XML Catalogs

Committee Specification 1.1, 22 July 2005

Document identifier:

cs-entity-xml-catalogs-1.1

Location:

http://www.oasis-open.org/committees/documents.php?wg_abbrev=entity

Editor:

Norman Walsh, Sun Microsystems, Inc. <Norman.Walsh@Sun.COM>

Abstract:

The requirement that all external identifiers in XML documents must provide a system identifier has unquestionably been of tremendous short-term benefit to the XML community. It has allowed a whole generation of tools to be developed without the added complexity of explicit entity management.

However, the interoperability of XML documents has been impeded in several ways by the lack of entity management facilities:

1. External identifiers may require resources that are not always available. For example, a system identifier that points to a resource on another machine may be inaccessible if a network connection is not available.

2. External identifiers may require protocols that are not accessible to all of the vendors' tools on a single computer system. An external identifier that is addressed with the `ftp:` protocol, for example, is not accessible to a tool that does not support that protocol.

3. It is often convenient to access resources using system identifiers that point to local resources. Exchanging documents that refer to local resources with other systems is problematic at best and impossible at worst.

The problems involved with sharing documents, or packages of documents, across multiple systems are large and complex. While there are many important issues involved and a complete solution is beyond the current scope, the OASIS membership agrees upon the enclosed set of conventions to address a useful subset of the complete problem. To address these issues, this Committee Specification defines an entity catalog that maps both external identifiers and arbitrary URI references to URI references.

Status:

This Committee Specification was approved for publication by the OASIS Entity Resolution Technical Committee. It represents the consensus of the committee.

Please send comments on this specification to the <entity-resolution-comment@lists.oasis-open.org> list. To subscribe, send an email message to <entity-

resolution-comment-request@lists.oasis-open.org> with the word "subscribe" as the body of the message.

Table of Contents

Appendixes

1. Introduction

In order to make optimal use of the information about an XML external resource, there needs to be some interoperable way to map the information in an XML external identifier into a URI reference for the desired resource.

This Committee Specification defines an entity catalog that handles two simple cases:

1. Mapping an external entity's public identifier and/or system identifier to a URI reference.

2. Mapping the URI reference of a resource (a namespace name, stylesheet, image, etc.) to another URI reference.

Though it does not handle all issues that a combination of a complete entity manager and storage manager addresses, it simplifies both the use of multiple products in a great majority of cases and the task of processing documents on different systems.

This entity catalog is designed to be compatible with [TR 9401] catalogs as mandated by the Technical Committee [Requirements].

This Committee Specification provides several schema language descriptions of XML Catalogs in non-normative appendices. The semantics of XML Catalogs are defined normatively by the prose of this specfiication, not by any one of those schemas.

2. Terminology

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, and *optional* in this Committee Specification are to be interpreted as described in [RFC 2119]. Note that for reasons of style, these words are not capitalized in this document.

The terms *URI* and *URI reference* are to be interpreted as described in [RFC 2396].

The term *external identifier* is to be interpreted as defined in Production 75 of [XML]. External identifiers have two parts, an optional public identifier and a system identifier. The terms *public identifier* and *system identifier* in this Committee Specification always refer to the respective part of an external identifier.

Note

All *system identifiers* are URI references, but not all URI references are system identifiers. A system identifer is always logically part of an *external identifier*, even when the *public identifer* is not provided.

The logical *input* to a catalog processor is an external identifier (some combination of public and system identifiers) or a URI reference. The logical *output* of the catalog processor is a URI reference. (This Committee Specification does not attempt to define an API for catalog processors so the logical interfaces and the practical interfaces may differ.)

A *catalog* is a logical structure that contains "mapping" information. A catalog may be physically contained in one or more *catalog entry files*. A *catalog entry file* is a document that contains a set of catalog entries.

3. An Entity Catalog

This Committee Specification defines an application-independent entity catalog that maps external identifiers and URI references to (other) URI references. It also defines a format for catalog entry files in terms of [XML] and [XML Namespaces].

The principal task of a catalog processor is to find entries in the catalog that *match* the input provided and return the associated URI reference as the output. The first such match is always used, and there is no requirement for the catalog processor to search for additional matches.

This catalog is used by an application's entity manager. This Committee Specification does not dictate when an entity manager should access this catalog; for example, an application may attempt other mapping algorithms before or after accessing this catalog.

The catalog is effectively an ordered list of (one or more) catalog entry files. It is up to the application to determine the ordered list of catalog entry files to be used as the logical catalog. (This Committee Specification uses the term "catalog entry file" to refer to one component of a logical catalog even though a catalog entry file can be any kind of storage object or entity including—but not limited to—a table in a database, some object identified by a URI reference, or some dynamically generated set of catalog entries.)

Each entry in the catalog associates a URI reference with information about an external reference that appears in an XML document. For example, the following are possible catalog entries that associate a URI reference with a public identifier:

```
<public publicId="ISO 8879:1986//ENTITIES Added Latin 1//EN"
        uri="iso-lat1.gml"/>
<public publicId="-//USA/AAP//DTD BK-1//EN"
        uri="aapbook.dtd"/>
<public publicId="-//Example, Inc.//DTD Report//EN"
        uri="http://www.example.com/dtds/report.dtd"/>
```

This Committee Specification defines the following catalog entry types: `catalog`, `delegatePublic`, `delegateSystem`, `delegateURI`, `group`, `nextCatalog`, `public`, `rewriteSystem`, `rewriteURI`, `system`, and `systemSuffix`, `uri`, and `uriSuffix`. In order to be conformant with this Committee Specification, an application must implement all of these entry types with the semantics described herein.

The namespace name defined by this Committee Specification is "`urn:oasis:names:tc:entity:xmlns:xml:catalog`". The public identifier for XML Catalogs is "`-//OASIS//DTD XML Catalogs V1.1//EN`".

This Committee Specification reserves all elements and attributes from its namespace for current and future use. In addition, unqualified attributes on elements in its namespace, other than the attributes explicitly described in this Committee Specification, are reserved for future use.

To provide for possible future extension and other applications of this catalog, its format allows for "other information" indicated by elements and attributes from namespaces other than the one defined by this Committee Specification.

4. Using Catalogs

A catalog can be used in two different, independent ways: (1) it can be used to locate the replacement text for an external entity, or (2) it can be used to locate an alternate URI reference for a resource. Although these functions are similar in nature, they are distinct and exercise two different sets of entries in the catalog.

In either case, the following entries in the catalog are interpreted as follows:

1. The `catalog` entry is the root of a catalog entry file. All other entries occur within this `catalog` element.

2. The `group` entry is simply a wrapper on which `prefer` (Section 4.1.1, “The `prefer` attribute”) and `xml:base` (Section 6.1, “Common Attributes”) can occur. It has no other effect on the entries that it contains. When examining entries in the catalog sequentially, the presence of a `group` entry does not effect the order in which they are examined.

3. The `nextCatalog` entry indicates that an entity manager must use the associated URI reference to locate an additional catalog entry file to be processed after the current catalog entry file.

The `nextCatalog` entry can be used to insert a new catalog entry file into the current list of catalog entry files. The `catalog` attribute on a `nextCatalog` entry is used to locate another catalog entry file that is inserted into the catalog entry file list after the current catalog entry file. Multiple `nextCatalog` entries are allowed, and the referenced catalog entry files are inserted into the existing working catalog entry file list in the order in which they occur in the current catalog entry file (document order).

Catalog entry files identified by `nextCatalog` entries will only be examined after all other entries in the current catalog entry file have been considered and none of them provide a match for the current input.

In the discussion that follows, note that catalog resolution semantics are not recursive. Once a matching catalog entry has been found, the value that results from that entry is returned without further examination of the catalog.

4.1. External Identifier Entries

External Identifiers, as defined in [Production 75] of [XML], identify the external subset, entities, and notations of an XML document. They are *not* used to identify other resources such as namespace names, stylesheets, and schema languages other than DTDs; URI entries are used for that purpose.

For the purposes of resolving external identifiers, a catalog-based resolver considers the following entries:

- The `public` entry indicates that an entity manager must use the associated URI reference to locate the replacement text for an entity with the specified *public identifier*.

- The `system` entry indicates that an entity manager must use the associated URI reference to locate the replacement text for an entity with the specified *system identifier*.

- The `rewriteSystem` entry indicates that an entity manager must rewrite the specified *system identifier* by replacing the matching prefix with the associated rewrite prefix. The resulting string must be used to locate the replacement text.

- The `systemSuffix` entry indicates that an entity manager must use the associated URI reference to locate the replacement text for an entity with a *system identifier* that matches the suffix.

- The `delegatePublic` entry indicates that external identifiers with a public identifier that starts with the specified string must be resolved by considering the catalog specified by the associated URI reference.

