Default Namespace - targetNamespace or XMLSchema?

Table of Contents

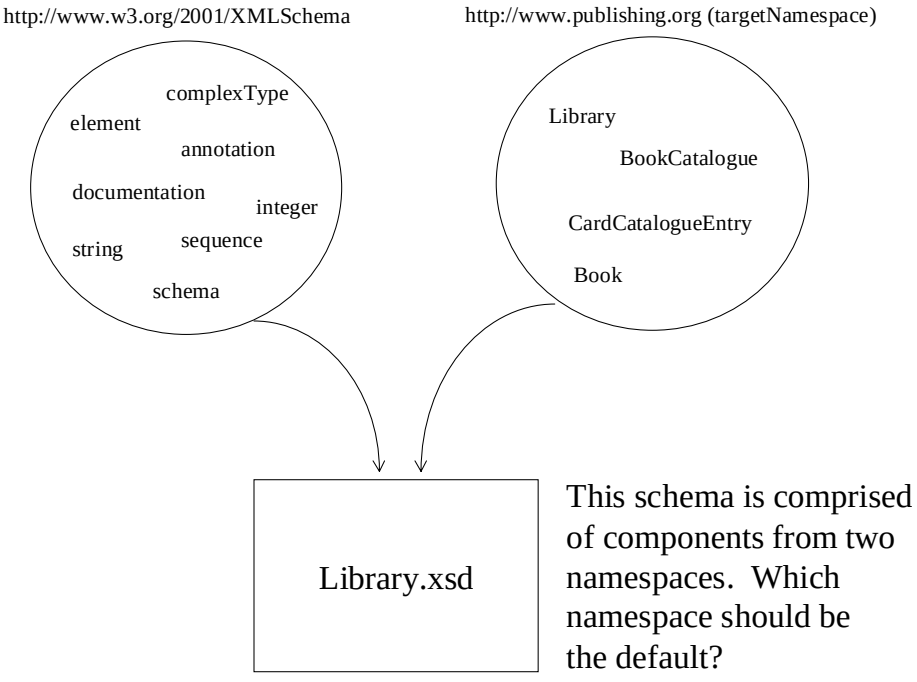
Issue
Introduction
Approach 1: Default XMLSchema, Qualify targetNamespace
Approach 2: Qualify XMLSchema, Default targetNamespace
Approach 3: No Default Namespace - Qualify both XMLSchema and targetNamespace
Best Practice

Issue

When creating a schema should XMLSchema (i.e., http://www.w3.org/2001/XMLSchema) be the default namespace, or should the targetNamespace be the default, or should there be no default namespace?

Introduction

Except for no-namespace schemas, every XML Schema uses at least two namespaces - the targetNamespace and the XMLSchema (http://www.w3.org/2001/XMLSchema) namespace, e.g.,

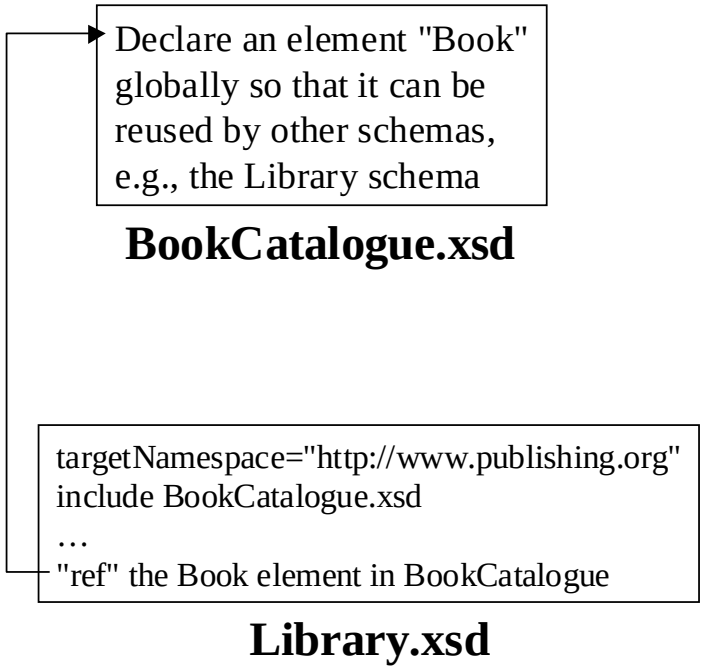


There are three ways to design your schemas, with regards to dealing with these two namespaces:

1. Make XMLSchema the default namespace, and explicitly qualify all references to components in the targetNamespace.
2. Vice versa - make the targetNamespace the default namespace, and explicitly qualify all components from the XMLSchema namespace.
3. Do not use a default namespace - explicitly qualify references to components in the targetNamespace and explicitly qualify all components from the XMLSchema namespace.

