

The second syntax binding uses an XML document syntax and is defined in Clause 5. The syntax rules are defined by a schema that conforms to the RELAX NG Compact Syntax defined in Part 2 of DSDL. This syntax is designed to enable declarations in this language to be expressed almost entirely in XML (in this case one XML processing instruction needs to be inserted in the DTD), to facilitate implementation using existing XML tools, either as a namespace-qualified fragment embedded within an XML instance or as a separate XML document.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

SGML, *ISO 8879:1986 Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*, http://www.iso.org/

W3C XML, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-20060816/

W3C XML-Names, *Namespaces in XML 1.0 (Second Edition)*, W3C Recommendation, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-names-20060816/

RELAX NG, *Grammar-based schema language (RELAX NG)*, ISO/IEC 19757 Part 2, http://www.dsdl.org/

DTLL, *Datatype Library Language*, ISO/IEC 19757 Part 5, http://www.dsdl.org/

IRI, *Internationalized Resource Identifiers (IRIs)*, IETF RFC 3987, January 2005, http://www.ietf.org/rfc/rfc3987

W3C XML Schema Datatypes, *XML Schema Part 2: Datatypes (Second Edition)*, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply:

3.1 external declarations subset

An XML resource containing a set of DTD extension declarations either in XML processing instruction syntax or in XML document syntax in accordance with this Part of DSDL.

3.2 DSDL-9-aware parser

A validating XML processor that is able correctly to interpret and process the extension declarations in accordance with this Part of DSDL, and is therefore able to validate instances whose validity or otherwise can only be correctly determined with reference to those extension declarations.

3.3 legacy parser

A validating XML processor that is unable correctly to interpret and process extension declarations in accordance with this Part of DSDL, and is therefore unable to validate instances whose validity or otherwise can only be correctly determined with reference to those extension declarations.

3.4 qualified datatype name

A datatype name containing a qualifying prefix and a local part separated by a colon ':', the prefix of which is bound to the IRI of a datatype library by a datatype library prefix binding declaration as described in this standard.

3.5 unqualified datatype name

A datatype name that does not contain a colon ':' and therefore contains only a local part.

4 DTD extension declarations using XML processing instruction syntax

The following set of syntax productions define how namespace and datatype declarations expressed in accordance with this standard may be represented as XML processing instructions. Syntax productions for *Enumeration*, *s* and *SystemLiteral* are given in W3C XML. The syntax production for *NCName* is given in W3C XML-Names.

DTD-extension-processing-instruction ::= *pio* "DSDL-9" *s* (*namespace-prefix-binding-declaration* | *namespace-name-binding-declaration* | *wildcard-namespace-qualifier-declaration* | *default-datatype-library-declaration* | *datatype-library-prefix-binding-declaration* | *datatype-binding-declaration* | *external-declarations-subset-locator*) *pic*

namespace-prefix-binding-declaration ::= "namespace-prefix-binding" *s* "namespace-iri=" *IRI-literal s* "prefix=" *prefix-literal*

namespace-name-binding-declaration ::= "namespace-name-binding" *s* "namespace-iri=" *IRI-literal s* "applies-to-element=" *name-locator-literal*

wildcard-namespace-qualifier-declaration ::= "wildcard-namespace-qualifier" *s* "namespace-iri=" *wildcard-namespace-literal s* "applies-to-element=" *name-locator-literal*

wildcard-namespace-literal ::= ((*lit wildcard-namespace-sequence lit*) | (*lita wildcard-namespace-sequence lita*))

wildcard-namespace-sequence ::= *IRI* (*s IRI*)*

name-locator-literal ::= ((*lit name-locator-sequence lit*) | (*lita name-locator-sequence lita*))

name-locator-sequence ::= (*s*)? (*Enumeration* | *any-name*) (*s*)?

any-name ::= "#any"

default-datatype-library-declaration ::= "default-datatype-library-iri=" *IRI-literal*

datatype-library-prefix-binding-declaration ::= "datatype-library-prefix-binding" *s* "datatype-library-iri=" *IRI-literal s* "prefix=" *prefix-literal*

datatype-binding-declaration ::= "datatype-binding" *s* "datatype-name=" *datatype-literal s* (("applies-to-element=" *name-locator-literal*) | ("applies-to-attribute=" *name-locator-literal* "of-element" *name-locator-literal*))

datatype-name ::= *NCName*

datatype-qname ::= *prefix* ":" *datatype-name*

datatype-literal ::= ((*lit* (*datatype-name* | *datatype-qname*) *lit*) | (*lita* (*datatype-name* | *datatype-qname*) *lita*))

prefix-literal ::= ((*lit prefix lit*) | (*lita prefix lita*))

prefix ::= *NCName*

external-declarations-subset-locator ::= "external-declarations-subset" *s* "location=" *IRI s* "syntax=" ((*lit* ("pi" | "xml") *lit*) | (*lita* ("pi" | "xml") *lita*))

IRI ::= *SystemLiteral*

pio ::= ">?"

pic ::= "?<"

lit ::= "'"

lita ::= '"'

NOTE - An *IRI* must be a URI that conforms to IRI.