- The `delegateSystem` entry indicates that external identifiers with a system identifier that starts with the specified string must be resolved by considering the catalog specified by the associated URI reference.

Although system identifiers are assumed to be "URI reference[s]…meant to be dereferenced to obtain input for the XML processor to construct the entity's replacement text", in some circumstances (such as when the document was generated on another system, when the document was generated in another location on the same system, or when some files referenced by system identifiers have moved since the document was generated), the specified system identifiers are not always the best identifiers for the replacement text. For this or other reasons, it may be desirable to prefer the public identifier over the system identifier in determining the entity's replacement text. Therefore, this Committee Specification defines two modes for searching the catalog: "prefer system identifier" mode and "prefer public identifier" mode.

1. If system identifiers are preferred, a system identifier was provided, and there is no matching `system`, `rewriteSystem`, or `systemSuffix` type entry, then the system identifier given in the external identifier is used as the URI reference to locate the entity's replacement text regardless of any public identifier given in the external identifier.

   This Committee Specification does not specify what happens if a preferred system identifier does not identify an accessible storage object; an application may look up the public identifier and/or entity name to find another URI reference, or it may simply report an error. An application should at least have the option of issuing a warning if the system identifier fails in this mode.

2. If public identifiers are preferred, a public identifier was specified, and there is no matching `system`, `rewriteSystem`, or `systemSuffix` type entry, the system identifier given in the external identifier is used as the URI reference to locate the entity's replacement text only if no mapping can be found in the catalog for the public identifier.

4.1.1. The `prefer` attribute

The `prefer` attribute can be used on `catalog` and `group` entry types to indicate, for the enclosed set of catalog entries, if system or public entry matches are preferred.

Each occurrence of a `prefer` attribute specifies the search strategy mode for entries contained within the `catalog` or `group` element on which it occurs. A `public` or `delegatePublic` entry encountered when `prefer` is "`public`" will be considered for possible matching whether or not the external identifier has an explicit system identifier. A `public` or `delegatePublic` entry encountered when `prefer` is "`system`" will be ignored during lookups for which the external identifier has an explicit system identifier. No other entry types are affected by the `prefer` attribute. The initial search strategy in force at the beginning of each catalog entry file depends on the preference as determined by the application.

Note that setting `prefer` to "`public`" does not make matching public entries more significant than matching system entries. It only makes matching public identifiers more significant than the specified system identifier if it does not have a matching entry in the catalog.

There are nine possible combinations for each of the possible settings of `prefer`.

When `prefer="public"`:

	The catalog contains a matching public entry, but not a matching system entry	**The catalog contains a matching system entry, but not a matching public entry**	**The catalog contains both a matching public entry and a matching system entry**
External identifier specifies only a public identifier[a]	Public entry is used	N/A	N/A
External identifier specifies only a system identifier	N/A	System entry is used	N/A
External identifier specifies both public and system identifiers	Public entry is used	System entry is used	System entry is used

[a]This cannot occur in XML except in the case of a notation declaration.

When `prefer="system"`:

	The catalog contains a matching public entry, but not a matching system entry	**The catalog contains a matching system entry, but not a matching public entry**	**The catalog contains both a matching public entry and a matching system entry**
External identifier specifies only a public identifier[a]	Public entry is used	N/A	N/A
External identifier specifies only a system identifier	N/A	System entry is used	N/A
External identifier specifies both public and system identifiers	Public entry *is not* used; the system identifier in the document is used	System entry is used	System entry is used

[a]This cannot occur in XML except in the case of a notation declaration.

An application must provide some way (e.g., a runtime argument, environment variable, preference switch) that allows the user to specify which of these modes to use in the absence of any occurrence of the `prefer` attribute on the `catalog` entry.

When doing a catalog lookup, an entity manager generally uses whatever is available from among the entity declaration's system identifier and public identifier to find catalog entries that match the given information. A match in one catalog entry file will take precedence over any match in a later catalog entry file (and, in fact, the entity manager need not process subsequent catalog entry files once a match has occurred).

4.2. URI Entries

URI references that are *not* part of an external identifier, such as namespace names, stylesheets, included files, graphics, and hypertext references, simply identify other resources. They are resolved using URI entries as described below. The input to a resolver that locates resources is simply the original URI reference.

For the purposes of resolving URI references, a catalog-based resolver considers the following entries:

- The `uri` entry indicates that an entity manager must use the associated URI reference to locate the resource.
- The `rewriteURI` entry indicates that an entity manager must rewrite the specified URI reference by replacing the matching prefix with the associated rewrite prefix. The resulting string must be used to locate the resource.
- The `uriSuffix` entry indicates that an entity manager must use the associated URI reference to locate a resource with an initial URI reference that matches the suffix.
- The `delegateURI` entry indicates that a URI reference that starts with the specified string must be resolved by considering the catalog specified by the associated URI reference.

As when resolving URI references, a match in one catalog entry file will take precedence over any match in a later catalog entry file (and, in fact, the entity manager need not process subsequent catalog entry files once a match has occurred).

4.3. Rewrite Entries

Rewrite entries are provided as a convenience for performing redirection of a whole set of entities with a single catalog entry. Typical uses are website mirroring and dealing with fragment identifiers. Note that in the case of fragment identifiers, rewriting can only be applied to the URI that precedes the fragment identifier. The resolver never sees the fragment identifier part of the URI reference (the # or the characters that follow it).

If the entire website at `http://example.com/` has been mirrored onto your local system in `file:///share/mirrors/example/`, it is likely that you want any system identifier reference to the website to be redirected to your local system.

One way of doing this would be to create a `system` entry for every relevant identifier. If there are many entities on the website, this may be tedious. Instead, a single rewrite entry can be used:

```
<rewriteSystem systemIdStartString="http://www.example.com/"
               rewritePrefix="file:///share/mirrors/example/"/>
```

Similarly, if you have a large number of references to a single document using many different fragment identifiers, it may be tedious to construct `uri` entries for every URI reference if the base document moves. Again, a single rewrite can be used instead:

```
<rewriteURI uriStartString="http://www.example.com/old-location/"
            rewritePrefix="http://www.example.com/new-location/"/>
```

4.4. Suffix Entries

Suffix entries are provided as a mechanism for matching based on the suffix of an identifier. The typical use case is to match a common entity based on its name rather than its full URI.

For example, you can match a DTD such as **xhtml1-strict.dtd** or a schema such as **docbook.rng** regardless of the full system identifier or URI used in the document.

For example, with this entry:

```
<systemSuffix systemIdSuffix="html1-strict.dtd"
              uri="file:///share/mirrors/w3c/xhtml1/xhtml1-strict.dtd"/>
```

any system identifier that ends in "**xhtml1-strict.dtd**" will be translated into the local copy of that DTD. This can be a great convenience if you receive documents from authors with different local configurations.

Similarly, if there is variation in the absolute URI reference used to locate a particular resource, you can select it with just the suffix:

```
<uriSuffix uriSuffix="/uniqueName.xsd"
           uri="file:///share/mirrors/schemas/example/uniqueName.xsd"/>
```

Naturally, the ability to use system identifier or URI suffixes depends on the uniqueness of the suffix as a means of identifying the entity or resource.

4.5. An XML Catalog Example

The catalog files in Example 1, "A DocBook XML Catalog File: **docbook.xml**." and Example 2, "A Stylesheet XML Catalog File: **stylesheet.xml**." are complete examples of XML Catalog files.

Example 1. A DocBook XML Catalog File: docbook.xml.

```
<!DOCTYPE catalog
  PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
         "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">

  <group xml:base="http://www.oasis-open.org/docbook/xml/4.1.2/">
    <public publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
            uri="docbookx.dtd"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Notations V4.1.2//EN"
            uri="dbnotnx.mod"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Character Entities V4.1.2//EN
            uri="dbcentx.mod"/>
    <public publicId="-//OASIS//ELEMENTS DocBook XML Information Pool V4.1.2//EN"
            uri="dbpoolx.mod"/>
    <public publicId="-//OASIS//ELEMENTS DocBook XML Document Hierarchy V4.1.2//EN
            uri="dbhierx.mod"/>
    <public publicId="-//OASIS//ENTITIES DocBook XML Additional General Entities V
            uri="dbgenent.mod"/>
    <public publicId="-//OASIS//DTD DocBook XML CALS Table Model V4.1.2//EN"
            uri="calstblx.dtd"/>
  </group>

  <public publicId="-//OASIS//DTD DocBook MathML Module V1.0//EN"
          uri="http://www.oasis-open.org/docbook/xml/mathml/1.0/dbmathml.dtd"/>

  <nextCatalog catalog="stylesheet.xml"/>

</catalog>
```

Example 2. A Stylesheet XML Catalog File: stylesheet.xml.

```
<!DOCTYPE catalog
  PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
         "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">

  <!-- Circumvent relative URI in spec.xsl that doesn't work online -->
  <uri name="http://www.oasis-open.org/committes/tr.xsl"
       uri="http://www.oasis-open.org/committes/entity/stylesheets/base/tr.xsl"/>
</catalog>
```

Together, these two catalog files provide sufficient resolution information to parse and format the XML source for this Committee Specification.

