# Hide (Localize) Namespaces Versus Expose Namespaces

## Table of Contents

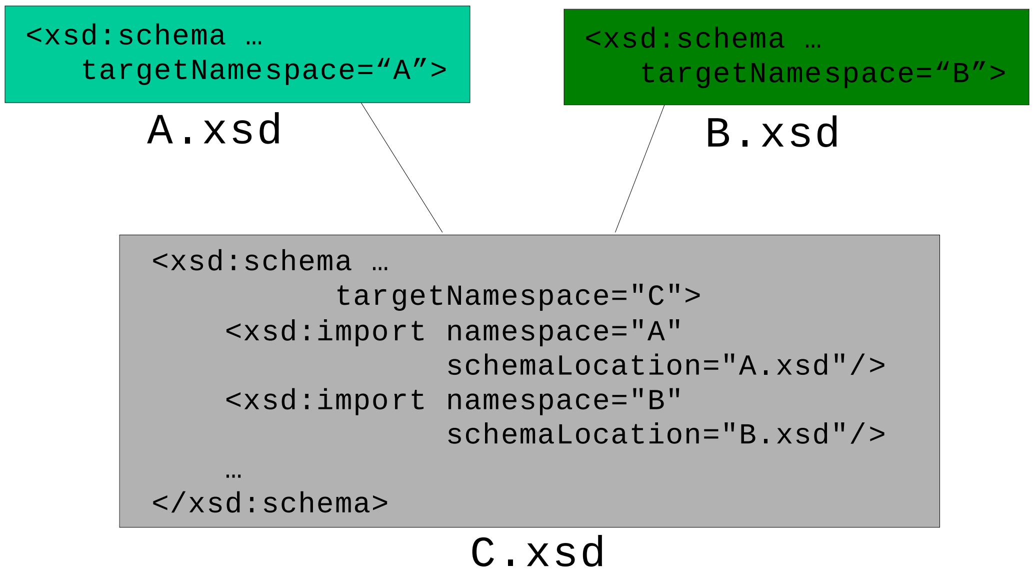
Issue
Introduction
Example
Technical Requirements for Hiding (Localizing) Namespaces
Best Practice

## Issue

*When should a schema be designed to hide (localize) within the schema the namespaces of the elements and attributes it is using, versus when should it be designed to expose the namespaces in instance documents?*

## Introduction

A typical schema will reuse elements and types from multiple schemas, each with different namespaces.



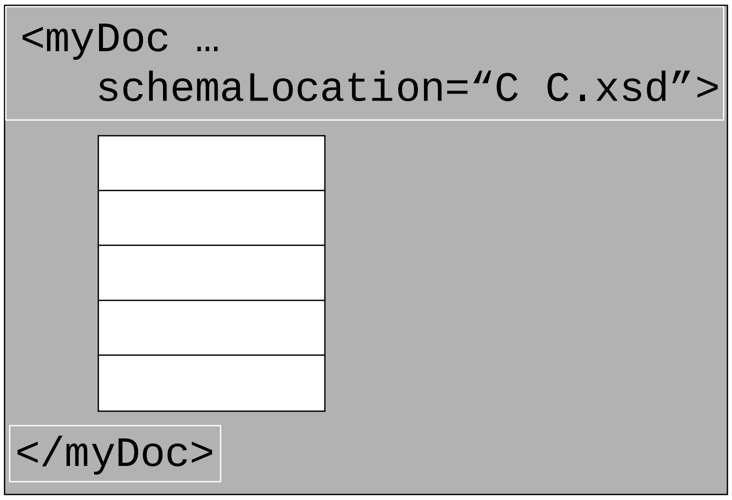
```
<xsd:schema …
   targetNamespace="A">
```

```
<xsd:schema …
   targetNamespace="B">
```

```
<xsd:schema …
          targetNamespace="C">
    <xsd:import namespace="A"
                schemaLocation="A.xsd"/>
    <xsd:import namespace="B"
                schemaLocation="B.xsd"/>
    …
</xsd:schema>
```

