Entity resolution with catalogs (2.1 legacy document)

Table of contents

1 Comments.....9

Table of contents

1 Introduction....3
2 Overview....3
3 Background....3
4 Demonstration #1....4
5 Catalogs overview....4
5.1 External entity declarations....4
5.2 Simple example catalog....4
6 Demonstration #2....5
7 Default configuration....6
8 Local configuration....6
8.1 Using cocoon.xconf....6
8.2 Using CatalogManager.properties....6
8.3 Resolver directives inside your catalog file....7
8.4 Example local configuration for Simplified DocBook....7
9 Implementation notes....7
10 Debugging the resolver configuration....8
11 Development notes....8
12 Other notes....8
13 Summary....8
14 Further information....8

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Introduction

Apache Cocoon has the capability to utilise an entity resolution mechanism. External entities (e.g. Document Type Definitions (DTDs), character entity sets, XML sub-documents) are resources that are declared by an XML instance document - they exist as separate objects. An entity catalog assists with entity management and the resolution of entities to accessible resources. It also reduces the necessity for expensive and failure-prone network retrieval of the required resources.

2. Overview

"Entities" represent the physical structure of an XML instance document, whereas "elements" represent the logical structure. The complete entity structure of the document defines which pieces need to be incorporated, so as to build the final document. Those entities are objects from some accessible place, e.g. local file system, local network, remote network, generated from a database. Example entities are: DTDs, XML sub-documents, sets of character entities to represent symbols and other glyphs, image files.

So how are you going to define the accessible location of all those pieces? How will you ensure that those resources are reliably available? Entity resolution catalogs to the rescue. These are simple standards-based plain-text files to map public identifiers and system identifiers to local or other resources.

Do you wonder why we cannot use the sitemap to resolve these resources? This is because the resolution of all entities that compose the XML document is under the direct control of the guts of the parser and the XML structure. The parser has no choice - it must incorporate all of the defined pieces. If it cannot retrieve them, then it is broken and reports an error.

With the powerful catalog support there are no such problems. This document provides the following sections to explain Cocoon capability for resolving entities ...

- Background - explains the need, explains some terminology, describes the solution
- Demonstration #1 - explains a remote resource and how it gets resolved
- Catalogs overview - briefly explains how catalogs resolve entity declarations
- Demonstration #2 - explains more detailed need and use of catalogs and shows catalogs in action
- Default configuration - explains the default automated configuration
- Local configuration - explains how to extend the default configuration for your local system requirements and provides an example
- Implementation notes - describes how support for catalogs is added to Cocoon
- Development notes - some minor issues need to be addressed
- Other notes - assorted dot-points
- Summary
- Further information - links to some useful resources

3. Background

The following article eloquently describes the need for all parsers and XML frameworks to be capable of utilising entity resolvers. "XML Entity and URI Resolvers" by Norman Walsh. Please read that document, then return here to apply entity catalogs to Cocoon.

(Note: The Apache XML Commons project provides the Java package that has been added to Cocoon as the lib/core/xml-commons-resolver.jar package. There are also API javadocs for resolver that have further information. However, you do not need to know the gory details to understand catalogs and configure them.)

4. Demonstration #1

This snippet from an XML instance shows the Document Type Declaration. Notice that it declares its ruleset, the Document Type Definition (DTD), as an external entity. Notice also that the resource is network-based.

```
<?xml version="1.0"?> <!DOCTYPE article PUBLIC "-//OASIS//DTD Simplified DocBook XML V4.1.2.5//EN" "http://www.oasis-open.org/docbook/xml/simple/4.1.2.5/sdocbook.dtd" <article> ... content goes here </article>
```

Now consider what will happen when Cocoon tries to process this XML instance. Whether you have set validation=yes or not, the parser will still want to resolve all of the entities that are required by the XML instance (i.e. the DTD and any other entities that the DTD might declare). So it will happily trundle across the network to get them. It will do this every time that the document is processed. This is obviously a needless overhead. Worse still, what happens if that host is down or the network is congested. Additionally, if your Cocoon is an off-line server then it is always broken because it cannot retrieve the network-based resources.

5. Catalogs overview

As the Walsh document explained, the secrets to entity resolution are the public identifiers, system identifiers, and the catalog to map between them. Here we provide an overview and show an example catalog which we will then use with the Demonstration #2 below.

