software AG

Tamino

XML Namespaces in Tamino

Version 10.1

April 2018

WEBMETHODS

This document applies to Tamino Version 10.1 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: INS-NAMESPACE-101-20180413

Table of Contents

Preface

This document explains the rules and concepts for the use of *XML Namespaces* in Tamino 4.2 in various areas, mainly the *XML Schema*-based *schema* definition language *TSD4* and the query languages *X-Query* and *XQuery*. For reference, please see **http://www.w3.org/TR/REC-xml-names/**.

This document is intended to be read by administrators, application developers and other schema creators.

This document contains the following sections:

General Information on Namespaces	A general description of what namespaces are and what they are good for
Namespace-Related Facilities in XML Schema and in TSD4	The basic facilities related to namespaces in XML Schema and in *TSD4*.
Upgrade and Migration	*Migrating* from earlier versions and updating
Namespace Handling for Specific XMachine Requests	The consequences of the introduction of namespaces on *collection* names and *doctype* names

This documentation uses a lot of namespace specific terminology that might be unfamiliar to some readers. For detailed explanations of the terms used, please refer to the Tamino glossary.

1 General Information on Namespaces

What are Namespaces?

Namespaces were introduced into *XML* to avoid name clashes when multiple vocabularies are used. The concept of namespaces allows us to mix language elements from multiple vocabularies, such as SVG and XHTML, by adding namespace prefixes. It also allows us to define schemas with *XML Schema*, because by using namespace prefixes we can separate XML Schema tags and our own element names.

Namespace identifiers must be globally unique - usually a *URI* is used for that purpose. Defining a namespace within an XML document is simple: a document node is given an `xmlns` *attribute* to define the default namespace. Similarly, additional namespaces can be introduced by defining namespace prefixes using namespace declaration attributes of the form `xmlns:`*`prefix=URI`*. The scope of such a definition is the node where it is defined plus all child nodes (child elements and attributes), unless a child element overrides it with another namespace declaration. So, if we declare namespaces in a document's root element, their scope is usually the whole document.

The most important difference between *TSD4* and*TSD3* is that *TSD4* supports *XML Namespaces*. This section explains why and how you should use namespaces.

Note: TSD3 and TSD4 make use of that separation feature to differentiate between XML Schema parts that are marked with the `xs:` prefix, and the Tamino-specific extensions that are marked with the `tsd:` prefix.

The name of an element or attribute is called “qualified” if it contains a prefix and is within the scope of a corresponding namespace declaration. In addition, only elements without a prefix are affected by the default namespace declaration. The lexical (or pseudo) *QName* - an abbreviation of qualified name - is just the name string *`prefix:localname`*. The prefix including the colon is optional. See ***XML Schema, Part 2***. The expanded (or standard) QName is the associated “tuple” (URI, local name) derived from the QName and the namespace declaration for the prefix. For an unqualified name, the URI part of this “tuple” is empty.

What are QNames?

Let us start with a definition of the terms for QNames (qualified names):

[expanded or standard] QName:
An *expanded QName* or *standard QName* is a tuple `(URI,localName)` which may also be derived from a string *`prefix`*`:`*`localName`* in conjunction with a namespace declaration like `xmlns:`*`prefix=URI`* or a *namespace context* (see below).

For unqualified elements or attributes the URI may be empty.

[pseudo or lexical] QName:
A *pseudo QName* or *lexical QName* is a string *`prefix`*`:`*`localName`* without a namespace context or declaration . The prefix may be empty.

2 Namespace-Related Facilities in XML Schema and in TSD4

This chapter discusses the following topics about namespace usage in XML Schema and TSD4:

The targetNamespace Attribute of the xs:schema Element

One major advantage of *XML Schema* compared to DTDs is that XML Schema fully supports *XML Namespaces*. In order to do so, XML Schema introduces the concept of the target namespace. Each schema document can declare at most one target namespace, and all definitions made in this schema document belong to this namespace. Thus, a schema may contain namespace-less definitions, or definitions belonging to the specified target namespace.

Does this really mean that we cannot define multi-namespace schemas? No; the emphasis is here on the schema document. We can always compose multi-namespace schemas by importing (see below) other schema documents into a schema, see the description of the `xs:import` element.