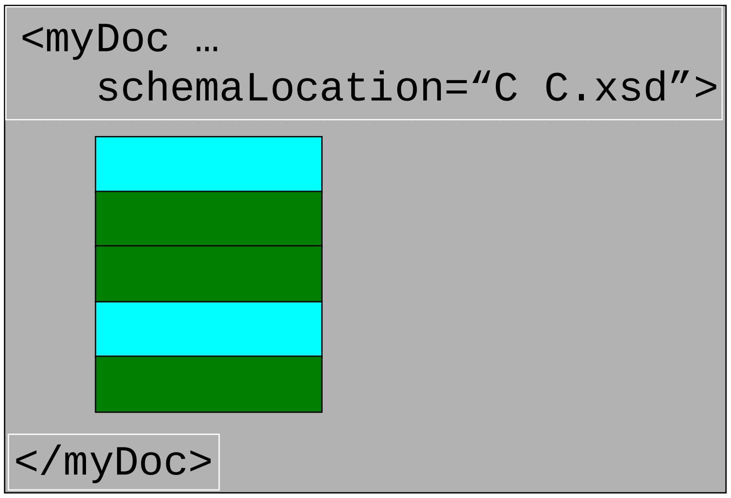
A schema, then, may be comprised of components from multiple namespaces. Thus, when a schema is designed the schema designer must decide whether or not the origin (namespace) of each element should be exposed in the instance documents.

**Instance Document**



The namespaces of the components are not visible in the instance documents.

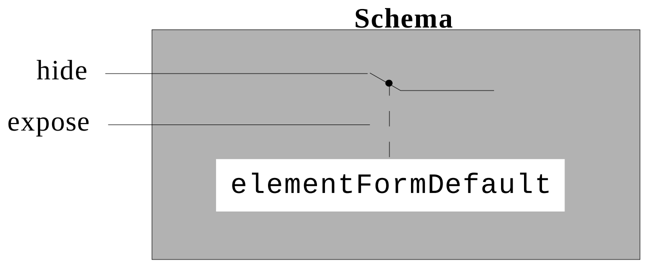
**Instance Document**



The namespaces of the components are visible in the instance documents.

A binary switch attribute in the schema is used to control the hiding/exposure of namespaces: by setting elementFormDefault="unqualified" the namespaces will be hidden (localized) within the schema, and by setting elementFormDefault="qualified" the namespaces will be exposed in instance documents.

## elementFormDefault - the Exposure "Switch"

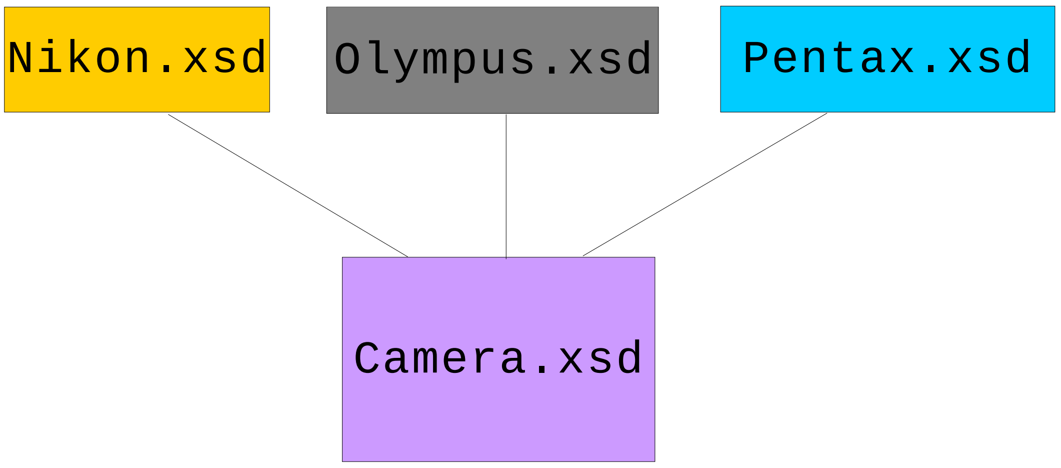


```
<xsd:schema …
     elementFormDefault="qualified">
```

vs

```
<xsd:schema …
     elementFormDefault="unqualified">
```

**Example:**



Below is a schema for describing a camera. The camera schema reuses components from other schemas - the camera's <body> element reuses a type from the Nikon schema, the camera's <lens> element reuses a type from the Olympus schema, and the camera's <manual_adaptor> element reuses a type from the Pentax schema.

