Normalization for XML DTD 1.0 files:
A Specification of the XML_DTD NORM_1 Format

Release Date: 07/24/2003

Author: Andrea Goethals, FCLA

Change History:

XML_DTD_norm_1 format: A single XML DTD file

Methodology (Some of the terms in the following paragraphs are defined in the specification of the XML_norm_1 format):

Each XML DTD 1.0 document that is submitted for archive ingest is parsed to see if it contains any file references to external files of any file format. If it does not, no normalized version is created for the DTD. If it contains at least one file reference, the archive program determines one-by-one if the physical files can be located that correspond to the named references. If at least one file reference can be resolved, a new DTD document (the normalized version) is created.

In the new DTD, the URIs of resolvable file references are substituted with the DFIDs plus file extensions of the referenced files. For example, "http://www.example.org/1.pdf" might be replaced with "AAAA2388.pdf", where 'AAAA2388' is the unique ID of the PDF in the archive. Any unresolvable file references to files are left 'as is' and these references are recorded in the archive database as 'broken links'. Broken links will not cause a package to be rejected from the ingest process.

In some cases, the externally-referenced file will also be a DTD or other format which can contain file references. The archive program will parse them to see if they have any file references. The way in which the archive program resolves the file references for these 'second tier' files is different than the way references are resolved for the 'first tier' (submitted) DTDs. The exact algorithm is contained in a separate FCLA document: "Resolving References in Distributed Objects", available on the FCLA digital archive website.

Detecting External References within DTD Documents:
The archive program will look for links to external files within DTD documents in the following locations (see the Action Plan Background of XML DTD 1.0 Documents for examples of each):