Let's look at each approach in detail. In the following discussions we will consider this scenario:

The BookCatalogue schema must either:
- have the same namespace as the Library schema, or
- have no namespace.



Approach 1: Default XMLSchema, Qualify targetNamespace

Below is a Library schema which demonstrates this design approach. It <include>s a BookCatalogue schema, which contains a declaration for a Book element. The Library schema references (“ref”s) the Book element.

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.library.org"
        xmlns:lib="http://www.library.org"
        elementFormDefault="qualified">
  <include schemaLocation="BookCatalogue.xsd"/>
  <element name="Library">
    <complexType>
      <sequence>
        <element name="BookCatalogue">
          <complexType>
            <sequence>
              <element ref="lib:Book"
                       maxOccurs="unbounded"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

Default namespace is XMLSchema

Set "lib" to point to the targetNamespace

Qualify the reference to Book

Note that XMLSchema is the default namespace. Consequently, all the components used to construct the schema - element, include, complexType, sequence, schema, etc - have no namespace qualifier on them.

There is a namespace prefix, lib, which is associated with the targetNamespace. Any references (using the “ref” attribute) to components in the targetNamespace (Library, BookCatalogue, Book, etc) are explicitly qualified with lib (in this example there is a ref to lib:Book).

Advantages:

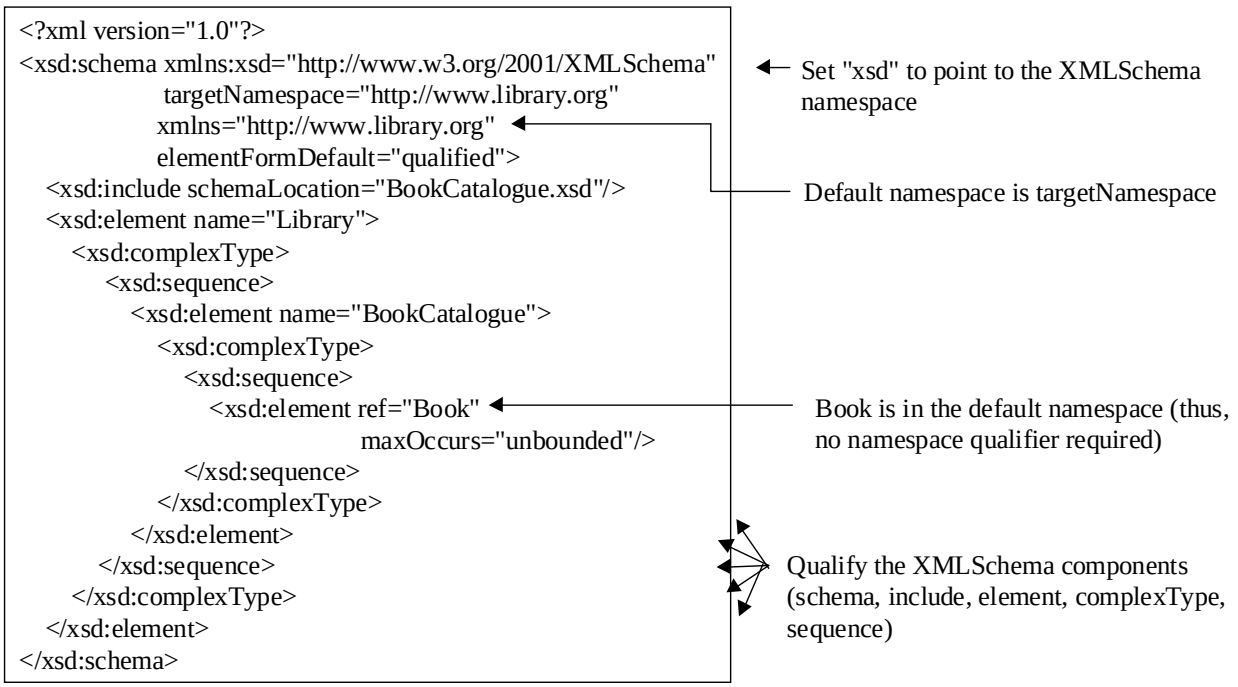
If your schema is referencing components from multiple namespaces then this approach gives a consistent way of referring to the components (i.e., you always qualify the reference).

Disadvantages:

Schemas which have no-targetNamespace must be designed so that the XMLSchema components (element, complexType, sequence, etc) are qualified. If you adopt this approach to designing your schemas then in some of your schemas you will qualify the XMLSchema components and in other schemas you won’t qualify the XMLSchema components. Changing from one way of designing your schemas to another way can be confusing.