**Camera.xsd**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.camera.org"
            xmlns:nikon="http://www.nikon.com"
            xmlns:olympus="http://www.olympus.com"
            xmlns:pentax="http://www.pentax.com"
            elementFormDefault="unqualified">
    <xsd:import namespace="http://www.nikon.com"
                schemaLocation="Nikon.xsd"/>
    <xsd:import namespace="http://www.olympus.com"
                schemaLocation="Olympus.xsd"/>
    <xsd:import namespace="http://www.pentax.com"
                schemaLocation="Pentax.xsd"/>
    <xsd:element name="camera">
        <xsd:complexType>
             <xsd:sequence>
                 <xsd:element name="body" type="nikon:body_type"/>
                 <xsd:element name="lens" type="olympus:lens_type"/>
                 <xsd:element name="manual_adapter"
                              type="pentax:manual_adapter_type"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

This schema is designed to hide namespaces

Note the three <import> elements for importing the Nikon, Olympus, and Pentax components. Also note that the <schema> attribute, elementFormDefault has been set to the value of unqualified. This is a critical attribute. Its value controls whether the namespaces of the elements being used by the schema will be hidden or exposed in instance documents (thus, it behaves like a *switch* turning namespace exposure on/off). Because it has been set to “unqualified” in this schema, the namespaces will be remain hidden (localized) within the schema, and will not be visible in instance documents, as we see here:

**Camera.xml (namespaces hidden)**

**Instance Document**

```
<?xml version="1.0"?>
<my:camera xmlns:my="http://www.camera.org"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation=
                   "http://www.camera.org
                    Camera.xsd">
    <body>
       <description>Ergonomically designed casing for easy handling</description>
    </body>
    <lens>
        <zoom>300mm</zoom>
        <f-stop>1.2</f-stop>
    </lens>
    <manual_adapter>
        <speed>1/10,000 sec to 100 sec</speed>
    </manual_adapter>
</my:camera>
```

Instance document with namespaces hidden (localized) in the schema.

--> The fact that the `<description>` element comes from the Nikon schema, the `<zoom>` and `<f-stop>` elements come from the Olympus schema, and the `<speed>` element comes from the Pentax schema is totally transparent to the instance document.

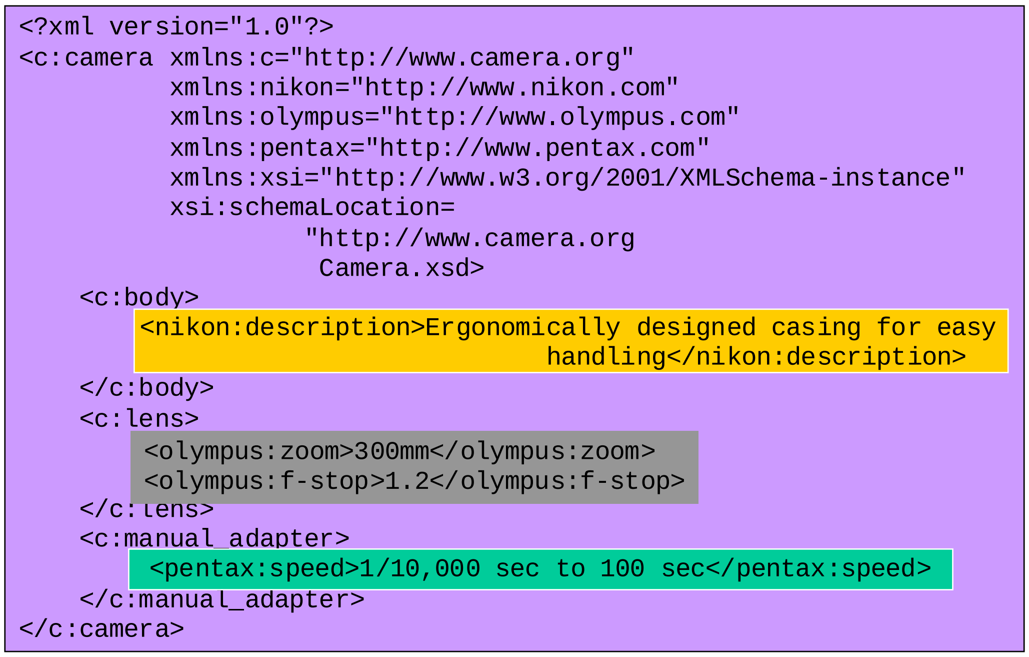
The only namespace qualifier exposed in the instance document is on the <camera> root element. The rest of the document is completely free of namespace qualifiers. The Nikon, Olympus, and Pentax namespaces are completely hidden (localized) within the schema!