Schema Composition via xs:import

The `xs:import` statement is used in XML Schema to compose multi-namespace schemas. A typical example is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema ↩
targetNamespace="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
    xmlns="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:i="http://www.softwareag.com/tamino/doc/examples/models/instruments">

  <xs:import ↩
namespace="http://www.softwareag.com/tamino/doc/examples/models/instruments"
            schemaLocation="instruments.xsd"/>

...
</xs:schema>
```

`xs:import` statements are always given at the beginning of a schema definition. An `xs:import` statement usually specifies a namespace to import, and may optionally specify a *URL* using the `schemaLocation` *attribute*. Note that this attribute only gives a hint to the *XML* processor, telling it where to find the schema document associated with the imported namespace. XML processors are not required to use this attribute, but may use their own logic to find the associated schema. Tamino supports only relative URLs in the `schemaLocation` attribute in `xs:import`. The referenced schemas must already be defined in the Tamino database when defining the referencing schema.

import (with schemaLocation)

The structure of the `schemaLocation` attribute may be any of the following alternatives:

- *../collectionName/schemaName*
- *./schemaName*
- *schemaName*

The last two alternatives are equivalent. Note: The `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instances of a schema (or *doctype*) are ignored by Tamino. Instead, a different logic based on collections and doctypes is applied to locate the schema used for validating the XML instance.

Example

The following schema shows the simultaneous use of the `namespace` and `schemaLocation` attributes of the `<xs:import>` element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    ↩
targetNamespace="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
   xmlns:e="http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
    xmlns:i="http://www.softwareag.com/tamino/doc/examples/models/instruments"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xs:import
     namespace="http://www.softwareag.com/tamino/doc/examples/models/instruments"
     schemaLocation="instruments.xsd"/>

  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="jazzMusician">
        <tsd:collection name="music"/>
        <tsd:doctype name="e:jazzMusician"/>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="jazzMusician">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="name" type="e:tName"/>
          ...
          <xs:element name="plays" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:choice>
```

```
                <xs:element ref="i:saxophone"/>
              </xs:choice>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
        ...
      </xs:complexType>
    </xs:element>
    ...
</xs:schema>
```

This example schema defines a global element named `jazzMusician`, and local elements `name` and `plays`. The local elements are qualified because of the `elementFormDefault="qualified"` attribute. All the elements defined in the schema belong to the namespace given by the `targetNamespace` attribute.

The `xs:import` element imports a schema with definitions for the *http://www.software-ag.com/tamino/doc/examples/models/instruments* namespace. The content model of element `plays` refers to an element named `saxophone` in the imported namespace.

The imported schema might look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    targetNamespace = ↩
"http://www.softwareag.com/tamino/doc/examples/models/instruments"
    xmlns:i   = "http://www.softwareag.com/tamino/doc/examples/models/instruments"
    xmlns:xs  = "http://www.w3.org/2001/XMLSchema"
    xmlns:tsd = "http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">

  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="instruments.xsd">
        <tsd:collection name="music"/>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="saxophone">
    ...
  </xs:element>
  ...
</xs:schema>
```

Usage of the `xs:import` element is subject to various constraints with respect to:

- the `targetNamespace` of the involved schema documents;
- usage of the `namespace` attribute;

- usage without the `schemaLocation` attribute.

Constraints (of xs:import statement)

The following constraints must be taken into account when using the `xs:import` element:

- If there is no `namespace` attribute present, the enclosing `xs:schema` element must have a target namespace declaration.
- If the `xs:import` element contains a `namespace` attribute, it may not match the value of the `targetNamespace` attribute of its enclosing `xs:schema` element.

Namespace of Elements and Attributes

Elements and attributes defined as child nodes of the `xs:schema` root element of a schema document are called global elements and attributes. If the `xs:schema` element has a `targetNamespace` attribute, all the global elements and attributes defined in that schema document are qualified, i.e. they belong to that namespace.

Once declared in this way, a global element or attribute can be used by referencing it in declarations (using the `ref` attribute of `xs:element` or `xs:attribute`). References to global elements or attributes are not restricted to a single schema; rather, they may refer to objects in a different namespace by using the prefix mapped to the `targetNamespace` of the imported schema in which the global object is defined.

By contrast, declarations of elements or attributes that are not immediate descendants of the `xs:schema` root node and do not just reference a global element or attribute definition via `ref="..."` are called local elements or local attributes. They can be defined e.g. inside a local or global (i.e. named) type. By default, these local nodes do not belong to the namespace given by the `targetNamespace` unless the corresponding element or attribute definition has a `form="qualified"` attribute. The default of `form="unqualified"` can be superseded by the `elementFormDefault="qualified"` or `attributeFormDefault="qualified"` attributes of the `xs:schema` element. These attributes specify whether the corresponding elements or attributes in the XML instance documents are qualified or not, if the `form` attribute is not present.

Global vs. Local Elements and Attributes

Namespace aspects of global and local elements and attributes in **XML Schema** are addressed in detail in the section **Advanced Concepts I: Namespaces, Schemas & Qualification** of the W3C **XML Schema Part 0: Primer**.

Wildcards

Another mechanism to allow for foreign namespaces is the usage of wildcards. A wildcard (i.e. an element or attribute whose content is not defined further) can be declared with the XML Schema elements `xs:any` or `xs:anyAttribute`. This allows for the inclusion of elements and attributes from foreign namespaces. For example, sections of XHTML, SVG, RDF or other content could be included into a document. A typical application is the description property in the style asset of our “Jazz” model, where we could use XHTML to mark up the content.

It is possible to constrain the namespace of the content of such a wildcard. This is done with the namespace attribute of the wildcard. This attribute can contain either:

- a whitespace-separated list of the following items:
  - explicit namespace URIs;
  - the string "`##targetNamespace`", which denotes the target namespace of the current schema file;
  - the string "`##local`", which denotes the absence of a namespace;
- or one of the following string values:
  - the string "`##any`", which allows for any namespace. This is also the default value of the namespace attribute. This value is often used together with `processContents="skip"`;
  - the string "`##other`", which allows for any namespace other than the target namespace.

Qualified Type References

References to types must be qualified with the namespace prefix bound to the namespace in which the referenced type is defined.

Note: This applies to both XML Schema predefined types and user-defined global types.

For example, the importing schema above contains a reference to a type `e:tName`, which might be defined as follows:

