Infrastructure Series

TechDoc

WebSphere Message Broker / IBM Integration Bus

XML Namespaces
(Message Flow Development)

Author(s): Glen Brumbaugh

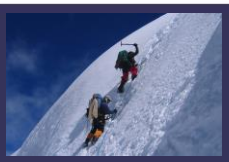

Table of Contents

Introduction

Document Version

This document describes XML Namespaces and their uses within a Message Broker (WMB up to v8.x) or Integration Node (IIB v9.0 +). This document should, however, apply to most versions of these products. The contents of this document have been specifically verified on the following production versions:

- WebSphere Message Broker v7.0.0.2
- IBM Integration Bus v9.0.0.0

This documentation has been created and maintained by:

- Glen Brumbaugh

This documentation has been reviewed by:

- Glen Brumbaugh

This documentation was last updated:

- Version 1.0 April 2015

XML Namespaces

Overview

XML (Extensible Markup Language) documents are groupings of hierarchical elements. These elements are given names by the document's developer. When different documents are used in combination, it is possible for names to "collide", a way of saying that the same element name exists in both documents. Any reference to this name would then be potentially ambiguous. The purpose of namespaces is to remove this ambiguity.

To remove the ambiguity, a "prefix" can be assigned to each element in the XML document. Different elements are then associated with their unique prefixes and any potential ambiguity is removed because each element now can have a unique name in the format of "prefix:element". By convention, and for brevity, these prefix names are generally of a very short length, sometimes only several characters. The resulting XML looks like this:

- <Tag1>*data*</Tag1> (An XML element named "Tag1")
- <ns:Tag1>*data*</ns:Tag1> (An XML element named "Tag1" in namespace "ns")

An XML "namespace" must be defined for each prefix that is needed. A namespace is simply a container for a set of XML element names and their corresponding hierarchy. The namespace has two attributes:

- Prefix (Name of namespace for use in XML)
- URI (Ensuring that the namespace is unique)

Note that the namespace itself has no separate name! The namespace is instead identified by its URI (Uniform Resource Identifier), which is itself unique, and is referenced in XML by its prefix. If two different XML documents with different namespaces use the same prefix, then one of the

prefixes would need to be modified if the documents needed to be combined. The resource indicated in the URI does not have to exist. For commercially defined namespaces, this URI is frequently a URL (Uniform Resource Locator) that points to a web page containing information about the namespace.

Note: The URI is NOT used by the parser to look up any information!

It is the responsible of the XML creator to ensure the uniqueness of the namespace URI. This, of course, depends upon the envisioned scope of usage for the XML. XML designed for use within a Corporation or single project has a far different requirement for uniqueness than XML designed for use across the entire web.

Uniform Resource Indicators (URI)

A Uniform Resource Indicator (URI) identifies a Web resource. A URI may have one of the following two formats:

- Uniform Resource Locator (URL) - Describes the location of a resource
- Uniform Resource Name (URN) - Describes the name of the resource

Note that both URIs and URLs must be unique within their own spaces. Also remember that these will not normally be "real" locations or names. When defining namespaces, the use of a URL versus a URN is a matter of personal preference. There are valid arguments for the use of either syntax. The general consensus seems to be that URLs are more widely understood.

The format of a URL is generally well understood due to its everyday use in Web Browsers. A URL is by definition unique and has the following format:

- url=*protocol*:*address*
  - *protocol* = Transport protocol. While there are over a dozen protocols defined for a URL, most people are primarily familiar with the 'HTTP' and 'FTP' protocols.
  - : = Separates the protocol from the address. Most users are familiar with the '://' syntax of the HTTP and FTP protocols. The two slashes are actually part of the address specification while the colon is the separator between the protocol and the address.
  - *address* = Defines the destination address to reach via the specified protocol.

Remember that the namespace URI is intended to uniquely identify a namespace, not to provide access to a specific website. An example of a URI in URL format is as follows:

- http://www.*MyCompany*.com/Namespaces/*MyApplication*/*MyXML*
  - *MyCompany*/ = The use of a primary domain means that the URI only needs to be unique within that domain.
  - Namespaces/ = The use of the 'Namespaces/' subdomain groups all of the namespaces together.
  - *MyApplication*/ = The use of a separate subdomain for applications or projects further helps to separate and group namespaces.
  - *MyXML*/ = This uniquely identifies the XML document. With the preceding domain and subdomains.

