XML NAMESPACE USE-CASES

March 6, 2006

1   Introduction

The Network Namespace Recommendation defines a systematic way of uniquely identifying and locating an XML schema. URL-formatted namespaces create a linkage between the XML schema identifier and the Exchange Network (EN) XML Repository.

This document covers some common usage scenarios and gives examples for development practices when implementing the recommendation.

2   XML Schema and Namespaces

2.1   What is a Root Schema?

The term "root schema" can be used when referring to the root element of an instance file or schema, or the root schema of a namespace. In this document "root schema" refers to the latter, the root schema of a namespace.

According to the Namespace recommendation, developers should maintain a one-to-one relationship between namespaces and root schema. This means that every namespace should have only one root schema and every root schema should have a corresponding namespace. This allows for instance files to conform by declaring a single root via the `targetNamespace` attribute. If developers follow the suggestions in this document, the declaration will permit an instance file to import all the types and elements in a namespace with a single statement.

2.2   Constructing a Root Schema File

A schema is identified by the namespace URI. It may contain many files that define types and elements, but suggested practice is to have a single file that loads other schemas.

In general, developers are encouraged to:
1. Define a single schema that includes, or recursively includes, all types and elements defined in the namespace.
2. Define the root element of instance documents in the file.

By following these practices, schema users can immediately resolve the location of a schema by following its namespace URI. If the root elements are defined in this file they can ascertain necessary information without having to browse additional files.

Assuming there are three parallel schema files in the namespace,
> **EN_EXAMPLE_SCHEMA1_V1.0.xsd**
> **EN_EXAMPLE_SCHEMA2_V1.0.xsd**
> **EN_EXAMPLE_SCHEMA3_V1.0.xsd**
the root schema can be constructed by including all three files.

```
<xsd:schema
    targetNamespace="http://www.exchangenetwork.net/schema/exa
    mple/1" elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="1.0"
    xmlns="http://www.exchangenetwork.net/schema/example/1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
<xsd:include
    schemaLocation=".\EN_EXAMPLE_SCHEMA1_V1.0.xsd"/>
<xsd:include schemaLocation=".\EN_EXAMPLE_SCHEMA2_V1.0.xsd
    "/>
<xsd:include schemaLocation=".\EN_EXAMPLE_SCHEMA3_V1.0.xsd
    "/>
        <xsd:element name="MyRootElement"
      type="MyRootElementType">
            <xsd:annotation>
              <xsd:documentation> It is best practice to define the
                top- level element in the root
                schema</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
</xsd:schema>
```

When this file is used, all the type and element definitions will be included automatically.

When including other schema files, developers should try to avoid:
- Including a schema file from a different namespace. Schema files from different namespace should be **imported.** An exception is when schema files have no `targetNamespace`; then they can be included.
    - According to the EN Namepsace recommendation, Shared Schema Components are the only schema files on the EN that do not declare a `targetNamespace.`
- Circular includes. This is a situation where FileA includes FileB, but FileB also includes FileA.

The Namespace Recommendation also requires that a default file, named `index.xsd`, occur in every namespace directory. This file should include the root schema of the namespace.

2.3 Using Schema Files in Different Namespaces

Using data definitions from published EN schema is a good practice. Developers should not, however, cut and paste those definitions into schema files. The cut and paste approach redefines the same types or elements that are already defined in other schema files. Developers should not redefine identical elements that are published in other EN schemas.

The best method of reusing schemas on the EN is importing other schemas into the new schema definition. Documents can import other schemas in three ways:
1. Import the schema, keeping its namespace unchanged.
2. Import the schema into another namespace.
3. Import the schema into the document's namespace (assuming there are no name collisions).

Assuming that the imported schema is to be used in a new namespace,
> **http://www.exchangenetwork.net/schemas/import_example/1**
the schema can be imported as below:

```
<xsd:schema
    targetNamespace="http://www.exchangenetwork.net/schema/import_example/1" elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="1.0"
    xmlns="http://www.exchangenetwork.net/schema/import_example/1"
    xmlns:ex="http://www.exchangenetwork.net/schema/example/1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import
      namespace="http://www.exchangenetwork.net/schema/example/1"
      schemaLocation="http://www.exchangenetwork.net/schema/example/1"/>
        <xsd:element name="MyNewRootElement"
        type="ex:MyRootElementType">
            <xsd:annotation>
              <xsd:documentation>It is best practice to define the
              top- level element in the root
              schema</xsd:documentation>
            </xsd:annotation>
        </xsd:element>
</xsd:schema>
```

In this example, all data definitions in
> **http://www.exchangenetwork.net/schema/example/1**
are imported, but leave the maintenance to the owner.

Best practices for importing schemas are:
1. Putting the import statement as the first child of the schema element.
2. Avoiding importing multiple files from the same namespace. Developers should avoid using multiple `schemaLocation` attributes under the `xsd:import` tag. Many tools will either ignore subsequent imports (because the namespace has already been imported), or replace the first imported file with the subsequent ones.
3. Not importing a schema that declares a `targetNamespace` into the importing document's namespace. Shared Schema components are the exception to this practice, since they do not declare a `targetNamespace`.