1. Within a general entity declaration
2. Within a parameter entity declaration
3. Within a notation declaration

Example 1: Submitted DTD document: XMLSchema.dtd (This example uses the DTD for XML Schema available on the W3C website at http://www.w3.org/2001/XMLSchema.dtd).
The references to external files that would be detected by the ingest program are underlined.

```
<!-- DTD for XML Schemas: Part 1: Structures
     Public Identifier: "-//W3C//DTD XMLSCHEMA 200102//EN"
     Official Location: http://www.w3.org/2001/XMLSchema.dtd -->
```

```
<!-- $Id: XMLSchema.dtd,v 1.31 2001/10/24 15:50:16 ht Exp $ -->
<!-- Note this DTD is NOT normative, or even definitive. --> <!--d-->
<!-- prose copy in the structures REC is the definitive version --> <!--d-->
<!-- (which shouldn't differ from this one except for this --> <!--d-->
<!-- comment and entity expansions, but just in case) -->  <!--d-->
<!-- With the exception of cases with multiple namespace
     prefixes for the XML Schema namespace, any XML document which is
     not valid per this DTD given redefinitions in its internal subset of the
     'p' and 's' parameter entities below appropriate to its namespace
     declaration of the XML Schema namespace is almost certainly not
     a valid schema. -->

<!-- The simpleType element and its constituent parts
     are defined in XML Schema: Part 2: Datatypes -->
<!ENTITY % xs-datatypes PUBLIC 'datatypes' 'datatypes.dtd' >

<!ENTITY % p 'xs:'> <!-- can be overriden in the internal subset of a
                         schema document to establish a different
                         namespace prefix -->
<!ENTITY % s ':xs'> <!-- if %p is defined (e.g. as foo:) then you must
                         also define %s as the suffix for the appropriate
                         namespace declaration (e.g. :foo) -->
<!ENTITY % nds 'xmlns%s;'>

<!-- Define all the element names, with optional prefix -->
<!ENTITY % schema "%p;schema">
<!ENTITY % complexType "%p;complexType">
<!ENTITY % complexContent "%p;complexContent">
<!ENTITY % simpleContent "%p;simpleContent">
<!ENTITY % extension "%p;extension">
<!ENTITY % element "%p;element">
<!ENTITY % unique "%p;unique">
<!ENTITY % key "%p;key">
<!ENTITY % keyref "%p;keyref">
<!ENTITY % selector "%p;selector">
<!ENTITY % field "%p;field">
<!ENTITY % group "%p;group">
<!ENTITY % all "%p;all">
<!ENTITY % choice "%p;choice">
<!ENTITY % sequence "%p;sequence">
<!ENTITY % any "%p;any">
<!ENTITY % anyAttribute "%p;anyAttribute">
<!ENTITY % attribute "%p;attribute">
<!ENTITY % attributeGroup "%p;attributeGroup">
<!ENTITY % include "%p;include">
<!ENTITY % import "%p;import">
<!ENTITY % redefine "%p;redefine">
<!ENTITY % notation "%p;notation">

<!-- annotation elements -->
<!ENTITY % annotation "%p;annotation">
<!ENTITY % appinfo "%p;appinfo">
<!ENTITY % documentation "%p;documentation">

<!-- Customisation entities for the ATTLIST of each element type.
     Define one of these if your schema takes advantage of the
     anyAttribute='##other' in the schema for schemas -->

<!ENTITY % schemaAttrs ''>
<!ENTITY % complexTypeAttrs ''>
<!ENTITY % complexContentAttrs ''>
```

```
<!ENTITY % simpleContentAttrs ''>
<!ENTITY % extensionAttrs ''>
<!ENTITY % elementAttrs ''>
<!ENTITY % groupAttrs ''>
<!ENTITY % allAttrs ''>
<!ENTITY % choiceAttrs ''>
<!ENTITY % sequenceAttrs ''>
<!ENTITY % anyAttrs ''>
<!ENTITY % anyAttributeAttrs ''>
<!ENTITY % attributeAttrs ''>
<!ENTITY % attributeGroupAttrs ''>
<!ENTITY % uniqueAttrs ''>
<!ENTITY % keyAttrs ''>
<!ENTITY % keyrefAttrs ''>
<!ENTITY % selectorAttrs ''>
<!ENTITY % fieldAttrs ''>
<!ENTITY % includeAttrs ''>
<!ENTITY % importAttrs ''>
<!ENTITY % redefineAttrs ''>
<!ENTITY % notationAttrs ''>
<!ENTITY % annotationAttrs ''>
<!ENTITY % appinfoAttrs ''>
<!ENTITY % documentationAttrs ''>

<!ENTITY % complexDerivationSet "CDATA">
      <!-- #all or space-separated list drawn from derivationChoice -->
<!ENTITY % blockSet "CDATA">
      <!-- #all or space-separated list drawn from
                      derivationChoice + 'substitution' -->

<!ENTITY % mgs '%all; | %choice; | %sequence;'>
<!ENTITY % cs '%choice; | %sequence;'>
<!ENTITY % formValues '(qualified|unqualified)'>


<!ENTITY % attrDecls    '((%attribute;|
%attributeGroup;)*,(%anyAttribute;)?)'>

<!ENTITY % particleAndAttrs '((%mgs; | %group;)?, %attrDecls;)'>

<!-- This is used in part2 -->
<!ENTITY % restriction1 '((%mgs; | %group;)?)'>

%xs-datatypes;

<!-- the duplication below is to produce an unambiguous content model
     which allows annotation everywhere -->
<!ELEMENT %schema; ((%include; | %import; | %redefine; | %annotation;)*,
                    ((%simpleType; | %complexType;
                      | %element; | %attribute;
                      | %attributeGroup; | %group;
                      | %notation; ),
                     (%annotation;)*)* )>
<!ATTLIST %schema;
   targetNamespace      %URIref;               #IMPLIED
   version              CDATA                  #IMPLIED
   %nds;                %URIref;               #FIXED
'http://www.w3.org/2001/XMLSchema'
   xmlns                CDATA                  #IMPLIED
   finalDefault         %complexDerivationSet; ''
   blockDefault         %blockSet;             ''
   id                   ID                     #IMPLIED
   elementFormDefault   %formValues;           'unqualified'
   attributeFormDefault %formValues;           'unqualified'
   xml:lang             CDATA                  #IMPLIED
   %schemaAttrs;>
<!-- Note the xmlns declaration is NOT in the Schema for Schemas,
     because at the Infoset level where schemas operate,
     xmlns(:prefix) is NOT an attribute! -->