5. Catalog Entry Files

Applications conforming to this Committee Specification must provide some (implementation dependent) mechanism that allows the user to establish the initial list of catalog entry files. This may be a preferences dialog, an environment variable, an application properties file, or any other appropriate mechanism.

All conforming processors must accept and process catalog entry files written in the format described by this specification. They may also accept and process other formats, but they are not required to do so. If an application encounters a catalog entry file in a format that it does not understand, it must treat it as a resource failure.

5.1. Document Control of Catalog Entry Files

If a document contains external identifiers or URI references, it may be useful for the document to identify a catalog that is likely to aid in the resolution of those references.

For example, XML documents stored on the **www.example.com** server may wish to indicate that **http://www.example.com/catalog** is a useful public catalog to use when parsing them.

This Committee Specification defines the processing instruction "`<?oasis-xml-catalog?>`" for this purpose. The `<?oasis-xml-catalog?>` processing instruction has a single pseudo-attribute, `catalog`, that identifies a single catalog entry file.

If a document contains one or more `<?oasis-xml-catalog?>` processing instruction(s), the catalog entry file(s) identified must be used during resolution of external identifiers and URI references within that document.

Catalog entry files referenced by the processing instruction are added to the end of any system- or user-defined catalog entry file list.

For example, in

```
<?xml version="1.0"?>
<?oasis-xml-catalog catalog="http://example.com/catalog.xml"?>
<!DOCTYPE doc PUBLIC "-//Example//DTD Document V1.0//EN"
                     "http://www.example.com/schema/doc.dtd">
...
```

The URI "**http://example.com/catalog.xml**" is added to the end of the of the list of catalog entry files used for resolution within this document.

The following constraints apply:

- The `<?oasis-xml-catalog?>` processing instruction must appear in the prologue after the XML declaration and before the start of the document type declaration.

  It is an error for the processing instruction to occur in the internal subset or after the document type declaration. Processors should recover from the error by ignoring any such processing instructions that occur after the start of the document type declaration.

  Likewise, it is an error for the processing instruction to occur after other processing instructions that contain URI references, such as stylesheet processing instructions ([XML Stylesheets]). Applications should recover by ignoring catalog entry files mentioned in such `<?oasis-xml-catalog?>` processing instructions.

- If more than one catalog processing instruction is present, each catalog entry file specified is added to the end of the catalog entry file list.

- If the catalog entry file is specified with a relative URI, it is relative to the base URI of the document that contains the processing instruction.

- The URI that identifies the catalog entry file is not subject to catalog resolution.

Catalog-aware applications should support the `<?oasis-xml-catalog?>` processing instruction. If the processing instruction is supported, they must provide a facility which allows a user to request that all `<?oasis-xml-catalog?>` processing instructions be ignored.

One common idiom for controlling parser features is the use of a feature URI. This Committee Specification defines the following feature URI for this purpose:

`http://www.oasis-open.org/committees/entity/features/catalog-pi`

If this feature is disabled, `<?oasis-xml-catalog?>` processing instructions must be ignored.

5.2. "Bootstrapping" Catalog Resolution

XML Catalog files are XML documents and as such may contain external identifiers and URI references. Conformant processors are not required to be able to perform resolution of those identifiers through the XML Catalog.

Implementations are encouraged to provide some sort of bootstrapping functionality to resolve external identifiers and URIs that the implementation needs to load catalog entry files.

For example, presented with the following catalog entry file:

```
<!DOCTYPE catalog
  PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
         "http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">
    <public publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
            uri="docbookx.dtd"/>
</catalog>
```

an implementation should recognize the standard external identifier used on the catalog and provide the parser with access to that DTD in some implementation defined way if it's necessary.

Users can avoid any problems that might arise by limiting the external identifiers and URIs used to those that do not need resolution. Note that this *only* applies to external identifiers and URIs that must be resolved in order to load the catalog entry file.

For example, if a local copy of the XML Catalog DTD is available at **/etc/xml/catalog.dtd**, the problems of resolution associated with loading this file can be avoided by pointing directly to that local copy in the catalog entry file:

```
<!DOCTYPE catalog
  PUBLIC "-//OASIS//DTD XML Catalogs V1.1//EN"
         "/etc/xml/catalog.dtd">
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         prefer="public">
    <public publicId="-//OASIS//DTD DocBook XML V4.1.2//EN"
            uri="docbookx.dtd"/>
</catalog>
```

5.3. Catalog Circularities

As noted, catalog resolution semantics are not recursive. Once a matching catalog entry has been found, the value that results from that entry is returned without further examination of the catalog. In other words, if the catalog contains the following entries and only these entries:

```
<uri name="http://example.com/path/resource"
     uri="http://example.com/alternate/resource"/>

<uri name="http://example.com/alternate/resource"
     uri="http://example.com/final/resource"/>
```

An attempt to resolve `http://example.com/path/resource` will return "`http://example.com/alternate/resource`". The fact that the URI returned would be subject to a different interpretation if *it* was passed to the resolver has no effect on the resolver: it stops when a match is found. (This example uses URI entries, but the semantics hold true for all entry types.)

This avoids an obvious opportunity for circular reference inside the resolver. However, applications are free to make multiple calls to the resolver if they wish, in which case it is the responsibility of the application to handle any circularities that arise.

Even so, catalog circularities may arise. Implementations should detect circularity, but it may be impractical or impossible in some circumstances. If a circularity is detected, it must be treated as an error. Applications may recover from this error by indicating to the calling application that no match was found.

Given the dynamic nature of resources on the internet, it may not always be possible for implementations to detect circular references. Failure to detect circularity of references is not a failure to conform to this specification.

6. XML Catalog Entries

Each catalog entry file consists of some number of catalog entries. Catalog entries are identified by the namespace name defined by this Committee Specification.

Elements and attributes from other namespaces are allowed, but their semantics are not defined by this specification. Implementations that encounter an element from a namespace they do not recognize must ignore it. If an element is ignored, all of its descendants must also be ignored, regardless of their namespace.

6.1. Common Attributes

There are two attributes common to most elements: `id` and `xml:base`. The `id` is provided so that individual entries can be uniquely identified; it has no impact on the semantics of the catalog as defined by this Committee Specification. The `xml:base` attribute changes the base URI for the entry on which it occurs (and all entries contained within it, unless further modified by another `xml:base` attribute). The semantics of `xml:base` are normatively defined in [XML Base].

All of the attributes defined by this Committee Specification are in the per-element-type partition. Use of qualified attributes, for example, `<cat:group cat:id="groupId">` instead of `<cat:group id="groupId">` is forbidden.

6.2. Public Identifier Normalization

In order to accurately and interoperably compare public identifiers, catalog processors must perform normalization on public identifiers in *both* the catalog and the input passed to them.

All strings of white space in public identifiers must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

6.3. System Identifier and URI Normalization

In order to accurately and interoperably compare system identifiers and URI references, catalog processors must perform normalization. The normalization described in this section must be performed on system identifiers and URI references passed as input to the resolver and on strings in the catalog that are compared to them.

URI references require encoding and escaping of certain characters. The disallowed characters include all non-ASCII characters, plus the excluded characters listed in Section 2.4 of [RFC 2396], except for the number sign (#) and percent sign (%) characters and the square bracket characters re-allowed in [RFC 2732]. These characters are summarized in Table 1, “Excluded US-ASCII Characters”.

Table 1. Excluded US-ASCII Characters

Hex Value	Character	Hex Value	Character	Hex Value	Character
00	NUL	0F	SI	1E	RS
01	SOH	10	DLE	1F	US
02	STX	11	DC1	20	(space)
03	ETX	12	DC2	22	"
04	EOT	13	DC3	3C	<
05	ENQ	14	DC4	3E	>
06	ACK	15	NAK	5C	\
07	BEL	16	SYN	5E	^
08	BS	17	ETB	60	`
09	HT	18	CAN	7B	{
0A	LF	19	EM	7C	|
0B	VT	1A	SUB	7D	}
0C	FF	1B	ESC	7F	DEL
0D	CR	1C	FS		
0E	SO	1D	GS		

Catalog processors must escape disallowed characters as follows:

1. Each disallowed character is converted to UTF-8 [RFC 2279] as one or more bytes.

2. Any octets corresponding to a disallowed character are escaped with the URI escaping mechanism (that is, converted to %HH, where HH is the hexadecimal notation of the octet value). If escaping must be performed, uppercase hexadecimal characters should be used.