5.1. External entity declarations

To define an external entity in an XML instance document, you must provide an external declaration consisting of at least a **system identifier** and optionally a **public identifier**. The system identifier defines the physical location of the external entity. The public identifier is a unique symbolic name that can be used to map to a certain physical location. Note that if you provide both a public and a system identifier, then the public identifier is listed first and the system identifier is not preceded by the keyword SYSTEM. Here are four separate examples ...

```
<!ENTITY pic SYSTEM "images/pic.gif" NDATA gif> <!ENTITY % ISOnum PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML" "ISOnum.pen"> <!DOCTYPE document SYSTEM "dtd/document-v10.dtd"> <!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1//EN" "http://www.oasis-open.org/docbook/xml/4.1/docbookx.dtd">
```

(In your XML instance document, or DTD, you would include those entities like this ... %ISOnum;)

None of those system identifiers looks reliable or easily managed. Use a catalog to make them so.

5.2. Simple example catalog

The catalog maps public identifiers to their corresponding physical locations. The catalog entries in an OASIS catalog are a simple whitespace-delimited format. (The specification fully defines the format.) There about a dozen different types of catalog entry - two important ones are:

- **PUBLIC** publicId systemId

- maps the public identifier publicId to the system identifier systemId
- **SYSTEM** systemId otherSystemId
  - maps the system identifier systemId to the alternate system identifier otherSystemId

-- this is the default OASIS catalog for Apache Cocoon -- OVERRIDE YES -- ISO public identifiers for sets of character entities -- PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN//XML" "ISOlat1.pen" PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN//XML" "ISOlat1.pen" PUBLIC "ISO 9573-15:1993//ENTITIES Greek Letters//EN//XML" "ISOgrk1.pen" PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN//XML" "ISOpub.pen" PUBLIC "ISO 8879:1986//ENTITIES General Technical//EN//XML" "ISOtech.pen" PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML" "ISOnum.pen" -- Document Type Definitions -- PUBLIC "-//APACHE//DTD Documentation V1.0//EN" "document-v10.dtd" PUBLIC "-//APACHE//DTD FAQ V1.0//EN" "faq-v10.dtd" -- (other declarations removed for brevity) -- -- these entries are used for the catalog-demo sample application -- OVERRIDE NO PUBLIC "-//Arbortext//TEXT Test Override//EN" "catalog-demo/override.txt" OVERRIDE YES PUBLIC "-//Arbortext//TEXT Test Public Identifier//EN" "catalog-demo/testpub.txt" SYSTEM "urn:x-arbortext:test-system-identifier" "catalog-demo/testsys.txt" PUBLIC "-//Indexgeo//DTD Catalog Demo v1.0//EN" "catalog-demo/catalog-demo-v10.dtd" -- end of entries for the catalog-demo sample application --

System identifiers can use full pathnames, filenames, relative pathnames, or URLs - in fact, any method that will define and deliver the actual physical entity. If it is just a filename or a relative pathname, then the Catalog Resolver will look for the resource relative to the location of the catalog.

When the parser needs to load a declared entity, then it first consults the Catalog Resolver to get a possible mapping to an alternate system identifier. If there is no mapping for an identifier in the catalogs (or in any sub-ordinate catalogs), then Cocoon will carry on to retrieve the resource using the original declared system identifier.

6. Demonstration #2

See catalogs in action with the Cocoon Samples The demonstration intends to be self-documenting. The top-level XML instance describes its role, and each included external entity reports how it came into being. This example builds upon the example provided by the Walsh article. (Tip: To see the error message that would result from not using a catalog, simply rename the default catalog file before starting Cocoon.)

Here is the source for the top-level XML instance document catalog-demo.xml ...

<?xml version="1.0"?> <!DOCTYPE catalog-demo PUBLIC "-//Indexgeo//DTD Catalog Demo v1.0//EN" "http://www.indexgeo.com.au/dtd/catalog-demo-v10.dtd" [<!ENTITY testpub PUBLIC "-//Arbortext//TEXT Test Public Identifier//EN" "bogus-system-identifier.xml"> <!ENTITY testsys SYSTEM "urn:x-arbortext:test-system-identifier"> <!ENTITY testovr PUBLIC "-//Arbortext//TEXT Test Override//EN" "testovr.txt"> <!ENTITY % ISOnum PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML" "ISOnum.pen"> %ISOnum; <!ENTITY note "Note:">]> <catalog-demo> <section> <para>This sample application demonstrates the use of catalogs for entity resolution. &note; see the Apache Cocoon documentation <link href="../../docs/userdocs/concepts/catalog.html">Entity resolution with catalogs</link> for the full background and explanation, and the XML source of this document (catalog-demo.xml). </para> <para>This top-level XML instance document is catalog-demo.xml - it declares three other XML sub-documents as external entities and then includes them in the sections below. The real system identifiers will be looked up in the