3   Namespace Design Approaches

Namespaces help schema designers organize data types and elements into different categories so that they can be shared without causing name collisions. However, too many namespaces can introduce complexity to the process of constructing or validating instance documents. Many tools create individual classes or objects for handling serialization and de-serialization of XML documents. Excessive namespaces can, in such situations, result in overly complicated software structures.

This section discusses some of the approaches for schema namespace design.

In general, schema designers should be cautious when creating multiple namespaces in a logical data flow, unless:
1. There is a potential for name collisions among schema components, or
2. Subcomponents are independently developed and maintained, or
3. Subcomponents need to be versioned independently or have independent version numbers.

3.1   Homogeneous Namespace

In a homogeneous namespace, all schema files declare the same namespace, reside in the same directory in the EN repository, and have the same `targetNamspace` URI.

If all schema files are in the same namespace, they can be included in the root schema. In addition, instance documents can use a single `targetNamespace` as the default namespace; therefore, fully qualified element names (element name with prefix) are not required. Instance documents are less verbose under a homogeneous namespace.

Below is an example of a homogeneous namespace with three message schema, all three of which would use identical namespace declarations:

> **EN_EXAMPLE_SCHEMA1_V1.0.xsd**
> **EN_EXAMPLE_SCHEMA2_V1.0.xsd**
> **EN_EXAMPLE_SCHEMA3_V1.0.xsd**

```xml
<xsd:schema
    targetNamespace="http://www.exchangenetwork.net/schema/example/1" elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0"
        ...
</xsd:schema>
```

The root schema can be constructed by using the same namespace declaration and including all three files.

```xml
<xsd:schema
    targetNamespace="http://www.exchangenetwork.net/schema/exa
    mple/1" elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="1.0"
    xmlns="http://www.exchangenetwork.net/schema/example/1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include
    schemaLocation=".\EN_EXAMPLE_SCHEMA1_V1.0.xsd"/>
  <xsd:include schemaLocation=".\EN_EXAMPLE_SCHEMA2_V1.0.xsd
    "/>
  <xsd:include schemaLocation=".\EN_EXAMPLE_SCHEMA3_V1.0.xsd
    "/>
      <xsd:element name="MyRootElement"
    type="MyRootElementType">
        <xsd:annotation>
         <xsd:documentation> It is best practice to define the
         top- level element in the root
         schema</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
</xsd:schema>
```

When possible, developers should strive to design logical data flows using homogeneous namespace.

3.2   Heterogeneous Namespace

In a heterogeneous namespace, one or more subcomponents declare different namespaces (i.e., having different `targetNamespace` names).

The approach should be carefully evaluated if the designer has control over all types and elements in the namespaces.

There are situations where heterogeneous namespaces have to be used. For instance, if a schema set needs to use common schemas that are in other namespaces, these foreign namespaces have to be introduced.

Note that schemas in foreign namespaces should be imported rather than included. The best approach of reusing a foreign schema is to reference it using absolute URL in the import statement. For example, if a document with the namespace
        http://www.exchangenetwork.net/schema/import_example/1
imports a schema with the namespace OtherSchema.xsd, the schema should use an absolute and fully-qualified URL in the `schemaLocation`, as shown below:

```
<xsd:import
    namespace="http://www.exchangenetwork.net/schema/import
    _example/1"
    schemaLocation="http://www.exchangenetwork.net/schema/im
    port_example/1"/>
```

Relative references (as shown below) should be avoided when importing schema in a foreign namespace.

```
<xsd:import
    namespace="http://www.exchangenetwork.net/schema/import
    _example/1" schemaLocation="./OtherSchema.xsd"/>
```

3.3   Chameleon Namespace

The chameleon namespace approach involves designing XML schema files without declaring a `targetNamespace`. The `targetNamespace` is declared in the root schema. In this approach, the subcomponents without namespaces are included in the root schema and assume whatever `targetNamespace` is declared in the root.

The word chameleon comes from the fact that the subcomponent assumes the namespace of the hosting environment, so the data and elements in the schema "**change**" namespaces from one application to another.

Chameleon namespace design simplifies sharing of common schemas across different namespaces. The approach is widely used in designing common shared schema components.

Below is an example of a chameleon namespace that includes a Shared Schema Component that does not declare a namespace:

```
<xsd:schema
    targetNamespace="http://www.exchangenetwork.net/schema/exa
    mple/1" elementFormDefault="qualified"
    attributeFormDefault="unqualified" version="1.0"
    xmlns="http://www.exchangenetwork.net/schema/example/1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include
        schemaLocation="http://www.exchangenetwork.net/schema/sc/SC_CountyI
        denty_v2.0.xsd" />
</xsd:schema>
```

However, processing of chameleon namespaces is not consistent in different XML tools, and there are reports that a predominant XML schema tool is unable to use such schema sets.