3. The original character is replaced by the resulting character sequence.

Note that this normalization process is idempotent: repeated normalization does not change a normalized URI reference.

6.4. URN "Unwrapping"

This Committee Specification requires processors to implement special treatment of URNs in the `publicid` URN Namespace ([RFC 3151]).

URNs of this form must, in some contexts, be "unwrapped" by the Catalog processor. This unwrapping translates the URN form of the public identifier back into the standard ISO 8879 form for the purposes of subsequent catalog processing.

Unwrapping a `urn:publicid:` URN is accomplished by transcribing characters in the URN according to the following table after discarding the leading `urn:publicid:` string:

URN Characters	Public Identifier Characters
+	" " (space)
:	//
;	::

URN Characters	Public Identifier Characters
%2B	+
%3A	:
%2F	/
%3B	;
%27	'
%3F	?
%23	#
%25	%

For example, the following URN in the `publicid` namespace:

```
urn:publicid:-:OASIS:DTD+DocBook+XML+V4.1.2:EN
```

Represents the public identifier:

```
-//OASIS//DTD DocBook XML V4.1.2//EN
```

URNs in the `publicid` namespace should always represent normalized public identifiers (Section 6.2, “Public Identifier Normalization”). In the event that an unwrapped public identifier is not normalized, the catalog processor must normalize it.

URNs in the `publicid` namespace are intended for use in documents. Since the resolver is required to unwrap them before searching the catalog, they should never be used literally in the catalog. (Nothing will ever match them.)

6.5. Catalog Elements

The root element of a catalog entry file is `catalog`. There are twelve other element types: `group`, `public`, `system`, `rewriteSystem`, `systemSuffix`, `delegatePublic`, `delegateSystem`, `rewriteURI`, `delegateURI`, `uri`, `uriSuffix`, and `nextCatalog`. Each of these element types is described in one of the following sections.

6.5.1. The `catalog` Entry

Each XML Catalog entry file consists of a single `catalog` element. This element may set the global `prefer` value and global base URI. It is otherwise just a container for the other elements.

```
      <catalog
      id = id
      prefer = "system" | "public"
      xml:base = uri-reference >
  <!-- Content: (group | public | system | rewriteSystem | systemSuffix
      | delegatePublic | delegateSystem | uri | rewriteURI | uriSuffix
      | delegateURI | nextCatalog)+ -->
      </catalog>
```

6.5.2. The `group` Entry

The `group` element is a convenience wrapper for specifying a `prefer` setting or base URI for a set of catalog entries. It has no semantics other than scoping these settings.

```
      <group
      id = id
      prefer = "system" | "public"
      xml:base = uri-reference >
 <!-- Content: (public | system | rewriteSystem | systemSuffix |
      delegatePublic | delegateSystem | uri | rewriteURI | uriSuffix
      | delegateURI | nextCatalog)+ -->
      </group>
```

Note

The ability to scope the `prefer` and and base URI settings is required in order to reasonably translate existing [TR 9401] catalogs into XML Catalogs.

6.5.3. The `public` Entry

The `public` element associates a URI reference with the *public identifier* portion of an *external identifier*.

```
      <public
      id = id
      publicId = public-identifier
      uri = uri-reference
      xml:base = uri-reference />
```

A `public` entry matches a public identifier if the normalized value (Section 6.2, "Public Identifier Normalization") of the public identifier is lexically identical to the normalized value of the `publicId` attribute of the entry.

If the value of the `uri` attribute is relative, it must be made absolute with respect to the base URI currently in effect. The URI reference should not include a fragment identifier.

6.5.4. The `system` Element

The `system` element associates a URI reference with the *system identifier* portion of an *external identifier*.

```
      <system
      id = id
      systemId = string
      uri = uri-reference
      xml:base = uri-reference />
```

A `system` entry matches a system identifier if the normalized value (Section 6.3, "System Identifier and URI Normalization") of the system identifier is lexically identical to the normalized value of the `systemId` attribute of the entry.

If the value of the `uri` attribute is relative, it must be made absolute with respect to the base URI currently in effect. The URI reference should not include a fragment identifier.

6.5.5. The `rewriteSystem` Element

The `rewriteSystem` element rewrites the beginning of a system identifier.

```
        <rewriteSystem
        id = id
        systemIdStartString = string
        rewritePrefix = uri-reference />
```

A `rewriteSystem` entry matches a system identifier if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the system identifier begins precisely with the normalized value of the `systemIdStartString` attribute of the entry.

If the value of the `rewritePrefix` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

Rewriting removes the matching prefix from the supplied system identifier and replaces it with the value of the `rewritePrefix` attribute, returning the entire URI with the new prefix.

If more than one `rewriteSystem` entry matches, the matching entry with the longest normalized `systemIdStartString` value is used.

Given the following catalog fragment:

```
<rewriteSystem systemIdStartString="http://www.oasis-open.org/"
               rewritePrefix="file:///share/doctypes/oasis/"/>
<rewriteSystem systemIdStartString="http://www.oasis-open.org/docbook/"
               rewritePrefix="file:///sourceforge/docbook/docbook/"/>
<rewriteSystem systemIdStartString="http://www.oasis-open.org/committees/"
               rewritePrefix="file:///projects/oasis/"/>
```

The first two entries match the system identifier "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd", but the third does not. The rewritten system identifier in this case is: "file:///sourceforge/docbook/docbook/xml/4.1.2/docbookx.dtd".

6.5.6. The `systemSuffix` Element

The `systemSuffix` element associates a URI reference with the *suffix* portion of a *system identifier*.

```
        <systemSuffix
        id = id
        systemIdSuffix = string
        uri = uri-reference
        xml:base = uri-reference />
```

A `systemSuffix` entry matches a system identifier if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the system identifier ends precisely with the normalized value of the `systemIdSuffix` attribute of the entry.

If the value of the `uri` attribute is relative, it must be made absolute with respect to the base URI currently in effect. The URI reference should not include a fragment identifier.

If more than one `systemSuffix` entry matches, the matching entry with the longest normalized `systemIdSuffix` value is used.

Given the following catalog fragment:

```
<systemSuffix systemIdSuffix="docbookx.dtd"
              uri="file:///share/doctypes/xml/4.4/docbookx.dtd"/>
```

```
<systemSuffix systemIdSuffix="4.3/docbookx.dtd"
              uri="file:///share/doctypes/xml/4.3/docbookx.dtd"/>
```

The first entry matches the system identifier "file:/C|/local/docbookx.dtd" but the second matches "file:/C|/local/backup/4.3/docbookx.dtd" because “`4.3/docbookx.dtd`” is the longer suffix.

6.5.7. The `delegatePublic` Element

The `delegatePublic` element associates an alternate catalog with a partial public identifier.

```
        <delegatePublic
        id = id
        publicIdStartString = public-identifier-prefix
        catalog = uri-reference
        xml:base = uri-reference />
```

A `delegatePublic` entry matches a public identifier if the normalized value (Section 6.2, “Public Identifier Normalization”) of the public identifier begins precisely with the normalized value of the `publicIdStartString` attribute of the entry.

Given the following catalog fragment:

```
<delegatePublic publicIdStartString="-//OASIS//"
                catalog="http://www.oasis-open.org/catalog"/>
<delegatePublic publicIdStartString="-//OASIS//DTD DocBook "
                catalog="http://www.oasis-open.org/docbook/catalog"/>
<delegatePublic publicIdStartString="-//OASIS//DTD XML Catalog //"
                catalog="http://www.oasis-open.org/committees/entity/catalog"/>
```

The first two entries match the public identifier "-//OASIS//DTD DocBook V4.1.2//EN", but the third does not.

If the value of the `catalog` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

6.5.8. The `delegateSystem` Element

The `delegateSystem` element associates an alternate catalog with a partial system identifier.

```
        <delegateSystem
        id = id
        systemIdStartString = string
        catalog = uri-reference
        xml:base = uri-reference />
```

A `delegateSystem` entry matches a system identifier if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the system identifier begins precisely with the normalized value of the `systemIdStartString` attribute of the entry.

Given the following catalog fragment:

```
<delegateSystem systemIdStartString="http://www.oasis-open.org/"
                catalog="http://www.oasis-open.org/catalog"/>
<delegateSystem systemIdStartString="http://www.oasis-open.org/docbook/"
                catalog="http://www.oasis-open.org/docbook/catalog"/>
```

```
<delegatePublic publicIdStartString="http://www.oasis-open.org/committees/"
                catalog="http://www.oasis-open.org/committees/catalog"/>
```

The first two entries match the system identifier "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd", but the third does not.