Looking at the instance document one would never realize that the schema got its components from three other schemas. Such complexities are localized to the schema. Thus, we say that the schema has been designed in such a fashion that its component namespace complexities are “hidden” from the instance document.

On the other hand, if the above schema had set elementFormDefault=“qualified” then the namespace of each element would be exposed in instance documents. Here’s what the instance document would look like:

**Camera.xml (namespaces exposed)**

## Instance Document



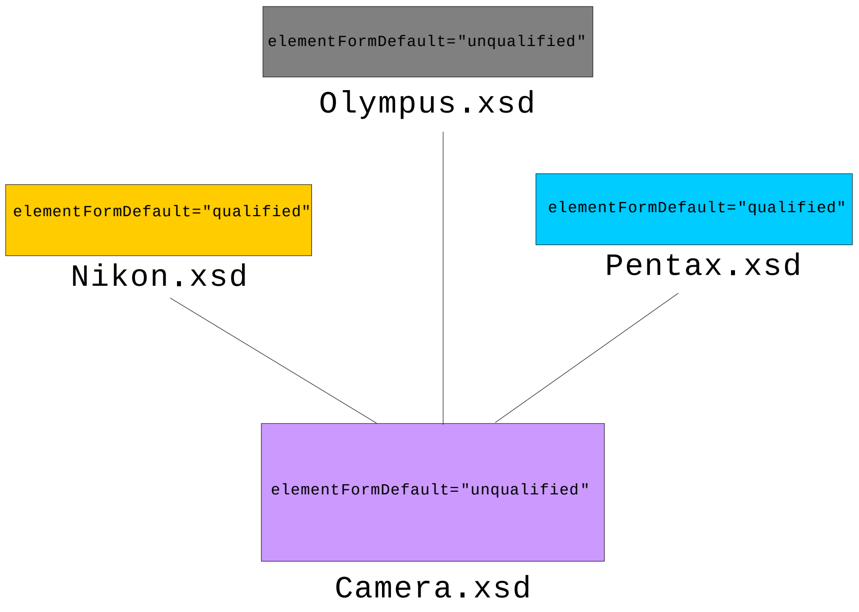
```
<?xml version="1.0"?>
<c:camera xmlns:c="http://www.camera.org"
          xmlns:nikon="http://www.nikon.com"
          xmlns:olympus="http://www.olympus.com"
          xmlns:pentax="http://www.pentax.com"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation=
                   "http://www.camera.org
                    Camera.xsd>
    <c:body>
        <nikon:description>Ergonomically designed casing for easy
                                   handling</nikon:description>
    </c:body>
    <c:lens>
        <olympus:zoom>300mm</olympus:zoom>
        <olympus:f-stop>1.2</olympus:f-stop>
    </c:lens>
    <c:manual_adapter>
        <pentax:speed>1/10,000 sec to 100 sec</pentax:speed>
    </c:manual_adapter>
</c:camera>
```

Instance document with namespaces exposed

Note that each element is explicitly namespace-qualified. Also, observe the declaration for each namespace. Due to the way the schema has been designed, the complexities of where the schema obtained its components have been "pushed out" to the instance document. Thus, the reader of this instance document is "exposed" to the fact that the schema obtained the description element from the Nikon schema, the zoom and f-stop elements from the Olympus schemas, and the speed element from the Pentax schema.