```
<xs:complexType name="tName">
  <xs:sequence>
    <xs:element name="firstName" type="xs:string"/>
    <xs:element name="middleName" type="xs:string" minOccurs="0"/>
    <xs:element name="lastName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

In the same way, references to a model group or attribute group definition in the same or a different schema document must use a prefix bound to a namespace that is identical to the respective schema's `targetNamespace`.

3 Upgrade and Migration

xml:lang and xml:space

In contrast to previous Tamino versions, both attributes, `xml:lang` and `xml:space`, must be explicitly declared in a schema. This can be done as follows:

```
<xs:import
  namespace="http://www.w3.org/XML/1998/namespace"
  schemaLocation="../ino:collection/xml_lang_space" />
```

4 Namespace Handling for Specific X-Machine Requests

This section contains information about the following:

Valid Collection and Doctype Names in TSD3/TSD4

Collection Names

As in *TSD3*, a *collection* name is a value of type `xs:NMTOKEN`. The name of a collection identifies it uniquely within the entire database.

Doctype Names

In *TSD4*, a *doctype* name is a QName, with the restriction that both the lexical and the *expanded QName* must be unique within the collection. The uniqueness of the lexical QName within the collection, effectively being just a plain string, is important especially for plain *URL* addressing (see *X-Machine Programming*).

Namespace Context of Doctypes

Each doctype in a Tamino database has a namespace context. From a logical point of view, the namespace context is a table describing a unique *mapping* of prefixes to URIs. It is extracted from the doctype's schema document(s) from the following locations:

- namespace declarations in the corresponding `tsd:doctype` element (used for that doctype only);
- namespace declarations in the `xs:schema` root element of the schema document containing the tsd:doctype element (used for all doctypes defined in that schema document);
- namespace declarations in the `tsd:collection` document, for the collection in which the doctype is contained. This document allows you to assign properties to a collection, e.g. the namespace context or whether the schemaless storage of documents is permitted in that collection. It can be stored in Tamino using the `_DEFINE` command. Please see below for an example. The namespace declarations in this document affect all doctypes in the collection.

Any prefix defined at one of these locations goes into the namespace context. The order in which the places are listed reflects their priority; for example, a namespace declaration given in the `tsd:doctype` element overrides a namespace declaration given in the `xs:schema` root element. For collections allowing for schemaless storage of documents, there may not be a schema document. Hence, only the third location applies.

Note: As the namespace context is crucial especially for the behavior of existing applications developed with previous versions of Tamino, namespace declarations contributing to the namespace context of any doctype may not be modified or removed when updating a schema.