If the value of the `catalog` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

6.5.9. The `uri` Element

The `uri` element associates an alternate URI reference with a URI reference that is not part of an *external identifier*.

```
        <uri
        id = id
        name = string
        uri = uri-reference
        xml:base = uri-reference />
```

A `uri` entry matches a URI reference if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the URI reference is lexically identical to the normalized value of the `name` attribute of the entry.

Given the following catalog fragment:

```
<uri name="http://www.oasis-open.org/committees/docbook/#membership"
          uri="file:///projects/oasis/docbook/website/#membership"/>
<uri name="http://www.oasis-open.org/committees/docbook/"
          uri="file:///projects/oasis/docbook/website/"/>
```

The second entry matches the URI reference "http://www.oasis-open.org/committees/docbook/", but the first does not.

If the value of the `uri` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

6.5.10. The `rewriteURI` Element

The `rewriteURI` element rewrites the beginning of a URI reference that is not part of an *external identifier*.

```
        <rewriteURI
        id = id
        uriStartString = string
        rewritePrefix = uri-reference />
```

A `rewriteURI` entry matches a URI reference if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the URI reference begins precisely with the normalized value of the `uriStartString` attribute of the entry.

If the value of the `rewritePrefix` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

Rewriting removes the matching prefix from the supplied URI reference and replaces it with the value of the value of the `rewritePrefix` attribute.

If more than one `rewriteURI` entry matches, the matching entry with the longest normalized `uriStartString` value is used.

6.5.11. The `uriSuffix` Element

The `uriSuffix` element associates a URI reference with the *suffix* portion of a URI reference that is not part of an *external identifier*.

```
<uriSuffix
id = id
uriSuffix = string
uri = uri-reference
xml:base = uri-reference />
```

A `uriSuffix` entry matches a URI if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the URI ends precisely with the normalized value of the `uriSuffix` attribute of the entry.

If the value of the `uri` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

If more than one `uriSuffix` entry matches, the matching entry with the longest normalized `uriSuffix` value is used.

6.5.12. The `delegateURI` Element

The `delegateURI` element associates an alternate catalog with a partial URI reference.

```
<delegateURI
id = id
uriStartString = string
catalog = uri-reference
xml:base = uri-reference />
```

A `delegateURI` entry matches a URI reference if the normalized value (Section 6.3, “System Identifier and URI Normalization”) of the URI reference begins precisely with the normalized value of the `uriStartString` attribute of the entry.

If the value of the `catalog` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

6.5.13. The `nextCatalog` Element

The `nextCatalog` elements indicate additional catalog entry file(s) to be considered during the process of resolution.

```
<nextCatalog
id = id
catalog = uri-reference
xml:base = uri-reference />
```

If the value of the `catalog` attribute is relative, it must be made absolute with respect to the base URI currently in effect.

Catalogs loaded due to a `nextCatalog` directive have an initial base URI that is dependent on the location of the loaded catalog entry file. No `xml:base` information is inherited from the originating catalog.

7. Catalog Resolution Semantics

This section describes how catalog resolution is performed.

Resolution begins with a list of catalog entry files and *either* an external identifier *or* a URI reference.

7.1. External Identifier Resolution

This section describes how catalog entries are used to resolve external identifiers.

7.1.1. Input to the Resolver

An external identifier will have at least one and perhaps both of the following:

1. a public identifier
2. a system identifier

Resolvers should respect the system identifiers provided by authors. If the system part of the external identifier is relative, resolution should use that relative URI as the system identifier. If resolution with the relative form fails, it's reasonable for resolvers to try again using the absolute form.

If the public identifier is a URN in the `publicid` namespace ([RFC 3151]), it is converted into another public identifier by "unwrapping" the URN (Section 6.4, “URN "Unwrapping"”). This may be done, for example, so that a URN can be specified as the public identifier and a URL as the system identifier, in the absence of widely deployed URN-resolution facilities.

If the system identifier is a URN in the `publicid` namespace, it is converted into a public identifier by "unwrapping" the URN. In this case, one of the following must apply:

1. No public identifier was provided. Resolution continues as if the public identifier constructed by unwrapping the URN was supplied as the original public identifier and no system identifier was provided.
2. The normalized public identifier provided is lexically identical to the public identifier constructed by unwrapping the URN. Resolution continues as if the system identifier had not been supplied.
3. The normalized public identifier provided is different from the public identifier constructed by unwrapping the URN. This is an error. Applications may recover from this error by discarding the system identifier and proceeding with the original public identifier.

7.1.2. Resolution of External Identifiers

Resolution follows the steps listed below, proceeding to each subsequent step if and only if no other action is indicated.

1. Resolution begins in the first catalog entry file in the current catalog entry file list.

2. If a system identifier is provided, and at least one matching `system` entry exists, the (absolutized) value of the `uri` attribute of the first matching `system` entry is returned.

3. If a system identifier is provided, and at least one matching `rewriteSystem` entry exists, rewriting is performed.

   If more than one `rewriteSystem` entry matches, the matching entry with the longest normalized `systemIdStartString` value is used.

   Rewriting removes the matching prefix and replaces it with the rewrite prefix identified by the matching `rewriteSystem` entry. The rewritten string is returned.

4. If a system identifier is provided, and at least one matching `systemSuffix` entry exists, the (absolutized) value of the `uri` attribute of the matching entry with the longest normalized `systemIdSuffix` value is returned.

5. If a system identifier is provided, and one or more `delegateSystem` entries match, delegation is performed.

   If delegation is to be performed, a new catalog entry file list is generated from the set of all matching `delegateSystem` entries. The (absolutized) value of the `catalog` attribute of each matching `delegateSystem` entry is inserted into the new catalog entry file list such that the delegate entry with the longest matching `systemIdStartString` is first on the list, the entry with the second longest match is second, etc.

   These are the only catalog entry files on the list, the current list is not considered for the purpose of delegation. If delegation fails to find a match, resolution for this entity does not resume with the current list. (A subsequent resolution attempt for a different entity begins with the original list; in other words the catalog entry file list used for delegation is distinct and unrelated to the "normal" catalog entry file list.)

   Catalog resolution restarts using exclusively the catalog entry files in this new list and the given system identifier; any originally given public identifier is ignored during the remainder of the resolution of this external identifier: return to step 1.

6. If a public identifier is provided, and at least one matching `public` entry exists, the (absolutized) value of the `uri` attribute of the first matching `public` entry is returned. If a system identifier is also provided as part of the input to this catalog lookup, only `public` entries that occur where the `prefer` setting is `public` are considered for matching.

7. If a public identifier is provided, and one or more `delegatePublic` entries match, delegation is performed. If a system identifier is also provided as part of the input to this catalog lookup, only `delegatePublic` entries that occur where the `prefer` setting is `public` are considered for matching.

   If delegation is to be performed, a new catalog entry file list is generated from the set of all matching `delegatePublic` entries. The value of the `catalog` attribute of each matching `delegatePublic` entry is inserted into the new catalog entry file list such that the delegate entry with the longest matching `publicIdStartString` is first on the list, the entry with the second longest match is second, etc.

   These are the only catalog entry files on the list, the current list is not considered for the purpose of delegation. If delegation fails to find a match, resolution for this entity does not resume with the current list. (A subsequent resolution attempt for a different entity begins with the original list; in other words

the catalog entry file list used for delegation is distinct and unrelated to the "normal" catalog entry file list.)

Catalog resolution restarts using exclusively the catalog entry files in this new list and the given public identifier; any originally given system identifier is ignored during the remainder of the resolution of this external identifier: return to step 1.

8. If the current catalog entry file contains one or more `nextCatalog` entries, the catalog entry files referenced by each `nextCatalog` entry's "catalog" attribute are inserted, in the order that they appear in this catalog entry file, onto the current catalog entry file list, immediately after the current catalog entry file.

9. If there are one or more catalog entry files remaining on the current catalog entry file list, load the next catalog entry file and continue resolution efforts: return to step 2.

10. Indicate to the calling application that no match was found.

7.2. URI Resolution

This section describes how catalog entries are used to resolve URI references.

7.2.1. Input to the Resolver

URI reference resolution always begins with a single URI reference.

Resolvers should respect the URI references provided by authors. If the URI is relative, resolution should use that relative URI. If resolution with the relative form fails, it's reasonable for resolvers to try again using the absolute form.

If the URI reference is a URN in the `publicid` namespace ([RFC 3151]), it is converted into a public identifier by "unwrapping" the URN (Section 6.4, “URN "Unwrapping"”). Resolution continues by following the semantics of external identifier resolution (Section 7.1, “External Identifier Resolution”) as if the public identifier constructed by unwrapping the URN had been provided and no system identifier had been provided. Otherwise, resolution of the URI reference proceeds according to the steps below.