5 DTD extension declarations using XML document syntax

The following schema in RELAX NG Compact Syntax specifies how a set of namespace and datatype declarations expressed in accordance with this standard may be represented as an XML document.

```
datatypes xsd = "http://www.w3.org/2001/XMLSchema-datatypes"
default namespace = "http://dsdl.org/dsdl-9"

start = document

document = element dtd-extension { head, body }

head = (element applies-to-dtd {public | system} )*

public = element public {text}, system?

system = element system {xsd:anyURI}

body = ( namespace-prefix-binding |
namespace-name-binding |
wildcard-namespace-qualifier |
default-datatype-library |
datatype-library-prefix-binding |
datatype-binding )+

namespace-prefix-binding = element namespace-prefix-binding
{ namespace-iri, prefix }

namespace-name-binding = element namespace-name-binding
{ namespace-iri, applies-to-element}

applies-to-element = element applies-to-element
{ (name+ | any-name) }

applies-to-attribute = element applies-to-attribute
{ (name+ | any-name), of-element }

of-element = element of-element
{ name+ | any-name }
```

```
name = element name {xsd:NCName}

any-name = element any {empty}

wildcard-namespace-qualifier = element wildcard-namespace-qualifier
{ (namespace-iri+, applies-to-element) }

default-datatype-library = element default-datatype-library
{ library-iri }

datatype-library-prefix-binding = element datatype-library-prefix-binding
{ (library-iri, prefix) }

library-iri = element library-iri {xsd:anyURI}

datatype-binding = element datatype-binding
{ name, (applies-to-element | applies-to-attribute) }

namespace-iri = element namespace-iri {xsd:anyURI}

prefix = element prefix {xsd:NCName}
```

6 Location and ordering of DTD extension declarations

DTD extension declarations in processing instruction syntax must be located in one or more of the following locations:

— a DTD's external subset,

— the local subset of a DTD declaration, or

— an external declarations subset whose location IRI is specified by declaring an external declarations subset locator within the DTD's external or local subset.

DTD extension declarations in XML document syntax must be located as an external declarations subset whose location IRI is specified by declaring an external declarations subset locator within the DTD's external or local subset.

The ordering of all the DTD extension declarations that are associated with a DTD is the same ordering that would be obtained by substituting all extension declarations in XML document syntax with equivalent declarations in processing instruction syntax and by treating each external declarations subset as if it were an external parameter entity of the DTD and its corresponding locator declaration as if it were a reference to that external parameter entity.

7 Inclusion of namespace information in DTDs

A namespace IRI may be associated with the names of any elements that are declared in a DTD, including both qualified and unqualified names. A namespace IRI may also be declared as a constraint on the content of an element whose declared content model is ANY.

7.1 Associating namespaces with qualified names

A namespace IRI may be associated with specified element names or attribute names that include qualifying prefixes, by declaring a corresponding namespace prefix binding.

Each qualifying prefix may only be bound once to a namespace IRI. If two or more namespace prefix binding declarations specify the same qualifying prefix, the second and subsequent declarations are ignored.

7.2 Associating namespaces with unqualified names

A namespace IRI may be associated with specified element names that do not include qualifying prefixes, by declaring a corresponding namespace name binding.

An unqualified name has no namespace unless a namespace name binding is declared that applies to this name.

A namespace IRI may be associated with any element name for which a namespace name binding has not previously been declared, by declaring a namespace name binding in which the "any name" indicator is included instead of a list of names. Any subsequent namespace name binding declarations are ignored.

Each unqualified name may only be bound once to a namespace IRI. If two or more namespace name binding declarations explicitly specify the same name, the second and subsequent declarations are ignored.