URNs are unfamiliar to most people and, while a formally defined syntax, there is debate within the Internet community about the relationship between URIs, URLs, and URNs. Since an URN is a URI, it must be unique. The format for specifying a URN is as follows:

- urn=*nid*:*nss*
  - *nid* = Namespace Identifier (1-31 alphanumeric or hyphen characters. Must begin with an alphanumeric character). Note there are some Namespace Identifiers ('ietf', 'issn', 'oasis', etc.) that have been formally registered with the IANA (Internet Assigned Numbers Authority).
  - *nss* = Namespace Specific String (Unlimited length character string. May contain alphanumeric characters and a number of special characters. Special characters include ':', '-', '_', '$', '+', '@', '!', '*', as well as parenthesis, commas, and semicolons).

Remember that the namespace URI is intended to uniquely identify a namespace, not to provide access to a specific website. An example of a URI in URL format is as follows:

- namespaces:*MyCompany*:*MyApplication*:*MyXML*
  - namespaces = Defines a "Namespace Identifier" named "namespaces". All namespaces will share this identifier.
  - *MyCompany*: *MyApplication*: *MyXML* = This defines the Namespace Specific String, which uniquely identifies the XML document. Note that this is purely a localized hierarchy; there is no hierarchy defined in the URN specification.

Defining a Namespace

A namespace is defined by an "xmlns" (XML Namespace) attribute in the start tag of any element. While frequently defined in the root element for readability and consistency, *xmlns* attributes may appear in any element. The scope of the namespace is thus determined by where it is defined within the XML structure. The namespace, and its prefix, are associated both with the element within which it is defined as well as all of the child elements. The syntax for defining a namespace is:

- <Element xmlns:*ns*="*URI*" … > </Element>
  - xmlns (Identifies an "xmlns" attribute)
  - : (Separates attribute name from prefix)
  - *ns* (Prefix associated with namespace)
  - =" (Specifies beginning of namespace URI)
  - *URI* (Namespace URI)
  - " (Specifies end of namespace URI)

Multiple *xmlns* attributes, and hence namespaces and their prefixes, may be defined within an element. Children elements may then include the appropriate prefix in their element names. Namespaces must be defined before their prefixes are referenced.

Note: Prefixes beginning with "xml" (in any combination of upper and lower case) are reserved and should not be defined!

Default Namespaces

Defining a "default" namespace eliminates the need for using prefixes within the scope of the namespace. A default namespace is declared in the same way that a normal namespace is declared

(with the "xmlns" attribute), but the ":prefix" portion of the syntax is left out. This creates a "null" prefix that does not need to be specified. This syntax is:

- <Element xmlns="*URI*" … > </Element>

TargetNamespaces

Namespaces are sufficient within a base XML document. XML Schema documents (.xsd), however, describe the required format of an actual XML document. Since these Schema documents are themselves XML documents, the Namespaces within those documents refer to the schema elements, not to the "target" elements of the actual XML document. In order to associate the "target" elements of the actual XML document with a namespace, a "target" namespace also needs to be defined.

A target namespace is defined as an attribute ('targetNamespace') of an element just as a namespace is ('xmlns'). In fact, if a target namespace is to be defined, it must be defined in the same element that the namespace is defined in. The values specified in both the *xmlns* and *targetNamespace* attributes are URIs.

The syntax for defining a target namespace is:

- <Element targetNamespace="*NameSpace*" xmnls= … > </Element>
    - targetNamespace (Identifies a target namespace attribute)
    - *NameSpace* (Namespace URI)

Note: If a target namespace is defined in a schema, then the same namespace must be referenced in the target XML document. This means that the URI specified in the *targetNamespace* attribute in the schema must be referenced in the *xmlns* attribute of the base XML document.

SchemaLocations

If an XML Schema has been defined, its location can be referenced in the base XML document. This allows the base XML document to be verified against its schema. The location reference is also an element attribute ('schemaLocation') and is defined in the same element as the *xmls* namespace attribute.

The *schemaLocation* attribute as two parts, separated by a space. The first part is the URI of the namespace. This URI is the same URI referenced in the *targetNamespace* and *xmlns* attributes. The second part, following the space, is the location where the schema is defined. This location is specified relative to the location of the base XML document.

Using Namespaces in ESQL

Namespaces are normally defined at the start of the main module so that the prefixes can be used instead of the full URI of the namespace. These namespace declarations only affect ESQL statements, they have no affect on the prefixes generated in the output message. Namespaces within ESQL are declared as follows:

- DECLARE *prefix* NAMESPACE '*URI*';

Once these ESQL namespaces have been declared, the prefixes associated with the namespace can be used to qualify element names with either the Input or Output trees. These qualifications will have the following syntax:

- *prefix*:*element* (Prefix and Element names are separated by a colon)

Namespaces in the output tree, if required, are controlled by namespace declarations placed in the output tree. These declarations use the 'XML.NamespaceDecl' field type. These declarations can define either a "default" or a "prefixed" namespace in the output tree. These namespace declarations are inserted into the output tree as follows:

- SET Outputroot.XMLNS. … .*Element*.(XML.NamespaceDecl)xmlns = '*URI*'
- SET Outputroot.XMLNS. … .*Element*.(XML.NamespaceDecl)xmlns:*prefix* = '*URI*'

The first example above declares a default namespace without a prefix. The second example declares a prefixed namespace. Two important notes about these statements:

(1) The prefix defined with the 'XML.NamespaceDecl' is never referenced in the ESQL! Only the prefix defined in the "DECLARE" statement is used when referring to elements in ESQL.

(2) The URIs defined in the "DECLARE" and corresponding "XML.NamespaceDecl" statements must be the same URI!

Using Namespaces in XPath Expressions

Namespaces are also used within Message Broker in XPath expressions. For instance, the "Route" node uses XPath expressions. XPath expressions can be built by hand or with the assistance of the Toolkit XPath expression editor. If the expressions require a namespace prefix, then the XPath editor must be used to define the namespace(s).

Namespaces are used within XPath expressions in the identical manner that they are used in ESQL statements: a "prefix" is added to an element name and the two are separated by a colon (':'). Again, note that an XPath prefix can only be associated with a Namespace by using the XPATH editor! Within the editor, the "Namespace settings" section can be used to define namespaces, and their prefixes, that will be used in the XPath expressions.

Best Practices

- **Use a Namespace prefix for every element in an XML document:** If there is only one namespace in the document, then define a default namespace in the root element. This means that every element without a prefix belongs to that default namespace.

- **Use a Default Namespace whenever possible:** If most, or even many, of the elements in an XML document belong to the same namespace, then define a default namespace in the root element. Then only elements not belonging to the default namespace will need to be prefixed.

- **When practical, declare Namespaces in the root element:** This makes the namespaces easy to locate. It also makes it possible to reorder portions of the XML document without impact. Ambiguities within the document should always have prefixes to define them and should not depend upon their location within the document.

- **Create Short Prefixes:** Keep the prefixes as short as possible while still being meaningful. This will enhance readability by reducing the overall size of the document. Remember that prefixes will be used many times.

- **Create Unique Prefixes:** Keep the prefixes as unique as possible. Naturally this practice is somewhat at odds with keeping the prefixes short. The tradeoff here involves considering the scope of the usage for the XML.

- **Use Meaningful Prefixes:** When multiple namespaces are involved, readability is improved if the prefix can be easily related to the namespace.

- **Use W3C Prefixes:** For example, when creating an XSLT Stylesheet, use the "xsl" prefix defined by the W3C to identify XSLT elements. This eliminates confusion.

- **Ensure URI Uniqueness:** While any "true" URI is unique (e.g. an actual URL), the values used for namespaces frequently do not represent real locations or names. Try to come up with a URI naming convention that ensures uniqueness. Consider using the URN format to avoid confusion over what the URI actually represents.

- **Do not version Namespaces:** ESQL code and XPath statements have these namespaces embedded within the code. Changing a namespace "name" (i.e. URI) will mean that all code using that namespace will have to be modified. To maintain backward compatibility, add an additional namespace reference (and prefix) inside the existing namespace to identify any new hierarchies or attributes.

- **Create a "version" attribute where the namespace is defined:** This attribute can then be queried, if required, to obtain version information.

References

- www.w3.org – XML (1.0) – Namespaces
  http://www.w3.org/TR/xml-names/

- www.w3.org – XML (1.1) – Namespaces
  http://www.w3.org/TR/xml11/

- W3schools – XML – Namespaces
  http://www.w3schools.com/xml/xml_namespaces.asp

- IBM - developerWorks – Using XML Namespaces
  http://www.ibm.com/developerworks/library/x-nmspace.html

- XML.org – Best Practices
  http://lists.xml.org/archives/xml-dev/200108/msg01286.html

- XML.com – Extensibility and Versioning Practices
  http://www.xml.com/pub/a/2004/07/21/design.html