7.2.2. Resolution of URI references

Resolution of a generic URI reference follows the steps listed below, proceeding to each subsequent step if and only if no other action is indicated.

1. Resolution begins in the first catalog entry file in the current catalog list.

2. If at least one matching `uri` entry exists, the (absolutized) value of the `uri` attribute of the first matching `uri` entry is returned.

3. If at least one matching `rewriteURI` entry exists, rewriting is performed.

   If more than one `rewriteURI` entry matches, the matching entry with the longest normalized `uriStartString` value is used.

   Rewriting removes the matching prefix and replaces it with the rewrite prefix identified by the matching `rewriteURI` entry. The rewritten string is returned.

4. If at least one matching `uriSuffix` entry exists, the (absolutized) value of the `uri` attribute of the matching entry with the longest normalized `uriSuffix` value is returned.

5. If one or more `delegateURI` entries match, delegation is performed.

   If delegation is to be performed, a new catalog entry file list is generated from the set of all matching `delegateURI` entries. The (absolutized) value of the `catalog` attribute of each matching `delegateURI` entry is inserted into the new catalog entry file list such that the delegate entry with the longest matching `uriStartString` is first on the list, the entry with the second longest match is second, etc.

   These are the only catalog entry files on the list, the current list is not considered for the purpose of delegation. If delegation fails to find a match, resolution for this entity does not resume with the current list. (A subsequent resolution attempt for a different entity begins with the original list; in other words the catalog entry file list used for delegation is distinct and unrelated to the "normal" catalog entry file list.)

   Catalog resolution restarts using exclusively the catalog entry files in this new list and the given URI reference: return to step 1.

6. If the current catalog entry file contains one or more `nextCatalog` entries, the catalog entry files referenced by each `nextCatalog` entry's "catalog" attribute are inserted, in the order that they appear in this catalog entry file, onto the current catalog entry file list, immediately after the current catalog entry file.

7. If there are one or more catalog entry files remaining on the current catalog entry file list, load the next catalog entry file and continue resolution efforts: return to step 2.

8. Indicate to the calling application that no match was found.

8. Resource Failures

The catalog processor is sometimes required to load a catalog entry file. This may occur at the beginning of processing, when dealing with the initial list of catalog entry files, or during subsequent processing of a `nextCatalog` entry or one of the delegate entries.

If the processor attempts to load a resource and fails (because the resource does not exist or is not reachable, for example), it must recover by ignoring the catalog entry file that failed and proceeding.

Similarly, if the resource retrieved is not an understandable catalog (because it is not in a format that the processor recognizes, or it purports to be XML but is not well-formed, or for any other reason), the processor must recover by responding as if the resource could not be loaded.

In order for a resource to be considered an XML Catalog, the following conditions must hold:

1. The resource retrieved must be well-formed [XML] consistent with [XML Namespaces].

2. The root element must be `catalog`.

3. The namespace name of the root element must be `urn:oasis:names:tc:entity:xmlns:xml:catalog`.

It is not an error for catalog processors to accept other forms of catalog documents, but their identification and specification is outside the scope of this Committee Specification.

9. Changes in XML Catalogs V1.1

XML Catalogs Working Draft 1.1 differs from the previous *XML Catalogs Committee Specification 1.0* in only one significant way: it introduces `systemSuffix` and `uriSuffix` entries.

By design, XML Catalogs defined by this Committee Specification use the same namespace name as *XML Catalogs Committee Specification 1.0*. Although additional elements have been defined, the semantics of all existing elements remain unchanged.

A. A W3C XML Schema for the XML Catalog (Non-Normative)

This [W3C XML Schema] grammar defines the syntax for OASIS XML Catalog Committee Specification entry files.

This grammar has the following identifier:

- System identifier: `http://www.oasis-open.org/committees/entity/release/1.1/catalog.xsd`

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:er="urn:oasis:names:tc:entity:xmlns:xml:catalog"
           targetNamespace="urn:oasis:names:tc:entity:xmlns:xml:catalog"
           elementFormDefault="qualified">

  <!-- $Id: catalog.xsd,v 1.14 2005/03/31 18:17:02 ndw Exp $ -->

  <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>

  <xs:simpleType name="pubIdChars">
    <!-- A string of the characters defined as pubIdChar in production 13
         of the Second Edition of the XML 1.0 Recommendation. Does not include
         the whitespace characters because they're normalized by XML parsing. -->
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9\-'\(\)+,./:=?;!*#@$_%]*"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="publicIdentifier">
    <xs:restriction base="er:pubIdChars"/>
  </xs:simpleType>

  <xs:simpleType name="partialPublicIdentifier">
    <xs:restriction base="er:pubIdChars"/>
  </xs:simpleType>

  <xs:simpleType name="systemOrPublic">
    <xs:restriction base="xs:string">
      <xs:enumeration value="system"/>
      <xs:enumeration value="public"/>
```

```
  </xs:restriction>
</xs:simpleType>

<!-- The global attribute xml:base is not explicitly declared; -->
<!-- it is allowed by the anyAttribute declarations. -->

<xs:complexType name="catalog">
  <xs:choice minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="er:public"/>
    <xs:element ref="er:system"/>
    <xs:element ref="er:uri"/>
    <xs:element ref="er:rewriteSystem"/>
    <xs:element ref="er:rewriteURI"/>
    <xs:element ref="er:uriSuffix"/>
    <xs:element ref="er:systemSuffix"/>
    <xs:element ref="er:delegatePublic"/>
    <xs:element ref="er:delegateSystem"/>
    <xs:element ref="er:delegateURI"/>
    <xs:element ref="er:nextCatalog"/>
    <xs:element ref="er:group"/>
    <xs:any namespace="##other" processContents="skip"/>
  </xs:choice>
  <xs:attribute ref="id"/>
  <xs:attribute name="prefer" type="er:systemOrPublic"/>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<xs:complexType name="public">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="publicId" type="er:publicIdentifier"
                    use="required"/>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="system">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="systemId" type="xs:string"
                    use="required"/>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="uri">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
```

```
      <xs:attribute name="name" type="xs:anyURI"
                    use="required"/>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="rewriteSystem">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="systemIdStartString"
                    type="xs:string"
                    use="required"/>
      <xs:attribute name="rewritePrefix" type="xs:string" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="rewriteURI">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="uriStartString"
                    type="xs:string"
                    use="required"/>
      <xs:attribute name="rewritePrefix" type="xs:string" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="systemSuffix">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="systemIdSuffix"
                    type="xs:string"
                    use="required"/>
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="uriSuffix">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="uriSuffix"
                    type="xs:string"
                    use="required"/>
```

```
      <xs:attribute name="uri" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="delegatePublic">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="publicIdStartString"
                    type="er:partialPublicIdentifier"
                    use="required"/>
      <xs:attribute name="catalog" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="delegateSystem">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="systemIdStartString"
                    type="xs:string"
                    use="required"/>
      <xs:attribute name="catalog" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="delegateURI">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="uriStartString"
                    type="xs:string"
                    use="required"/>
      <xs:attribute name="catalog" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="nextCatalog">
  <xs:complexContent>
    <xs:restriction base="xs:anyType">
      <xs:attribute name="catalog" type="xs:anyURI" use="required"/>
      <xs:attribute ref="id"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
    </xs:restriction>
  </xs:complexContent>
```

```
  </xs:complexType>

  <xs:complexType name="group">
    <xs:choice minOccurs="1" maxOccurs="unbounded">
      <xs:element ref="er:public"/>
      <xs:element ref="er:system"/>
      <xs:element ref="er:uri"/>
      <xs:element ref="er:rewriteSystem"/>
      <xs:element ref="er:rewriteURI"/>
      <xs:element ref="er:uriSuffix"/>
      <xs:element ref="er:systemSuffix"/>
      <xs:element ref="er:delegatePublic"/>
      <xs:element ref="er:delegateSystem"/>
      <xs:element ref="er:delegateURI"/>
      <xs:element ref="er:nextCatalog"/>
      <xs:any namespace="##other" processContents="skip"/>
    </xs:choice>
    <xs:attribute name="prefer" type="er:systemOrPublic"/>
    <xs:attribute ref="id"/>
    <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:element name="catalog" type="er:catalog"/>
  <xs:element name="public" type="er:public"/>
  <xs:element name="system" type="er:system"/>
  <xs:element name="uri" type="er:uri"/>
  <xs:element name="rewriteSystem" type="er:rewriteSystem"/>
  <xs:element name="rewriteURI" type="er:rewriteURI"/>
  <xs:element name="systemSuffix" type="er:systemSuffix"/>
  <xs:element name="uriSuffix" type="er:uriSuffix"/>
  <xs:element name="delegatePublic" type="er:delegatePublic"/>
  <xs:element name="delegateSystem" type="er:delegateSystem"/>
  <xs:element name="delegateURI" type="er:delegateURI"/>
  <xs:element name="nextCatalog" type="er:nextCatalog"/>
  <xs:element name="group" type="er:group"/>