4 Shared Schema Components

Shared Schema Components (SSC) are data types and elements that can be used by more than one schema. The SSCs will be located in the EN repository at the address:

       http://www.exchangenetwork.net/schema/sc/

In the future, subdirectories may also be created within the sc directory to hold logical subcategories. This section discusses strategies that will assist in consistently managing common schema definitions.

Unlike data flow schemas, version changes in shared schemas could potentially affect many other schema sets. This section discusses an approach to minimize such impact while allowing changes of individual schema files in the repository.

4.1 Categorization and Versioning

Unlike the normal schema sets, SSCs typically define basic data types and elements, and they tend to be used individually by other schema designers. The most efficient means of using SSCs in other schema is the use of the `include` element. Because schemas cannot be included across namespaces, and the import statement only allows importation of a single file, managed SSCs on the Exchange Network will not declare a `targetNamespace.` This allows them to be included in other EN schema.

Shared component can be classified into different categories from data and element type point of view:

1. **Shared Schema Components**: These components will be located in subdirectories containing their major version number. For example, **http://www.exchangenetwork.net/schema/sc/** will contain all of the SSCs.
2. **Application Shared Components**: These data types are application specific, but shared by many data flow schemas. The directory for application data type could be constructed as: **http://www.exchangenetwork/schema/flow/subcategory/version**

4.1.1 Component Namespaces

For simple and complex common data types located within the …/sc/ folder, we recommend the chameleon schema design approach (i.e., the schema definition should have no namespace of its own). The data types will assume parent schema namespace when included. This is to simplify schema development and instance document construction.

For application specific data types, a namespace with URL of
   http://www.exchangenetwork/schema/AppId/Version/
is recommended, and there should be a root schema file that includes all data types in the namespace.

4.1.2 Component Versioning

For simple and complex data types that do not declare a target namespace, the version number should be used in the file name. For instance, a definition for County Identity should be named as:
  SC_CountyIdentity_v1.0.xsd
  SC_CountyIdentity_v2.0.xsd

It can be included into other schema using the following include statement:

```
<xsd:include
schemaLocation="http://www.exchangenetwork.net/schema/sc/SC_CountyI
denty_v2.0.xsd" />
```

Note that the schemaLocation has a specific file name in this case, which indicates the version of the schema to be included. Simple and complex data types that do not declare a target namespace are versioned by filename. All versions will be located in the **http://www.exchangenetwork.net/sc/** directory.

This recommendation makes it easier for developers to locate SSCs. However, it also avoids a proliferation of directories and minimizes the maintenance required by developers when upgrading minor versions.

For application specific components that have declared a target namespace, the namespace URI must contain a version number, just as the normal data flow schemas. Such schema can be imported using the following import statement:

```
<xsd:import namespace="
http://www.exchangenetwork/schema/AQS/geo/1" schemaLocation="
http://www.exchangenetwork/schema/AQS/geo/1" />
```

The application specific component is versioned as a whole because the namespace URI needs to be changed whenever a single schema file is modified.

5   Constructing Instance Documents

Instance documents on the EN should be constructed using the Network Namespace Recommendation. Instance documents that do not declare a target namespace are not permitted under this recommendation.

5.1   Use of the `schemaLocation` Attribute

The `schemaLocation` attribute in an instance document is optional. If used, it should point to a schema in the XML registry for network-wide document exchanges, as shown in the following example:

```
<MyRootElement xmlns=
"http://www.exchangenetwork.net/schema/example/1",
schemaLocation="http://www.exchangenetwork.net/schema/example/1">
    …
</MyRootElement>
```

Since resolvable URL-formatted namespaces are being introduced with the namespace recommendation, in most instances the `schemaLocation` attribute will be redundant in instance files. As in the preceding example, the `schemaLocation` element is exactly the same as the namespace URL. The CDX node and the network validation server ignore the `schemaLocation` attribute in instance documents. It is recommended that all applications on the Exchange Network do the same. The `schemaLocation` could be an ingredient for schema poisoning and a security hazard (it could point to a virus file or other malicious contents).

Another potential issue with the `schemaLocation` attribute is that tools generate XML instance documents using local schemas and automatically insert a `schemaLocation`:

```
<MyRootElement xmlns=
"http://www.exchangenetwork.net/schema/example/1",
schemaLocation="c:\mySchemas\aqs.xsd">
    …
</MyRootElement>
```

Note that the `schemaLocation` points to the user's local hard drive. The document will become invalid as soon as it leaves the computer.

5.2   Using Default Namespace

In almost all data flows, an instance document has to use schemas in different namespaces, so the question is which namespace should be used as the default. Although the choice may not affect the correctness of the document, it could have dramatic impact on the size of the document.

Elements in the default namespace do not need to be qualified with a prefix. The best strategy is to choose a namespace that has the most elements in an instance document. In other words, selecting a namespace that governs the main part of the XML document as the default namespace allows developers to create leaner and cleaner files.