**All Schemas must have a Consistent Value for elementFormDefault!**

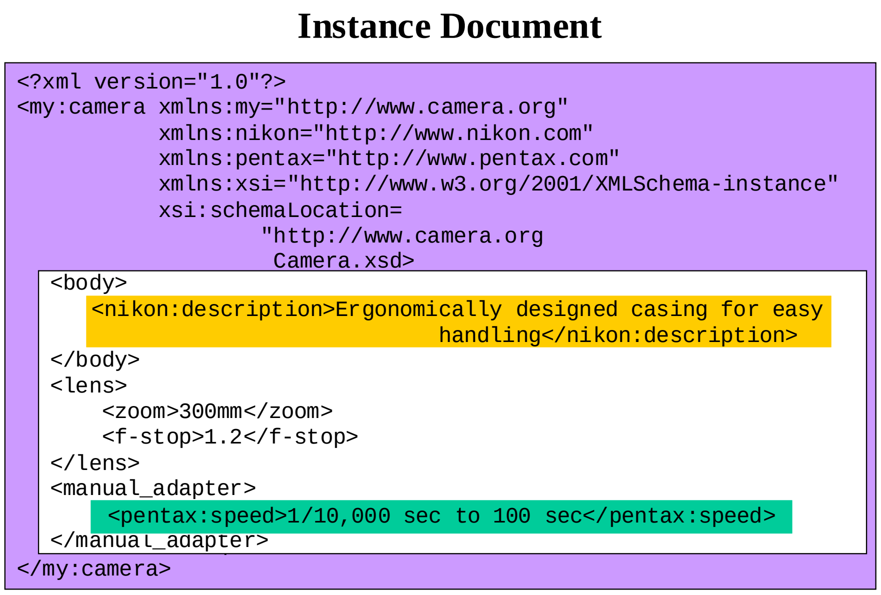
Be sure to note that **elementFormDefault applies just to the schema that it is in**. It does not apply to schemas that it includes or imports. Consequently, if you want to hide namespaces then **all** schemas involved must have set elementFormDefault="unqualified". Likewise, if you want to expose namespaces then **all** schemas involved must have set elementFormDefault="qualified". To see what happens when you "mix" elementFormDefault values, let's suppose that Camera.xsd and Olympus.xsd have both set in their schema elementFormDefault="unqualified", whereas Nikon.xsd and Pentax.xsd have both set elementFormDefault="qualified".



elementFormDefault="unqualified"
Olympus.xsd
elementFormDefault="qualified"
Nikon.xsd
elementFormDefault="qualified"
Pentax.xsd
elementFormDefault="unqualified"
Camera.xsd

Here’s what an instance document looks like with this “mixed” design:

**Camera.xml (mixed design)**



**Instance Document**

```
<?xml version="1.0"?>
<my:camera xmlns:my="http://www.camera.org"
           xmlns:nikon="http://www.nikon.com"
           xmlns:pentax="http://www.pentax.com"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation=
                   "http://www.camera.org
                    Camera.xsd>
  <body>
     <nikon:description>Ergonomically designed casing for easy
                           handling</nikon:description>
  </body>
  <lens>
      <zoom>300mm</zoom>
      <f-stop>1.2</f-stop>
  </lens>
  <manual_adapter>
      <pentax:speed>1/10,000 sec to 100 sec</pentax:speed>
  </manual_adapter>
</my:camera>
```

**Hiding/exposure mix:** This instance document has the Nikon and Pentax namespaces exposed, while the Camera and Olympus namespaces are hidden.

Observe that in this instance document some of the elements are namespace-qualified, while others are not. Namely, those elements from the Camera and Olympus schemas are not qualified, whereas the elements from the Nikon and Pentax schemas are qualified.

## Technical Requirements for Hiding (Localizing) Namespaces

There are two requirements on an element for its namespace to be hidden from instance documents:

[1] The value of elementFormDefault must be “unqualified”.

[2] The element must not be globally declared. For example:

```
<?xml version=“1.0”?>
<xsd:schema ...>
  <xsd:element name=“foo”>
  ...
</xsd:schema ...>
```

The element foo can never have its namespace hidden from instance documents, regardless of the value of elementFormDefault. foo is a *global* element (i.e., an immediate child of <schema>) and therefore must always be qualified. To enable namespace hiding the element must be a local element.

## Best Practice

For this issue there is no definitive Best Practice with respect to whether to design your schemas to hide/localize namespaces, or design it to expose namespaces. Sometimes it’s best to hide the namespaces. Othertimes it’s best to expose the namespaces. Both have their pluses and minus, as is discussed below.