</xs:schema>
```

B. A RELAX NG Grammar for the XML Catalog (Non-Normative)

This [RELAX NG] grammar defines the syntax for OASIS XML Catalog Committee Specification entry files.

This grammar has the following identifier:

- System identifier: `http://www.oasis-open.org/committees/entity/release/1.1/catalog.rng`

```
<?xml version="1.0"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
```

```
        ns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
        datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- $Id: catalog.rng,v 1.4 2005/02/25 18:54:25 ndw Exp $ -->

<start>
  <choice>
    <ref name="Catalog"/>
  </choice>
</start>

<define name="pubIdChars">
  <data type="string">
    <param name="pattern">[a-zA-Z0-9\-'\(\)+,./:=?;!*#@$_% ]*</param>
  </data>
</define>

<define name="publicIdentifier">
  <ref name="pubIdChars"/>
</define>

<define name="partialPublicIdentifier">
  <ref name="pubIdChars"/>
</define>

<define name="systemOrPublic">
  <choice>
    <value>system</value>
    <value>public</value>
  </choice>
</define>

<define name="uriReference">
  <data type="anyURI"/>
</define>

<define name="OptionalAttributes">
  <optional>
    <attribute name="id">
<data type="ID"/>
    </attribute>
  </optional>
  <zeroOrMore>
    <attribute>
      <anyName>
        <except>
          <nsName ns=""/>
          <nsName/>
        </except>
      </anyName>
    </attribute>
  </zeroOrMore>
</define>
```

```
<define name="PreferAttribute">
  <attribute name="prefer">
    <ref name="systemOrPublic"/>
  </attribute>
</define>

<define name="Catalog">
  <element name="catalog">
    <ref name="OptionalAttributes"/>
    <optional>
      <ref name="PreferAttribute"/>
    </optional>
    <oneOrMore>
      <choice>
        <ref name="Group"/>
        <ref name="Public"/>
        <ref name="System"/>
        <ref name="Uri"/>
        <ref name="RewriteSystem"/>
        <ref name="RewriteURI"/>
        <ref name="SystemSuffix"/>
        <ref name="UriSuffix"/>
        <ref name="DelegatePublic"/>
        <ref name="DelegateSystem"/>
        <ref name="DelegateURI"/>
        <ref name="NextCatalog"/>
        <ref name="AnyOtherElement"/>
      </choice>
    </oneOrMore>
  </element>
</define>

<define name="Group">
  <element name="group">
    <ref name="OptionalAttributes"/>
    <optional>
      <ref name="PreferAttribute"/>
    </optional>
    <oneOrMore>
      <choice>
        <ref name="Public"/>
        <ref name="System"/>
        <ref name="Uri"/>
        <ref name="RewriteSystem"/>
        <ref name="RewriteURI"/>
        <ref name="SystemSuffix"/>
        <ref name="UriSuffix"/>
        <ref name="DelegatePublic"/>
        <ref name="DelegateSystem"/>
        <ref name="DelegateURI"/>
        <ref name="NextCatalog"/>
        <ref name="AnyOtherElement"/>
      </choice>
    </oneOrMore>
```

```
    </element>
  </define>

  <define name="Public">
    <element name="public">
      <attribute name="publicId">
<ref name="publicIdentifier"/>
      </attribute>
      <attribute name="uri">
<ref name="uriReference"/>
      </attribute>
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
  </define>

  <define name="System">
    <element name="system">
      <attribute name="systemId"/>
      <attribute name="uri">
<ref name="uriReference"/>
      </attribute>
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
  </define>

  <define name="Uri">
    <element name="uri">
      <attribute name="name"/>
      <attribute name="uri">
<ref name="uriReference"/>
      </attribute>
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
  </define>

  <define name="RewriteSystem">
    <element name="rewriteSystem">
      <attribute name="systemIdStartString"/>
      <attribute name="rewritePrefix"/>
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
  </define>

  <define name="RewriteURI">
    <element name="rewriteURI">
      <attribute name="uriStartString"/>
      <attribute name="rewritePrefix"/>
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
```

```
 </define>

 <define name="SystemSuffix">
   <element name="systemSuffix">
     <attribute name="systemIdSuffix"/>
     <attribute name="uri">
<ref name="uriReference"/>
     </attribute>
     <ref name="OptionalAttributes"/>
     <empty/>
   </element>
 </define>

 <define name="UriSuffix">
   <element name="uriSuffix">
     <attribute name="uriSuffix"/>
     <attribute name="uri">
<ref name="uriReference"/>
     </attribute>
     <ref name="OptionalAttributes"/>
     <empty/>
   </element>
 </define>

 <define name="DelegatePublic">
   <element name="delegatePublic">
     <attribute name="publicIdStartString"/>
     <attribute name="catalog"/>
     <ref name="OptionalAttributes"/>
     <empty/>
   </element>
 </define>

 <define name="DelegateSystem">
   <element name="delegateSystem">
     <attribute name="systemIdStartString"/>
     <attribute name="catalog"/>
     <ref name="OptionalAttributes"/>
     <empty/>
   </element>
 </define>

 <define name="DelegateURI">
   <element name="delegateURI">
     <attribute name="uriStartString"/>
     <attribute name="catalog"/>
     <ref name="OptionalAttributes"/>
     <empty/>
   </element>
 </define>

 <define name="NextCatalog">
   <element name="nextCatalog">
     <attribute name="catalog"/>
```

```
      <ref name="OptionalAttributes"/>
      <empty/>
    </element>
  </define>

  <define name="AnyOtherElement">
    <element>
      <anyName>
 <except>
   <nsName ns=""/>
   <nsName/>
 </except>
      </anyName>
      <zeroOrMore>
 <attribute>
   <anyName/>
 </attribute>
      </zeroOrMore>
      <choice>
 <text/>
 <ref name="AnyOtherElement"/>
      </choice>
    </element>
  </define>
</grammar>
```

C. A DTD for the XML Catalog (Non-Normative)

This [XML] DTD grammar partially[1] defines the syntax for OASIS XML Catalog Committee Specification entry files.

This DTD has the following identifiers:

- Public identifier: `-//OASIS//DTD XML Catalogs V1.1//EN`
- System identifier: `http://www.oasis-open.org/committees/entity/release/1.1/catalog.dtd`

```
<!-- $Id: catalog.dtd,v 1.14 2005/04/13 20:47:06 ndw Exp $ -->

<!ENTITY % pubIdChars "CDATA">
<!ENTITY % publicIdentifier "%pubIdChars;">
<!ENTITY % partialPublicIdentifier "%pubIdChars;">
<!ENTITY % uriReference "CDATA">
<!ENTITY % string "CDATA">
<!ENTITY % systemOrPublic "(system|public)">

<!ENTITY % p "">
<!ENTITY % s "">
```

[1] Any catalog file which is valid according to this DTD is valid according to this Committee Specification. However, catalog files which make use of extension elements or attributes may be valid according to this Committee Specification but invalid according to this DTD, due to the limits of DTD validation with respect to namespaces.