Approach 2: Qualify XMLSchema, Default targetNamespace

This design approach is the mirror image of the first approach:



With this approach all the components used to construct a schema are namespace qualified (with xsd:).

There is a default namespace declaration that declares the targetNamespace to be the default namespace. Any references to components in the targetNamespace are not namespace qualified (note that the ref to Book is not namespace qualified).

Advantages:

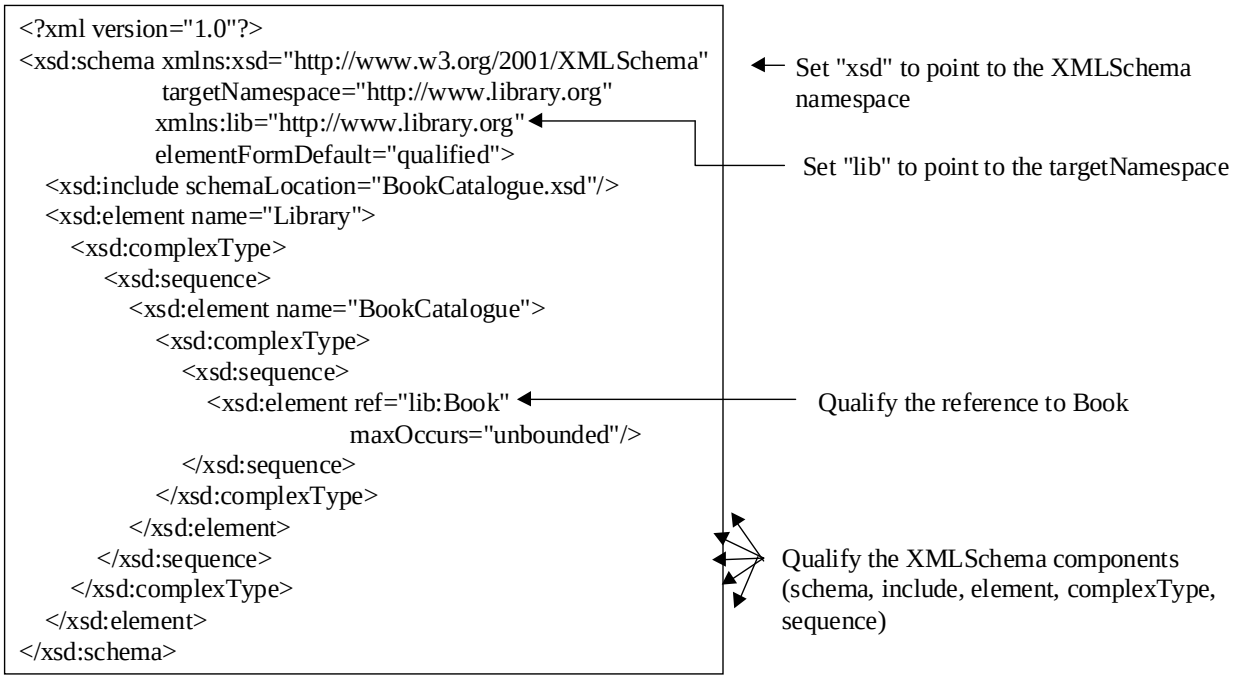
Schemas which have no-targetNamespace must be designed so that the XMLSchema components (element, complexType, sequence, etc) are qualified. This approach will work whether your schema has a targetNamespace or not. Thus, with this approach you have a consistent approach to designing your schemas - always namespace-qualify the XMLSchema components.

Disadvantages:

If your schema is referencing components from multiple namespaces then for some references you will namespace-qualify the reference, whereas other times you will not (i.e., when you are referencing components in the targetNamespace). This variable use of namespace qualifiers in referencing components can be confusing.

Approach 3: No Default Namespace - Qualify both XMLSchema and targetNamespace

This design approach does not have a default namespace:



Note that both the XMLSchema components are explicitly qualified, as well as are references to components in the targetNamespace.

Advantages:

[1] Schemas which have no-targetNamespace must be designed so that the XMLSchema components (element, complexType, sequence, etc) are qualified. With this approach all your schemas are designed in a consistent fashion.

[2] If your schema is referencing components from multiple namespaces then this approach gives a consistent way of referencing components (i.e., you always qualify the reference).

Disadvantages:

Very cluttered: being very explicit by namespace qualifying all components and all references can be annoying when reading the schema.

Best Practice

There is no clear-cut best practice with regards to this issue. In large part it is a matter of personal preference.