Example:

Consider the following schema:

```
   <xs:schema
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
     xmlns:p1="URI1"
     xmlns:p2="URI2"
     targetNamespace="URI1" >
     <xs:annotation>
       <xs:appinfo>
         <tsd:schemaInfo name="mySchema">
           <tsd:collection name="myCollection"/>
           <tsd:doctype name="p1:root"/>
           <tsd:doctype name="p1:other" xmlns:p2="URI2a" xmlns:p3="URI3" ↩
xmlns:p5="URI5"/>
         </tsd:schemaInfo>
       </xs:appinfo>
     </xs:annotation>
   ...
   </xs:schema
```

with the following collection object:

```
   <tsd:collection name="myCollection"
     xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
     xmlns:p2="URI2b"
     xmlns:p3="URI3a"
     xmlns:p4="URI4"
   />
 ↩
```

The namespace context for the doctype with the lexical QName `p1:root` is:

Prefix	URI
p1	URI1
p2	URI2
p3	URI3a
p4	URI4

For doctype `p1:other`, the namespace context is:

Prefix	URI
p1	URI1
p2	URI2a
p3	URI3
p4	URI4
p5	URI5

The namespace context of a doctype is used for different purposes:

- Provide mapping of prefixes to URIs for *X-Query* requests (see the `_XQL` and `_DELETE` commands).

_XQL, _DELETE

Lexical QNames within `_XQL` and `_DELETE` requests are translated to expanded QNames before further processing. This translation is based on the namespace context of the corresponding doctype. If a query, e.g.

```
_XQL=/*/p:name
```

does not uniquely identify the doctype, it is first transformed to a representation where each expression refers to a unique doctype within the current collection.

Assuming the collection contains doctypes `dt1`, `dt2` and `dt3`, the query above can be transformed to:

```
_XQL= /dt1/p:name | /dt2/p:name | /dt3/p:name
```

Then, the prefix `p:` can be mapped to a *URI* for each of the doctypes `dt1`, `dt2`, and `dt3`, based on the corresponding namespace context. Note that the URI to which `p:` is mapped may differ between the doctypes. If a prefix in the request cannot be mapped based on the namespace context, it is assumed that there is no matching *QName* in the documents. If a lexical QName does not have a prefix, it is assumed that it does not belong to any namespace at all.

Default Namespace

A special case occurs when a doctype with the schema's `targetNamespace` attribute is associated with the default namespace in the respective schema document. Consider the following schema:

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
  xmlns="URI"
  targetNamespace="URI" >
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name="mySchema">
        <tsd:collection name="myCollection"/>
        <tsd:doctype name="root"/>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema
```

The lexical QName of the doctype defined in the schema is just `root` without any prefix. Its expanded QName is `(URI,root)`. In order to distinguish the doctype's lexical QName from lexical QNames corresponding to expanded QNames not belonging to any namespace, a prefix mapped to “URI” may be introduced via the `tsd:collection` document. For example:

```
<tsd:collection name="myCollection"
     xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
       xmlns:p="URI" />
```

maps the prefix "`p:`" to "`URI`". After this document has been stored in Tamino via `_DEFINE`, a query `_XQL=p:root` can be used to query all instances.

Plain URL Addressing

Plain URL addressing is based on a doctype's lexical QName, which is required to be unique within the scope of a collection. Instances within the doctype are identified either by their `ino:id` (a unique number) or by their document name. Neither method depends on namespaces.

XQuery

XQuery requests sent to Tamino (using the `_XQUERY` command) are based on the W3C's XQuery draft. They must specify a URI for each prefix used in the request. Hence, the namespace context of a doctype is not needed.

Special Handling of the Prefix ino:

ino:id

The `ino:` namespace prefix can be used in various contexts. If no namespace declaration is in scope, the following is assumed:

```
xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
```

The `ino:` prefix may not be bound to any other URI.

Note: This is not correct for XQuery.

When executing a `_PROCESS` command, Tamino checks for the presence of an `ino:id` attribute in the root element. If present, the request is interpreted as an attempt to replace the original document identified by that `ino:id` (if any; otherwise an error occurs). Any prefix other than `ino:` does not lead to interpretation as a request for replacement of the document — even when mapped to the same URI — and the document is rejected.

`_XQL` queries and `_DELETE` requests may use the `ino:` prefix for the pseudo-attributes `ino:id` and `ino:docname` without the need for the `ino:` prefix to be mapped via the doctype's namespace context.

Security

Instances of doctype `ino:acl` in collection `ino:security` are used to configure access to specific collections, doctypes and nodes within doctypes based on simple path expressions given inside the `ino:ace` element(s). A valid entry for the previous music schema might be:

```
<ino:acl
  xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
  ino:ac lname="myAcl">
  <ino:ace
    xmlns:e = "http://www.softwareag.com/tamino/doc/examples/models/jazz/encyclopedia"
     xmlns:i = "http://www.softwareag.com/tamino/doc/examples/models/instruments"
     ino:access="change">/e:jazzMusician/e:plays/i:saxophone</ino:ace>
</ino:acl>
```

Index