7.3 Associating namespace constraints with elements with content model ANY

A namespace IRI may be associated with the content of an element whose content model is declared to be ANY, by defining a wildcard namespace qualifier declaration for that element. A namespace IRI may only be specified in a wildcard namespace qualifier declaration if it occurs either in a namespace prefix binding declaration or in a namespace name binding declaration for the same DTD.

A wildcard namespace qualifier declaration may only associate a namespace IRI with the content of elements whose content model is ANY. The name of any element whose content model is not ANY will be ignored.

8 Inclusion of datatype information in DTDs

A datatype may be associated with a specified element or attribute, by declaring (in a datatype binding declaration) a datatype name together with the name of the element or attribute to which it applies.

Every datatype name must be associated with a declared datatype library. Unqualified datatype names are associated with a default datatype library, while qualified datatype names are associated with declared datatype libraries by declaring a binding between the qualifying prefix and the associated datatype library.

The IRI specified in either a datatype library prefix binding declaration or a default datatype library declaration must identify a datatype library, for example a library that conforms to Part 5 of this International Standard (DTLL).

If two or more datatype binding declarations apply to the same element content or attribute value, the second and subsequent declarations are ignored.

If a datatype binding declaration is associated with an element whose content model is mixed or ANY, the datatype binding declaration shall only apply to instances of that element that contain a single text node.

8.1 Associating datatype libraries with qualified datatype names

If a specified datatype name includes a qualifying prefix, a corresponding datatype library prefix binding must be declared.

Each qualifying prefix may only be bound once to a datatype library. If two or more datatype library prefix binding declarations specify the same prefix, the second and subsequent declarations are ignored.

8.2 Associating datatype libraries with unqualified datatype names

If a specified datatype name has no qualifying prefix, a default datatype library must be declared.

If two or more default datatype libraries are declared, the second and subsequent default datatype library declarations are ignored.

9 Validity of DTDs and instances

A legacy parser - i.e. a validating XML processor that is unable to process XML processing instructions that contain

extension declarations in accordance with this Part of DSDL - is by itself incapable of correctly parsing all instances for which the DTDs contain extension declarations. This clause details the difference between stand-alone legacy parsers and DSDL-9-aware parsers in the way in which they report the validity or otherwise of DTDs and instances.

It is possible to construct a DSDL-9-aware parser that uses a legacy parser as one of its components. See Annex A for description of an example processing model that involves the use of a legacy parser. This clause assumes the standard definition of a legacy parser as defined in Clause 3.

9.1 Validity of a DTD that includes extension declarations

If a DTD that includes extension declarations in accordance with this Part of DSDL is valid according to a legacy parser, it must be valid according to a DSDL-9-aware parser, regardless of whether or not there are errors in the extension declarations or inconsistences between the extension declarations and the other declarations in the DTD.

Similarly, if such a DTD is invalid according to a legacy parser, it must be invalid according to a DSDL-9-aware parser, which must report the same errors as a legacy parser.

If such a DTD is valid according to a legacy parser and contains extension declarations that are intended to be expressed in accordance with this Part of DSDL, but any of these declarations contain errors of any kind, a DSDL-9-aware parser shall report all such errors in warning messages but shall not report the DTD as being invalid.

9.2 Validity of an instance whose associated DTD does not include extension declarations

If an instance is valid according to a legacy parser and its DTD does not contain extension declarations in accordance with this Part of DSDL, it must be valid according to a DSDL-9-aware parser.

Similarly, if such an instance is invalid according to a legacy parser and its DTD does not contain extension declarations in accordance with this Part of DSDL, it must be invalid according to a DSDL-9-aware parser, which must report the same errors as a legacy parser.

9.3 Validity of an instance whose associated DTD includes extension declarations

If an instance whose associated DTD includes extension declarations is parsed using a legacy parser, the output of the parser will take no account of the extensions and results are likely to be misleading. At worst a legacy parser will only reliably report whether or not the instance is well-formed.

Results are especially unreliable in the case of the use of namespace declarations and qualifying prefixes in the instance. Unless the instance makes exactly the same use of qualifying prefixes and default namespaces as the DTD, the validity of the instance is uncertain. For example, a legacy parser will report as invalid an instance that is namespace-valid, but which happens to include namespace declaration attributes on elements for which such attributes are not declared in the DTD.

The following describes the level of unreliability that can be expected when using a legacy parser with instances and DTDs that use extension declarations in accordance with this Part of DSDL.