<!-- The declaration of xmlns is a convenience for schema authors -->

<!-- The id attribute here and below is for use in external references
     from non-schemas using simple fragment identifiers.
     It is NOT used for schema-to-schema reference, internal or
     external. -->

<!-- a type is a named content type specification which allows attribute
     declarations-->
<!-- -->

<!ELEMENT %complexType; ((%annotation;)?,
                         (%simpleContent;|%complexContent;|
                          %particleAndAttrs;))>

<!ATTLIST %complexType;
          name      %NCName;                        #IMPLIED
          id        ID                              #IMPLIED
          abstract  %boolean;                       #IMPLIED
          final     %complexDerivationSet;          #IMPLIED
          block     %complexDerivationSet;          #IMPLIED
          mixed (true|false) 'false'
          %complexTypeAttrs;>

<!-- particleAndAttrs is shorthand for a root type -->
<!-- mixed is disallowed if simpleContent, overriden if complexContent
     has one too. -->

<!-- If anyAttribute appears in one or more referenced attributeGroups
     and/or explicitly, the intersection of the permissions is used -->

<!ELEMENT %complexContent; ((%annotation;)?, (%restriction;|%extension;))>
<!ATTLIST %complexContent;
          mixed (true|false) #IMPLIED
          id    ID           #IMPLIED
          %complexContentAttrs;>

<!-- restriction should use the branch defined above, not the simple
     one from part2; extension should use the full model  -->

<!ELEMENT %simpleContent; ((%annotation;)?, (%restriction;|%extension;))>
<!ATTLIST %simpleContent;
          id    ID           #IMPLIED
          %simpleContentAttrs;>

<!-- restriction should use the simple branch from part2, not the
     one defined above; extension should have no particle  -->

<!ELEMENT %extension; ((%annotation;)?, (%particleAndAttrs;))>
<!ATTLIST %extension;
          base  %QName;      #REQUIRED
          id    ID           #IMPLIED
          %extensionAttrs;>
```

```
<!-- an element is declared by either:
 a name and a type (either nested or referenced via the type attribute)
 or a ref to an existing element declaration -->

<!ELEMENT %element; ((%annotation;)?, (%complexType;| %simpleType;)?,
                     (%unique; | %key; | %keyref;)*)>
<!-- simpleType or complexType only if no type|ref attribute -->
<!-- ref not allowed at top level -->
<!ATTLIST %element;
            name              %NCName;               #IMPLIED
            id                ID                     #IMPLIED
            ref               %QName;                #IMPLIED
            type              %QName;                #IMPLIED
            minOccurs         %nonNegativeInteger;   #IMPLIED
            maxOccurs         CDATA                  #IMPLIED
            nillable          %boolean;              #IMPLIED
            substitutionGroup %QName;                #IMPLIED
            abstract          %boolean;              #IMPLIED
            final             %complexDerivationSet; #IMPLIED
            block             %blockSet;             #IMPLIED
            default           CDATA                  #IMPLIED
            fixed             CDATA                  #IMPLIED
            form              %formValues;           #IMPLIED
            %elementAttrs;>
<!-- type and ref are mutually exclusive.
     name and ref are mutually exclusive, one is required -->
<!-- In the absence of type AND ref, type defaults to type of
     substitutionGroup, if any, else the ur-type, i.e. unconstrained -->
<!-- default and fixed are mutually exclusive -->

<!ELEMENT %group; ((%annotation;)?,(%mgs;)?)>
<!ATTLIST %group;
          name        %NCName;               #IMPLIED
          ref         %QName;                #IMPLIED
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %groupAttrs;>

<!ELEMENT %all; ((%annotation;)?, (%element;)*)>
<!ATTLIST %all;
          minOccurs   (1)                    #IMPLIED
          maxOccurs   (1)                    #IMPLIED
          id          ID                     #IMPLIED
          %allAttrs;>

<!ELEMENT %choice; ((%annotation;)?, (%element;| %group;| %cs; | %any;)*)>
<!ATTLIST %choice;
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %choiceAttrs;>

<!ELEMENT %sequence; ((%annotation;)?, (%element;| %group;| %cs; | %any;)*)>
<!ATTLIST %sequence;
          minOccurs   %nonNegativeInteger;   #IMPLIED
          maxOccurs   CDATA                  #IMPLIED
          id          ID                     #IMPLIED
          %sequenceAttrs;>
```

```
<!-- an anonymous grouping in a model, or
     a top-level named group definition, or a reference to same -->

<!-- Note that if order is 'all', group is not allowed inside.
     If order is 'all' THIS group must be alone (or referenced alone) at
     the top level of a content model -->
<!-- If order is 'all', minOccurs==maxOccurs==1 on element/any inside -->
<!-- Should allow minOccurs=0 inside order='all' . . . -->

<!ELEMENT %any; (%annotation;)?>
<!ATTLIST %any;
            namespace       CDATA                   '##any'
            processContents (skip|lax|strict)       'strict'
            minOccurs       %nonNegativeInteger;    '1'
            maxOccurs       CDATA                   '1'
            id              ID                      #IMPLIED
            %anyAttrs;>

<!-- namespace is interpreted as follows:
                  ##any      - - any non-conflicting WFXML at all

                  ##other    - - any non-conflicting WFXML from namespace
other
                                  than targetNamespace