```
<!ENTITY % nsdecl "xmlns%s;">

<!ENTITY % catalog "%p;catalog">
<!ENTITY % public "%p;public">
<!ENTITY % system "%p;system">
<!ENTITY % uri "%p;uri">
<!ENTITY % rewriteSystem "%p;rewriteSystem">
<!ENTITY % rewriteURI "%p;rewriteURI">
<!ENTITY % systemSuffix "%p;systemSuffix">
<!ENTITY % uriSuffix "%p;uriSuffix">
<!ENTITY % delegatePublic "%p;delegatePublic">
<!ENTITY % delegateSystem "%p;delegateSystem">
<!ENTITY % delegateURI "%p;delegateURI">
<!ENTITY % nextCatalog "%p;nextCatalog">
<!ENTITY % group "%p;group">

<!ENTITY % local.catalog.mix "">
<!ENTITY % local.catalog.attribs "">

<!ELEMENT %catalog; (%public;|%system;|%uri;
                     |%rewriteSystem;|%rewriteURI;
       |%systemSuffix;|%uriSuffix;
                     |%delegatePublic;|%delegateSystem;|%delegateURI;
                     |%nextCatalog;|%group; %local.catalog.mix;)+>
<!ATTLIST %catalog;
 %nsdecl; %uriReference;  #FIXED
  'urn:oasis:names:tc:entity:xmlns:xml:catalog'
 id  ID   #IMPLIED
 prefer  %systemOrPublic; #IMPLIED
 xml:base %uriReference;  #IMPLIED
 %local.catalog.attribs;
>

<!ENTITY % local.public.attribs "">

<!ELEMENT %public; EMPTY>
<!ATTLIST %public;
 id  ID   #IMPLIED
 publicId %publicIdentifier; #REQUIRED
 uri  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
       %local.public.attribs;
>

<!ENTITY % local.system.attribs "">

<!ELEMENT %system; EMPTY>
<!ATTLIST %system;
 id  ID   #IMPLIED
 systemId %string;  #REQUIRED
 uri  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
       %local.system.attribs;
>
```

```
<!ENTITY % local.uri.attribs "">

<!ELEMENT %uri; EMPTY>
<!ATTLIST %uri;
 id  ID   #IMPLIED
 name  %string;  #REQUIRED
 uri  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
        %local.uri.attribs;
>

<!ENTITY % local.rewriteSystem.attribs "">

<!ELEMENT %rewriteSystem; EMPTY>
<!ATTLIST %rewriteSystem;
 id  ID   #IMPLIED
 systemIdStartString %string; #REQUIRED
 rewritePrefix  %string;  #REQUIRED
        %local.rewriteSystem.attribs;
>

<!ENTITY % local.rewriteURI.attribs "">

<!ELEMENT %rewriteURI; EMPTY>
<!ATTLIST %rewriteURI;
 id  ID   #IMPLIED
 uriStartString %string;  #REQUIRED
 rewritePrefix %string;  #REQUIRED
        %local.rewriteURI.attribs;
>

<!ENTITY % local.systemSuffix.attribs "">

<!ELEMENT %systemSuffix; EMPTY>
<!ATTLIST %systemSuffix;
 id  ID   #IMPLIED
 systemIdSuffix  %string; #REQUIRED
 uri   %string; #REQUIRED
        %local.systemSuffix.attribs;
>

<!ENTITY % local.uriSuffix.attribs "">

<!ELEMENT %uriSuffix; EMPTY>
<!ATTLIST %uriSuffix;
 id  ID   #IMPLIED
 uriSuffix  %string; #REQUIRED
 uri   %string; #REQUIRED
        %local.uriSuffix.attribs;
>

<!ENTITY % local.delegatePublic.attribs "">
```

```
<!ELEMENT %delegatePublic; EMPTY>
<!ATTLIST %delegatePublic;
 id  ID   #IMPLIED
 publicIdStartString %partialPublicIdentifier; #REQUIRED
 catalog  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
        %local.delegatePublic.attribs;
>

<!ENTITY % local.delegateSystem.attribs "">

<!ELEMENT %delegateSystem; EMPTY>
<!ATTLIST %delegateSystem;
 id  ID   #IMPLIED
 systemIdStartString %string; #REQUIRED
 catalog  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
        %local.delegateSystem.attribs;
>

<!ENTITY % local.delegateURI.attribs "">

<!ELEMENT %delegateURI; EMPTY>
<!ATTLIST %delegateURI;
 id  ID   #IMPLIED
 uriStartString %string;  #REQUIRED
 catalog  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
        %local.delegateURI.attribs;
>

<!ENTITY % local.nextCatalog.attribs "">

<!ELEMENT %nextCatalog; EMPTY>
<!ATTLIST %nextCatalog;
 id  ID   #IMPLIED
 catalog  %uriReference;  #REQUIRED
 xml:base %uriReference;  #IMPLIED
        %local.nextCatalog.attribs;
>

<!ENTITY % local.group.mix "">
<!ENTITY % local.group.attribs "">

<!ELEMENT %group; (%public;|%system;|%uri;
                   |%rewriteSystem;|%rewriteURI;
     |%systemSuffix;|%uriSuffix;
                   |%delegatePublic;|%delegateSystem;|%delegateURI;
                   |%nextCatalog; %local.group.mix;)+>
<!ATTLIST %group;
 id  ID   #IMPLIED
 prefer  %systemOrPublic; #IMPLIED
 xml:base %uriReference;  #IMPLIED
```

```
        %local.group.attribs;
>
```

D. Support for TR9401 Catalog Semantics (Non-Normative)

This Committee Specification defines a subset of the catalog entry types described in [TR 9401] that are applicable to XML. For implementors wishing to provide full TR9401 support, this appendix defines the elements that should be used for the remaining TR9401 catalog entry types.

The elements described in this appendix provide full TR9401 semantics in the XML Catalog format. These are implemented as extension elements in the namespace: "`urn:oasis:names:tc:entity:xmlns:tr9401:catalog`".

For a complete description of the semantics of these elements see [TR 9401].

1. The `doctype` Element

The `doctype` element associates a DTD with a document element.

```
        <doctype
        id = id
        name = name
        uri = uri-reference
        xml:base = uri-reference />
```

2. The `document` Element

The `document` element identifies a default document.

```
        <document
        id = id
        uri = uri-reference
        xml:base = uri-reference />
```

3. The `dtddecl` Element

The `dtddecl` element associates an SGML declaration with a public identifier.

```
        <dtddecl
        id = id
        publicId = public-identifier
        uri = uri-reference
        xml:base = uri-reference />
```

4. The `entity` Element

The `entity` element associates a document with an entity name.

```
<entity
id = id
name = name
uri = uri-reference
xml:base = uri-reference />
```

5. The `linktype` Element

The `linktype` element associates an external subset with a linktype declaration name.

```
<linktype
id = id
name = name
uri = uri-reference
xml:base = uri-reference />
```

6. The `notation` Element

The `notation` element associates a URI reference with a notation name.

```
<notation
id = id
name = name
uri = uri-reference
xml:base = uri-reference />
```

7. The `sgmldecl` Element

The `sgmldecl` element provides the location of a default SGML declaration.

```
<sgmldecl
id = id
uri = uri-reference
xml:base = uri-reference />
```

E. OASIS Entity Resolution Committee (Non-Normative)

The following individuals are members of the committee that developed this Committee Specification:

Adam Di Carlo, Debian
Anthony Coates, Individual Member
Paul Grosso, Arbortext
Mark Johnson, Debian
Jirka Kosek, Individual Member
Craig Salter, IBM
Norman Walsh, Sun Microsystems (Editor)
Lauren Wood, Individual Member (Chair)

The following additional individuals were members of the committee during the development of previous versions:

John Cowan, Reuters Health
David Leland, Elsevier Science London
Normand Montour, IBM

F. Notices

G. Intellectual Property Rights

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Entity Resolution web page (http://www.oasis-open.org/committees/entity/)

References

Normative

[XML] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, Eve Maler, editors. *Extensible Markup Language (XML) 1.0 Second Edition*. World Wide Web Consortium, 2000.

[XML Namespaces] Tim Bray, Dave Hollander, and Andrew Layman, editors. *Namespaces in XML*. World Wide Web Consortium, 1999.

[RFC 2119] IETF (Internet Engineering Task Force). *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. S. Bradner. 1997.

[RFC 3151] IETF (Internet Engineering Task Force). *RFC 3151: A URN Namespace for Public Identifiers*. N. Walsh, J. Cowan, P. Grosso, 2001.

[XML Base] Jonathan Marsh, editor. *XML Base*. World Wide Web Consortium, 2000.

Non-Normative

[XML Schema Datatypes] Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. World Wide Web Consortium, 2000.

[RELAX NG] James Clark and MURATA Makoto, editors. *RELAX NG Specification*. OASIS. 2001.

[XML Stylesheets] James Clark, editor. *Associating Style Sheets with XML Documents Version 1.0*. World Wide Web Consortium. 1999.

[TR 9401] Paul Grosso, editor. *OASIS Technical Resolution 9401:1997 (Amendment 2 to TR 9401)*. OASIS. 1997.

[RFC 2279] IETF (Internet Engineering Task Force). *RFC 2279: UTF-8, a transformation format of ISO 10646*. F. Yergeau. 1998.

[RFC 2396] IETF (Internet Engineering Task Force). *RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax*. T. Berners-Lee, R. Fielding, L. Masinter. 1998.

[RFC 2732] IETF (Internet Engineering Task Force). *RFC 2732: Format for Literal IPv6 Addresses in URL's*. R. Hinden, B. Carpenter, L. Masinter. 1999.

[W3C XML Schema] Henry S. Thompson, David Beech, Murray Maloney, et al. editors. *XML Schema Part 1: Structures*. World Wide Web Consortium, 2000.

[Requirements] Norman Walsh, editor. *OASIS Entity Resolution Technical Committee Requirements Document*. OASIS. 2000.