catalog, to resolve the actual location of the resource. </para> <para>The Document Type Definition (DTD) is declared using both a public identifier and a system identifier. The system identifier for the DTD is a network-based resource (which is deliberately non-existent). However, the catalog overrides that remote DTD to instead use a copy from the local filesystem at the location defined by the catalog entry. Note that it is via the use of a public identifier that we gain this power. </para> <para>The internal DTD subset of the top-level document instance goes on to declare the three external sub-document entities using various means. It also declares and includes the ISOnum set of character entities, so that we can use entities like "½" (to represent ½). Finally the internal DTD subset declares an internal general entity for "&note;". </para> </section> <section> <para>testpub ... this entity is declared with a PUBLIC identifier and a bogus system identifier (which will be overridden by the catalog) </para> <para>&note; &testpub;</para> </section> <section> <para>testsys ... this entity is declared with a SYSTEM identifier (which will be resolved by the catalog) </para> <para>&note; &testsys;</para> </section> <section> <para>testovr ... is declared with a PUBLIC identifier and a system identifier (the catalog is set to not override this one, so the declared system identifier is used) </para> <para>&note; &testovr;</para> </section> </catalog-demo>

Here is the source for one of the included sub-document external entities testpub.txt (a slab of plain text) ...

This paragraph is automatically included from the testpub.txt external file. The entity declaration deliberately used a non-existent file as the system identifier. The catalog then used the declared public identifer to resolve to a specific location on the local filesystem.

7. Default configuration

A default catalog and some base entities (e.g. ISO*.pen character entity sets) are included in the Cocoon distribution at WEB-INF/entities/ - the default catalog is automatically loaded when Cocoon starts.

If you suspect problems, then you can raise the level of the verbosity property (to 2 or 3) and watch the messages going to standard output when Cocoon starts and operates. You would also do this to detect any misconfiguration of your own catalogs.

8. Local configuration

You can extend the default configuration to include local catalogs for site-specific requirements. This is achieved via various means.

8.1. Using cocoon.xconf

Parameters (properties) for the resolver component can be specified in the src/webapp/WEB-INF/cocoon.xconf configuration file. See the detailed internal notes - here is a precis.

- catalog ... The main catalog file. Its path name is relative to the Cocoon context directory.
- local-catalog ... The full filesystem pathname to a single local catalog file.
- verbosity ... The level of messages from the resolver (loading catalogs, identifier resolution, etc.). It value may range from 0 (no messages), to 10 detailed log messages.

8.2. Using CatalogManager.properties

An annotated CatalogManager.properties file is included with the distribution - modify it to suit your needs. You can add your own local catalogs using the catalogs property. (See the notes inside the properties file).

The file is at webapp/WEB-INF/classes/CatalogManager.properties thereby making it available to the Java classpath during startup of the servlet engine.

If you see an error message going to STDOUT when Cocoon starts (Cannot find CatalogManager.properties) then this means that the properties file is not available to the Java classpath. Note that this does not mean that entity resolution is disabled, rather that no local configuration is being effected. Therefore no local catalogs will be loaded and no entity resolution messages will be received (verbosity level is zero by default).

That may truly be the intention, and not just a configuration mistake. You can still use cocoon.xconf to effect your local configuration.

8.3. Resolver directives inside your catalog file

The actual "catalog" files have a powerful set of directives. For example, the **CATALOG** directive facilitates the inclusion of a sub-ordinate catalog. The list of resources below will lead to further information about catalog usage.

8.4. Example local configuration for Simplified DocBook

We use the Simplified DocBook XML DTD for some of our documentation. Here are the few steps that we followed to configure Cocoon to be able to process our XML instances.

- Downloaded a recent copy of the Simplified DocBook DTD (the flattened DTD will suffice) from here and place it at /usr/local/sgml/docbook/simple/sdocbook.dtd
- Created a catalog file at /usr/local/sgml/docbook/simple/sdocbook.cat with a single entry for the Simplified DocBook XML DTD
- Added the parameter (local-catalog) to the WEB-INF/cocoon.xconf (using the full pathname to the sdocbook.cat catalog).

