# DTDs and XML Catalogs

Richard Dallaway <richard@dallaway.com>
December 2002

From my current perspective as a developer who has to process XML documents and store them in a database, DTDs mostly just cause me pain. In working out some of the issues I now know that half of the problem was a misunderstanding about DTDs. The other half of the problem was solved by learning about XML catalogs and how to us them with JAXP (Xerces, in fact) and Java (JDK 1.4). These are my notes on the experience.

Modified:
2003-02-16: Added notes on XSL-T.

## The problems I was having with DTDs

In the task I'm currently working on, I receive XML documents, extract a few attributes, and then store the whole document in a database. A web application reads the XML out of the database and makes use of it.

You would think that a DTD would make life easier for me: the DTD lays out the rules for the XML, guaranteeing that certain attributes or nodes will be available. If the XML I'm processing doesn't validate against these rules, I can reject the document. However:

1. I don't supply the DTD as it's specified in the XML document. So the sender could change the DTD or point to a different DTD completely. Where's my control?
2. One of the servers involved in processing the XML is behind a firewall and can't see out to the Internet. How can I get at the DTD to use it? I can't just ignore it because it might contain important entity definitions. Even if I could connect to get the DTD, that's an expensive operation each time I want to parse a document.

These issues can be resolved. The control comes from understanding the public identifier part of a doctype. The second point is fixed by using XML catalogs.

## Public identifiers

Take a look at this document type definition from an XML document:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
           "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd" >
```

The `-//W3C//DTD SVG 20001102//EN` part is the public identifier and the `http://...` part is the URI. I've typically let the parser use the URI part to fetch the DTD, which has caused me problems (either worries about the DTD changing or not being able to fetch the document). It turns out that the URI is just hint for which DTD to use. It all became clear on page 30 of *XML in a Nutshell* (see References):

> *"The name of the public ID uniquely identifies the XML application in use. Generally a URI is also given as a backup in case the validator [parser] does not recognize the public ID. ... In practice, public IDs aren't used very often. Almost all XML parsers rely on the URI to actually validate the document."*

In other words the URI isn't the definitive place for the DTD: the ID is the important thing. Most parsers use the URI, but it doesn't have to be that way. So my first change to the application was to reject any public IDs that I don't recognize. This makes the code less flexible in that if the ID changes, the code will reject the document. Presumably, though, the public ID will only change if the interface described in the DTD changes. I think that's a reasonable contract to assume.

## XML Catalogs

The problem of how to access DTDs in a location-independent way is solved using XML Catalogs. In summary, a catalog maps a public ID into some other resource. Normal Walsh has a great write up on this (see References) which I recommend you read.

For my specific application this meant mapping public IDs to .dtd files held in a JAR file (for deployment) or on my classpath (during development). For every DTD I wanted to use, I made sure I had a local copy. Then I just included Sun's `resolver.jar` and modified my parsing code:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setValidating(true); // If you want DTD validation
DocumentBuilder parser = factory.newDocumentBuilder();

// Start of catalog mapping magic:
CatalogResolver resolver = new CatalogResolver();
parser.setEntityResolver(resolver);
// End of magic

Document dom = parser.parse(whatever...);
```

A couple of files tell the resolver where to locate the DTDs. First, the CatalogResolver implementation looks for a file called `CatalogManager.properties` anywhere on your classpath. Mine looks like this:

```
verbosity=4
catalogs=./dtd/catalog.xml
prefer=public
static-catalog=yes
relative-catalogs=no
```

...which more-or-less says “the mappings are in ./dtd/catalog.xml”. That location is relative to wherever the `CatalogManger.properties` file was found.

The catalog itself looks like this:

```
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
<public publicId="-//W3C//DTD SVG 20001102//EN"
        uri="external/svg-20001102.dtd"/>
</catalog>
```

...which maps the public ID to a specific `.dtd` file, the final location of which is relative to the location of `catalog.xml`. In my case my source tree would look like this:

```
src/com/foo/baz/Blah.java...
src/CatalogManager.properties
src/dtd/catalog.xml
src/dtd/external/svg-20001102.dtd
```

When I deployed my application, I made sure I included the `CatalogMangager.properties` and the dtd folder in the JAR.

And that's pretty much all there is to do: DTDs will be resolved from local files, and unrecognized DTDs can be rejected by comparing the public ID against the public IDs in the catalog. When parsing documents using the CatalogResolver, you'll see debugging output telling you that certain public IDs are being mapped to local files.

## Summary

Public IDs are the important thing about DTDs. Ignore the URIs by downloading the DTD and making use of an XML catalog, and in particular Sun's implementation of a catalog (`resolver.jar`). Doing this brings control back to the receiver of an XML document.

## A note about XSL-T

The `resover.jar` code can also be used in XSL transformations to locate included files. Here's a code snippet to show how it's done:

```
TransformerFactory factory = TransformerFactory.newInstance();
// Start of  magic:
CatalogResolver resolver = new CatalogResolver();
factory.setURIResolver(resolver);
// End of magic
template = factory.newTemplates(whatever...);
```

With this in place `<xsl:import>` tags read when creating the template will look to the catalog rules to locate xsl files.

## References

- Harold, E.R & Means, W. Scott *XML In A Nutshell* http://www.dallaway.com/reading/archive/xmlnut.xml
- Walsh, Norman. *XML Entity and URI Resolvers* (including resolver.jar) http://wwws.sun.com/software/xml/developers/resolver/article/