A legacy parser will report as valid an instance that a DSDL-9-aware parser will report as invalid in the following circumstances:

— If the instance contains an occurrence of an element or attribute whose name is qualified by a specific prefix and the DTD contains a valid attribute declaration for a (required and fixed) namespace declaration attribute for that prefix on that element or on an ancestor element, and the occurrence in the instance contains the required namespace declaration attribute, but the DTD contains no namespace prefix binding declaration for that prefix.

— If the instance contains an occurrence of an element whose name is unqualified and the DTD contains a valid attribute declaration for a (required and fixed) default namespace declaration attribute on that element or on an ancestor element, and the occurrence in the instance contains the required default namespace declaration attribute, but the DTD contains no namespace name binding declaration for that element's name.

— If the DTD contains a wildcard namespace qualifier declaration for a given element whose content model is declared to be ANY, and the instance contains occurrences of this element that contain a child element whose name is in a namespace other than those listed in the wildcard namespace qualifier declaration.

— If the DTD contains a datatype binding declaration for the data content of a specified element or value of a specified attribute, and the instance contains occurrences of this element or attribute whose data content or value is not in the datatype's value range.

NOTE: If a DTD contains a datatype binding declaration for the value of an attribute that is inconsistent with that attribute's declared XML attribute type, all instances based upon such a DTD will always be invalid, even if a DSDL-9-aware parser is unable to determine that the inconsistency is in the DTD.

A legacy parser will report as invalid an instance that a DSDL-9-aware parser will report as valid in the following circumstances:

— If the DTD contains an attribute declaration for a namespace declaration attribute on a specific element and also contains the appropriate namespace binding declaration in accordance with this Part of DSDL, and the instance legitimately contains the corresponding namespace declaration attribute on an ancestor of the specified element and not on the specified element.

— If the DTD contains a namespace prefix binding declaration, and the instance includes a namespace declaration attribute for the associated prefix that is not declared in the DTD.

— If the DTD contains a namespace name binding declaration, and the instance includes a default namespace declaration attribute on an occurrence of the element with that name or on an ancestor, and this attribute is not declared in the DTD.

— If the DTD follows a different convention for the use of qualifying prefixes and default namespaces to that followed in the instance, but the DTD contains the appropriate namespace binding declarations in accordance with this Part of DSDL.

Annex A
(informative)

Example processing model

A DSDL-9-aware parser could be implemented in a number of ways, but is most likely to be implemented as a pipeline process built on top of a legacy parser. If this were the case, the following processing model might be chosen.

1. The namespace binding declarations in the DTD are read and a table of namespaces constructed in which each distinct namespace is represented by a canonical prefix.

2. Using the table of namespaces, names in both the DTD and instance are transformed so that all names in specified namespaces are qualified using the corresponding canonical prefix. The purpose is to avoid prefix conflicts by transforming prefixes in both the DTD and instance to a canonical form for each distinct namespace.

3. All existing attribute declarations for namespace declaration attributes are removed from the DTD and replaced by a canonical set of namespace declaration attribute declarations on every element in the DTD. The canonical set of attribute declarations comprises one qualified namespace declaration for each canonical prefix, with a required and fixed declared value, and one default namespace declaration for each distinct namespace. The purpose is to ensure that a legacy parser will recognise that an instance can be valid with a namespace declaration attribute on any element for any namespace, whether qualified or default.

4. The content model of each element for which a wildcard namespace qualifier declaration is included in the DTD and which has the content model `ANY` is replaced by a mixed content model containing all elements whose names are in the namespaces specified for that element.

5. The instance is parsed using the legacy parser.

6. The instance is processed by a namespace-aware XML processor to ensure that it is namespace well-formed.

7. The datatype binding declarations in the DTD are used to construct a set of Schematron rules in accordance with Part 3 of DSDL that can be used to check that corresponding element data values and attribute values in the instance are consistent with the declared datatypes.

Annex B
(informative)

Example declarations

B.1 Examples in XML processing instruction syntax

The following example demonstrates the use of the XML processing instruction syntax to include declarations in a DTD in accordance with this Part of DSDL.

```
<?DSDL-9 namespace-prefix-binding namespace-iri="http://dsdl.org/" prefix="dsdl"?>
<?DSDL-9 namespace-prefix-binding namespace-iri="http://www.w3.org/1999/xhtml" prefix="html"?>
<?DSDL-9 namespace-name-binding namespace-iri="http://www.w3.org/1999/xhtml"
         applies-to-element="table caption col colgroup thead tfoot tbody th tr td"?>
<?DSDL-9 wildcard-namespace-qualifier namespace-iri="http://www.w3.org/1999/xhtml"
         applies-to-element="web-documentation"?>
<?DSDL-9 datatype-library-prefix-binding
         library-iri="http://www.w3.org/2001/XMLSchema-datatypes" prefix="xs"?>
<?DSDL-9 datatype-binding datatype-name="xs:anyURI" applies-to-attribute="href"
         of-element="#any"?>
<?DSDL-9 datatype-binding datatype-name="xs:string"
         applies-to-element="foo"?>
```