-- Catalog file (sdocbook.cat) for Simplified DocBook -- -- See www.oasis-open.org/docbook/ -- -- Driver file for the Simplified DocBook XML DTD -- PUBLIC "-//OASIS//DTD Simplified DocBook XML V4.1.2.5//EN" "sdocbook.dtd" -- end of catalog file for Simplified DocBook --

We could similarly configure Cocoon for the full DocBook XML DTD and related entities. In fact, the DocBook distribution already contains a catalog file. We need only append the pathname to our catalogs property.

There are a few important starting points for further information about using and configuring the DocBook DTDs.

9. Implementation notes

The SAX Parser interface provides an entityResolver hook to allow an application to resolve the external entities. This is enabled via org.apache.excalibur.xml.EntityResolver

The Apache XML Commons project has org.apache.xml.resolver which provides a **CatalogResolver**. This is incorporated into Cocoon via org.apache.excalibur.xml.EntityResolver

Default configuration is achieved via org.apache.cocoon.components.resolver.DefaultResolver.java which initialises the catalog resolver and loads a default system catalog. The DefaultResolver.java

enables local configuration by applying properties from the CatalogManager.properties file and then further configuration from WEB-INF/cocoon.xconf parameters.

10. Debugging the resolver configuration

Raise the verbosity level as described in cocoon.xconf and watch the messages that go to standard output.

The "Resolved public" messages should show that the Public Identifiers are being used to find the local copies of the resources. If a "Resolved public" message does not occur for a particular resource, then Cocoon will be retrieving it from the specified location. Use a packet watching tool like "ngrep" to see the HTTP request for the resource.

11. Development notes

- Keep up-to-date with releases of XML Character Entities by OASIS.

12. Other notes

- OASIS Catalogs (TR 9401:1995 Entity Management) are plain-text files with a simple delimited format. There is also a new standard being developed for XML Catalogs, using an xml-based structured plain-text file (gee :-). Links to both standards are provided below. Both catalog formats can be currently used with this entity resolver. However, the latter standard is not yet settled. OASIS TR9401 catalogs will suffice.
- There has been a recent flood of XML tools - unfortunately, many do not implement entity resolution (other than by brute-force retrieval), so those tools are crippled and cannot be used for serious XML processing. Please ensure that you choose proper XML tools for the preparation and validation of your XML instance documents.
- The default catalog that is shipped with the Cocoon distribution is deliberately basic. You will need to supplement it with your own catalog devised to suit your particular needs.

13. Summary

Most XML documents that we would want to serve with Cocoon are already in existence in another information system. The XML document instances have a declaration of their DTD Document Type Definition as an external file. This external DTD also includes entity sets such as ISOnum, ISOlat1, etc. Also the DTD declaration has a Formal Public Identifier and a System Identifier which points to a remote URL. These XML instance documents cannot be altered to make workaround solutions like ../dtd/document-1.0.dtd

Entity management is effected by providing a standards-based mechanism to resolve public identifiers and system identifiers to local filenames or other identifiers or even to other remote network resources. So references to external DTDs, sets of character entities such as mathematical symbols, fragments of XML documents, complete sub-documents, non-xml data chunks (like images), etc. can all be centrally managed and resolved locally.

14. Further information

Here are some links to documents which extol entity management:

- OASIS Entity Resolution Technical Committee - see especially the specification for OASIS Catalogs (TR 9401:1995 Entity Management) and the specification for XML Catalogs
- SGML/XML Special Topics: Entity Sets and Entity Management at the XML Cover Pages

- SGML/XML Special Topics: Catalogs, Formal Public Identifiers, Formal System Identifiers at the XML Cover Pages
- Arbortext column by Norman Walsh Standard Deviations from Norm - Issue Three: If You Can Name It, You Can Claim It!
- The Apache XML Commons project provides the entity resolver Java classes (which are used in Cocoon) and evolution of the Arbortext article into XML Entity and URI Resolvers.
- XML-Deviant article 2000-11-29 What's in a Name?
- DocBook: The Definitive Guide - Section 2.3 Public Identifiers, System Identifiers, and Catalog Files
- FAQ Catalogs and Docbook
- OASIS is the official home of the DocBook DTDs (see also DocBook Open Repository project at SourceForge)
- Organization for the Advancement of Structured Information Standards (OASIS)

1. Comments

add your comments