However, there are Best Practices with regards to other aspects of this issue. They are:

1. Whenever you create a schema, make **two** copies of it. The copies should be identical, except that in one copy set elementFormDefault=“qualified”, whereas in the other copy set elementFormDefault=“unqualified”. If you make two versions of all your schemas then people who use your schemas will be able to implement either design approach - hide (localize) namespaces, or expose namespaces.
2. Minimize the use of global elements and attributes so that elementFormDefault can behave as an “exposure switch”. The rationale for this was described above, in *Technical Requirements for Hiding (Localizing) Namespaces*

### Advantages of Hiding (Localizing) Component Namespaces within the Schema

The instance document is simple. It’s easy to read and understand. There are no namespace qualifiers cluttering up the document, except for the one on the document element (which is okay because it shows the domain of the document). The knowledge of where the schema got its components is irrelevant and localized to the schema.

Design your schema to hide (localize) namespaces within the schema ...

– when simplicity, readability, and understandability of instance documents is of utmost importance when namespaces in the instance document provide no necessary additional information. In many scenarios the users of the instance documents are not XML-experts. Namespaces would distract and confuse such users, where they are just concerned about structure and content.

– when you need the flexibility of being able to change the schema without impact to instance documents. To see this, imagine that when a schema is originally designed it imports elements/types from another namespace. Since the schema has been designed to hide (localize) the namespaces, instance documents do not see the namespaces of the imported elements. Then, imagine that at a later date the schema is changed such that instead of importing the elements/types, those elements and types are declared/defined right within the schema (inline). This change from using elements/types from another namespace to using elements/types in the local namespace has no impact to instance documents because the schema has been designed to shield instance documents from where the components come from.

**Advantages of Exposing Namespaces in Instance Documents**

If your company spends the time and money to create a reusable schema component, and makes it available to the marketplace, then you will most likely want recognition for that component. Namespaces provide a means to achieve recognition. For example,

```
<nikon:description>
    Ergonomically designed casing for easy handling</
nikon:description>
```

There can be no misunderstanding that this component comes from Nikon. The namespace qualifier is providing information on the origin/lineage of the description element.

Another case where it is desirable to expose namespaces is when processing instance documents. Oftentimes when processing instance documents the namespace is required to determine how an element is to be processed (e.g., “if the element comes from this namespace then we’ll process it in this fashion, if it comes from this other namespace then we’ll process it in a different fashion”). If the namespaces are hidden then your application is forced to do a lookup in the schema for every element. This will be unacceptably slow.

Design your schema to expose namespaces in instance documents ...

– when lineage/ownership of the elements are important to the instance document users (such as for copyright purposes).

– when there are multiple elements with the same name but different semantics then you may want to namespace-qualify them so that they can be differentiated (e.g, publisher:body versus human:body). [In some cases you have multiple elements with the same name and different semantics but the context of the element is sufficient to determine its semantics. Example: the title element in <person><title> is easily distinguished from the title element in <chapter><title>. In such cases there is less justification for designing your schema to expose the namespaces.]

– when processing (by an application) of the instance document elements is dependent upon knowledge of the namespaces of the elements.

**Note about elementFormDefault and xpath Expressions**

We have seen how to design your schema so that elementFormDefault acts as an “exposure switch”. Simply change the value of elementFormDefault and it dictates whether or not elements are qualified in instance documents. In general, no other changes are needed in the schema other than changing the value of elementFormDefault. However, if your schema is using <key> or <unique> then you will need to make modifications to the xpath expressions when you change the value of elementFormDefault.

If elementFormDefault=“qualified” then you must qualify all the references in the xpath expression.

**Example:**

```
<xsd:key name=“PK”>

   <xsd:selector xpath=“c:Camera/c:lens”>

   <xsd:field xpath=“c:zoom”>

</xsd:key>
```

Note that each element in the xpath expression is namespace-qualified.

If elementFormDefault=“unqualified” then you must NOT qualify the references in the xpath expression.

**Example:**

```
<xsd:key name=“PK”>

   <xsd:selector xpath=“Camera/lens”>

   <xsd:field xpath=“zoom”>

</xsd:key>
```

Note that none of the elements in the xpath expressions are namespace-qualified.

So, as you switch between exposing and hiding namespaces you will need to take the xpath changes into account.