Namespaces and Schema Organization

Introduction and Issues

Mike Rawlins for XML Forum
March 2, 2001

Introduction

In developing projects for high level languages such as C/C++, Java, or COBOL, it is common to set up a library or directory of standard header (or include) files that are used throughout the project. XML provides facilities for doing this, but offers many more choices. This paper provides a brief introduction to the main concepts and facilities used for this purpose, and highlights a few of the issues involved in the various choices. It is not meant to be an exhaustive discussion and present all of the technical concepts, but serve as a starting point for discussion.

Namespaces

At the very simplest, a *namespace* is nothing more than a label. Namespaces are identified in XML using URI syntax. For example, http://www.example.com/PO1 is one of the namespace examples used in the XML Schema Primer. Namespaces are primarily used as a way to distinguish between identical names (XML elements or attributes) that have different semantic meaning. They are primarily used when a schema or instance document is combined using elements and attributes from two different vocabularies that have common elements. For example, if in my own schema for chemical markup language I have a name "element" I must distinguish it from the way "element" is used in the XML schema definition. There are various mechanisms for doing this, but the most common way is to associate a qualifier with a namespace and then prefix the element or attribute with the qualifier. Using the same example, I could associate the qualifier "xsd" with the XML schema namespace http://www.w3.org/2000/10/XMLSchema. Whenever I used "xsd:element" my processor would know to use the XML schema meaning of "element" rather than the chemistry meaning.

Namespaces are not part of XML 1.0, but are used in most of the other XML recommendations. With a few exceptions, using namespaces for schemas designed for data exchange is optional and not required. The way namespaces are used in schema definition affects the way the instance documents built according to the schemas look. For example, I may define my chemical schema so that "element" when referring to a chemical element may be unqualified, as <element>, or require a qualifier, as <chemspace:element>. This is one of the primary issues that designers must deal with in determining how to use namespaces.

There is a very important point to realize regarding a namespace URI. In and of itself, even though a namespace URI may look like the URL for a web page or another resource on the Internet, the namespace URI is only a label. There may or may not be any content associated with the URI. In other words, there may be no there there. For example, there may not even be an Internet domain called "example.com", not to mention a schema file called PO1 at that location.

There are several other issues involving the use of namespaces. How should the namespace be organized? Should there be a single namespace for the complete project, or should it be hierarchical with two or more

levels? Shall we use names, and thus one or more namespaces that have been defined by another organization?

Building Schemas from Multiple Sources

Where namespaces become especially complicated is when schemas are built from two or more sources. This will probably be the predominant way in which schemas are constructed since it facilitates reuse of components. The choices can have a great impact on several important factors such as ease of use and maintainability.

Most high level programming languages provide only one way to include header files. The XML schema facility, in contrast, provides two ways with several variations. This complicates the task of designing the architecture. The two primary mechanisms are *include* and *import*. Both are intimately related to how namespaces are used.

include is very similar in concept to a C/C++ or Java *#include* preprocessor directive. The included schema file is treated by the processor as if it were a part of the original schema file. The namespace implication of using *include* is that the included file must have the same target namespace as the original schema. This means that for included files to be used in several schemas, all schemas and include files either must have the same target namespace or not declare a target namespace.

import is similar to include, but in strict terms a *namespace* is *imported* instead of a file being physically included. Two directives are necessary to do an import. The first is a namespace declaration, and the second is the actual import of the namespace. With an *import*, there must be content at the URI location. So, in this use of namespaces, there must be a there there. After the import is done, the same options and issues regarding use of namespaces remain, as discussed above.

Issues and choices involving when to use *import* and *include* are very bound up with the overall choices for namespace architecture. Other issues involve whether to build a single schema file with all commonly used elements, or whether to build one file per element. This choice is related to overall project decisions for version control and release schedules (One should always assume that we won't get it right the first time, and that there will be multiple releases). Another potential issue is compatibility with base XML 1.0 and processors that don't understand namespaces or schemas. Several aspects of schemas can be rendered backwardly into XML 1.0 DTDs, but a restriction for optimal support for this lends itself to an architecture which uses *include* and no target namespaces rather than *import* and namespaces.

Selected Reference Documents

- Namespaces in XML - W3C Recommendation 14-January-1999
- XML Schema, Parts 0 and 1 - W3C Candidate Recommendation 22-September-2000
- Uniform Resource Identifiers (URI): Generic Syntax - IETF RFC 2396