                  ##local    - - any unqualified non-conflicting
WFXML/attribute
                  one or     - - any non-conflicting WFXML from
                  more URI        the listed namespaces
                  references

                  ##targetNamespace ##local may appear in the above list,
                    with the obvious meaning -->

<!ELEMENT %anyAttribute; (%annotation;)?>
<!ATTLIST %anyAttribute;
            namespace       CDATA               '##any'
            processContents (skip|lax|strict)   'strict'
            id              ID                  #IMPLIED
            %anyAttributeAttrs;>
<!-- namespace is interpreted as for 'any' above -->

<!-- simpleType only if no type|ref attribute -->
<!-- ref not allowed at top level, name iff at top level -->
<!ELEMENT %attribute; ((%annotation;)?, (%simpleType;)?)>
<!ATTLIST %attribute;
          name      %NCName;      #IMPLIED
          id        ID            #IMPLIED
          ref       %QName;       #IMPLIED
          type      %QName;       #IMPLIED
          use       (prohibited|optional|required) #IMPLIED
          default   CDATA         #IMPLIED
          fixed     CDATA         #IMPLIED
          form      %formValues;  #IMPLIED
          %attributeAttrs;>
<!-- type and ref are mutually exclusive.
     name and ref are mutually exclusive, one is required -->
<!-- default for use is optional when nested, none otherwise -->
<!-- default and fixed are mutually exclusive -->
<!-- type attr and simpleType content are mutually exclusive -->
```

```
<!-- an attributeGroup is a named collection of attribute decls, or a
     reference thereto -->
<!ELEMENT %attributeGroup; ((%annotation;)?,
                       (%attribute; | %attributeGroup;)*,
                       (%anyAttribute;)?) >
<!ATTLIST %attributeGroup;
                 name       %NCName;       #IMPLIED
                 id         ID             #IMPLIED
                 ref        %QName;        #IMPLIED
                 %attributeGroupAttrs;>

<!-- ref iff no content, no name.  ref iff not top level -->

<!-- better reference mechanisms -->
<!ELEMENT %unique; ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %unique;
          name     %NCName;       #REQUIRED
          id       ID             #IMPLIED
          %uniqueAttrs;>

<!ELEMENT %key;    ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %key;
          name     %NCName;       #REQUIRED
          id       ID             #IMPLIED
          %keyAttrs;>

<!ELEMENT %keyref; ((%annotation;)?, %selector;, (%field;)+)>
<!ATTLIST %keyref;
          name     %NCName;       #REQUIRED
          refer    %QName;        #REQUIRED
          id       ID             #IMPLIED
          %keyrefAttrs;>

<!ELEMENT %selector; ((%annotation;)?)>
<!ATTLIST %selector;
          xpath %XPathExpr; #REQUIRED
          id    ID          #IMPLIED
          %selectorAttrs;>
<!ELEMENT %field; ((%annotation;)?)>
<!ATTLIST %field;
          xpath %XPathExpr; #REQUIRED
          id    ID          #IMPLIED
          %fieldAttrs;>

<!-- Schema combination mechanisms -->
<!ELEMENT %include; (%annotation;)?>
<!ATTLIST %include;
          schemaLocation %URIref; #REQUIRED
          id             ID       #IMPLIED
          %includeAttrs;>

<!ELEMENT %import; (%annotation;)?>
<!ATTLIST %import;
          namespace      %URIref; #IMPLIED
          schemaLocation %URIref; #IMPLIED
          id             ID       #IMPLIED
          %importAttrs;>

<!ELEMENT %redefine; (%annotation; | %simpleType; | %complexType; |
                      %attributeGroup; | %group;)*>
<!ATTLIST %redefine;
```

```
          schemaLocation %URIref; #REQUIRED
          id             ID       #IMPLIED
          %redefineAttrs;>

<!ELEMENT %notation; (%annotation;)?>
<!ATTLIST %notation;
          name        %NCName;    #REQUIRED
          id          ID          #IMPLIED
          public      CDATA       #REQUIRED
          system      %URIref;    #IMPLIED
          %notationAttrs;>

<!-- Annotation is either application information or documentation -->
<!-- By having these here they are available for datatypes as well
     as all the structures elements -->

<!ELEMENT %annotation; (%appinfo; | %documentation;)*>
<!ATTLIST %annotation; %annotationAttrs;>

<!-- User must define annotation elements in internal subset for this
     to work -->
<!ELEMENT %appinfo; ANY>   <!-- too restrictive -->
<!ATTLIST %appinfo;
          source     %URIref;      #IMPLIED
          id         ID         #IMPLIED
          %appinfoAttrs;>
<!ELEMENT %documentation; ANY>   <!-- too restrictive -->
<!ATTLIST %documentation;
          source     %URIref;   #IMPLIED
          id         ID         #IMPLIED
          xml:lang   CDATA      #IMPLIED
          %documentationAttrs;>

<!NOTATION XMLSchemaStructures PUBLIC
           'structures' 'http://www.w3.org/2001/XMLSchema.xsd' >
<!NOTATION XML PUBLIC
           'REC-xml-1998-0210' 'http://www.w3.org/TR/1998/REC-xml-19980210' >
```

The above file has 3 links to external files:
1. datatypes.dtd
2. http://www.w3.org/2001/XMLSchema.xsd
3. http://www.w3.org/TR/1998/REC-xml-19980210

Following the algorithm described in "Resolving References in Distributed Objects" the archive program would attempt to resolve the three references. If at least one of these references were resolved, a new DTD (the normalized version) would be created and the named references would be substituted with the DFIDs plus file extensions of the files. Any unresolved references would be recorded as 'broken links' for this file and would be recorded in the archive database.