The following example contains the same declarations as in the preceding example, but scattered throughout a DTD fragment.

```
...
<?DSDL-9 namespace-prefix-binding namespace-iri="http://dsdl.org/" prefix="dsdl"?>
<?DSDL-9 namespace-name-binding namespace-iri="http://www.w3.org/1999/xhtml"
         applies-to-element="table caption col colgroup thead tfoot tbody th tr td"?>
<!ELEMENT table
         caption?, (col*|colgroup*), thead?, tfoot?, (tbody+|tr+))>
<!ELEMENT caption  %Inline;>
<!ELEMENT thead    (tr)+>
<!ELEMENT tfoot    (tr)+>
<!ELEMENT tbody    (tr)+>
<!ELEMENT colgroup (col)*>
<!ELEMENT col      EMPTY>
<!ELEMENT tr       (th|td)+>
<!ELEMENT th       %Flow;>
<!ELEMENT td       %Flow;>
...
<!ELEMENT web-documentation ANY>
<?DSDL-9 wildcard-namespace-qualifier namespace-iri="http://www.w3.org/1999/xhtml"
         applies-to-element="web-documentation"?>
...
<?DSDL-9 namespace-prefix-binding namespace-iri="http://www.w3.org/1999/xhtml" prefix="html"?>
<?DSDL-9 datatype-library-prefix-binding
         library-iri="http://www.w3.org/2001/XMLSchema-datatypes" prefix="xs"?>
<?DSDL-9 datatype-binding datatype-name="xs:anyURI" applies-to-attribute="href"
         of-element="#any"?>
<!ELEMENT html:a %a.content;>
<!ATTLIST html:a
          ...
          href CDATA #IMPLIED
          ...
>
...
<!ELEMENT foo {#PCDATA)>
<?DSDL-9 datatype-binding datatype-name="xs:string"
         applies-to-element="foo"?>
...
```

The following example demonstrations the use of a single XML processing instruction to associate a DTD with an

external set of declarations in XML document syntax in accordance with Clause 5.

```
<?DSDL-9 external-declarations-subset location="http://www.example.com" syntax="xml"?>
```

B.2 Example in XML document syntax

The following example contains the same declarations as in the preceding clause, but in XML syntax.

```
<?xml version="1.0" encoding="UTF-8"?>
<dtd-extension xmlns="http://dsdl.org/dsdl-9">
     <namespace-prefix-binding>
          <namespace-iri>http://dsdl.org/</namespace-iri>
          <prefix>dsdl</prefix>
     </namespace-prefix-binding>
     <namespace-prefix-binding>
          <namespace-iri>http://www.w3.org/1999/xhtml</namespace-iri>
          <prefix>html</prefix>
     </namespace-prefix-binding>
     <namespace-name-binding>
          <namespace-iri>http://www.w3.org/1999/xhtml</namespace-iri>
          <applies-to-element>
               <name>table</name>
               <name>caption</name>
               <name>col</name>
               <name>colgroup</name>
               <name>thead</name>
               <name>tfoot</name>
               <name>tbody</name>
               <name>th</name>
               <name>tr</name>
               <name>td</name>
          </applies-to-element>
     </namespace-name-binding>
     <wildcard-namespace-qualifier>
          <namespace-iri>http://www.w3.org/1999/xhtml</namespace-iri>
          <applies-to-element>
               <name>web-documentation</name>
          </applies-to-element>
     </wildcard-namespace-qualifier>
     <datatype-library-prefix-binding>
          <library-iri>http://www.w3.org/2001/XMLSchema-datatypes</library-iri>
          <prefix>xs</prefix>
     </datatype-library-prefix-binding>
     <datatype-binding>
          <name>xs:anyURI</name>
          <applies-to-attribute>
               <name>href</name>
               <of-element>
                    <any/>
               </of-element>
          </applies-to-attribute>
     </datatype-binding>
     <datatype-binding>
          <name>xs:string</name>
          <applies-to-element>
               <name>foo</name>
          </applies-to-element>
     </datatype-binding>
</dtd-extension>
```

Summary of editorial comments: