

# Namespaces

**Sang Shin**

**JPassion.com**

**“Learn with JPassion!”**



# Agenda

- Need for Namespaces
- Namespace syntax
- Default Namespace
- Target Namespace and Source Namespace
- Importing XML schema with schemaLocation

# **Need for Namespaces**

# Need for Namespaces

- A XML document could use **multiple XML vocabularies**
  - > Possibility of name collision
- Examples
  - > XHTML document might contain XML elements from both SVG and MathML
  - > Both SVG and MathML have “set” element
- So how to avoid **Name collisions?**
  - > Use namespace

# Need for Namespaces

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

```
<catalog>
```

```
<RDF>
```

```
<Description about="http://ibiblio.org/examples/impressionists.xml">
```

```
<!-- title of a webpage -->
```

```
<title> Impressionist Paintings </title>
```

```
<creator> Elliotte Rusty Harold </creator>
```

```
<description>
```

```
  A list of famous impressionist paintings organized  
  by painter and date
```

```
</description>
```

```
<date>2000-08-22</date>
```

```
</Description>
```

```
</RDF>
```

# Need for Namespaces (continued)

```
<painting>
```

```
  <!-- title of a painting -->
```

```
  <title>Memory of the Garden at Etten</title>
```

```
  <artist>Vincent Van Gogh</artist>
```

```
  <date>November, 1888</date>
```

```
  <description>
```

```
    Two women look to the left. A third works in her garden.
```

```
  </description>
```

```
</painting>
```

```
<painting>
```

```
  <title>The Swing</title>
```

```
  <artist>Pierre-Auguste Renoir</artist>
```

```
  <date>1876</date>
```

```
  <description>
```

```
    A young girl on a swing. Two men and a toddler watch.
```

```
  </description>
```

```
</painting>
```

```
<!-- Many more paintings... -->
```

```
</catalog>
```

# Need for Namespaces

- Changing element names (to avoid collision) is not a convenient option
  - > Especially if you are not the owner
- Some collisions are inevitable
  - > If both are standard vocabularies
    - > SVG's "set" vs. MathML's "set"
- Grouping names is useful anyway
  - > XSLT processor needs to know which are XSLT instructions and which are result-tree element

# Need for Namespaces

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<catalog>

<rdf:RDF xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#">
  <rdf:Description xmlns:dc="http://purl.org/dc/"
    about="http://ibiblio.org/examples/impressionists.xml">
    <dc:title> Impressionist Paintings </dc:title>
    <dc:creator> Elliotte Rusty Harold </dc:creator>
    <dc:description>
      A list of famous impressionist paintings organized
      by painter and date
    </dc:description>
    <dc:date>2000-08-22</dc:date>
  </rdf:Description>
</rdf:RDF>
```



# Namespace Syntax

# Namespace Syntax

- Two parts
  - > Namespace declaration
  - > Elements and attributes

# Namespace Declaration

- A *prefix* is associated with **URI**
- The association is defined as an attribute within an element
  - > *xmlns:prefix*
- *xmlns* is Namespace keyword, prefix is user-defined

```
<classes xmlns:XMLclass="http://www.brandeis.edu/rseg-0151-g">  
  <XMLclass:syllabus>  
    ...  
  </XMLclass:syllabus>  
</classes>
```

# Namespace Declaration

- Can be declared in a root element or at lower level element
- Multiple different namespaces can be defined in a single document
- Same prefix can be redefined within a same document
  - > Scope of Namespace declaration is within the element where it is defined

# Elements and attributes with Namespace prefix

- Examples
  - > XMLClass:syllabus
  - > svg:set
  - > mathml:set
- *prefix: local part*
  - > prefix identifies the namespace an element or an attribute belongs to
  - > local part identifies the particular element or attribute within the namespace
  - > Together makes up a Qualified name

# Elements and attributes with Namespace prefix

- Prefix
  - > Can be composed from any legal XML name character except the “:”
  - > “xml” (in any case combination) is reserved so cannot be used as prefix
- Local part
  - > Cannot contain “:”

# Namespace URI

- URI cannot be prefix
  - > “/”, “%”, and “~” are not legal in XML element names
- URI could be **standardized** (by industry standard orgs) while prefixes are just convention
- URI are just “identifiers”
  - > URI does not have to be in “http” form
  - > URI does not have to be resolved
  - > It is like a “constant value”

# Default Namespace



# Default Namespace

- Declared with *xmlns* attribute with **no prefix**
- Applied only to unprefixed element and its descendant elements

# Default Namespace

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:xlink="http://www.w3.org/1999/xlink">
  <head><title>Three Namespaces</title></head>
  <body>
    <h1 align="center">An Ellipse and a Rectangle</h1>
    <svg xmlns="http://www.w3.org/2000/svg"
         width="12cm" height="10cm">
      <ellipse rx="110" ry="130" />
      <rect x="4cm" y="1cm" width="3cm" height="6cm" />
    </svg>
    <p xlink:type="simple" xlink:href="ellipses.html">
      More about ellipses
    </p>
    <p xlink:type="simple" xlink:href="rectangles.html">
      More about rectangles
    </p>
    <hr/>
    <p>Last Modified May 13, 2000</p>
  </body>
</html>
```

# **Example XML Documents**

# Maven pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.javapassion.examples</groupId>
  <artifactId>mvc_basics_helloworld1</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>mvc_basics_helloworld1 Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <org.springframework-version>3.0.4.RELEASE</org.springframework-version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!-- Spring framework -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${org.springframework-version}</version>
    </dependency>

    <!-- Spring MVC framework -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${org.springframework-version}</version>
    </dependency>

  </dependencies>
  <build>
    <finalName>mvc_basics_helloworld1</finalName>
  </build>
</project>
```

# HelloService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService" targetNamespace="http://helloservice.org/wsdl"
  xmlns:tns="http://helloservice.org/wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns2="http://helloservice.org/types">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://helloservice.org/types"
        schemaLocation="http://helloservice.org/types/HelloTypes.xsd"/>
    </xsd:schema>
  </types>
  <message name="Hello_hello">
    <part name="parameters" element="ns2:HelloRequest"/>
  </message>
  <message name="Hello_helloResponse">
    <part name="result" element="ns2:HelloResponse"/>
  </message>
  <portType name="Hello">
    <operation name="hello">
      <input message="tns:Hello_hello"/>
      <output message="tns:Hello_helloResponse"/>
    </operation>
  </portType>
  <binding name="HelloBinding" type="tns:Hello">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="hello">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="HelloService">
    <port name="HelloPort" binding="tns:HelloBinding">
      <soap:address location="http://helloservice.org/Hello"/>
    </port>
  </service>
</definitions>
```

# Schema Namespaces

# Types of Namespaces

- target Namespace
  - > Namespace for the XML schema document itself
- source Namespaces
  - > Namespaces external to the XML schema document

# targetNamespace

- It is the namespace that is going to be assigned to the schema you are creating
  - > The names of elements and attributes defined in a schema are said to belong to its target namespace
- It is the namespace a document instance uses to access the types it declares



# targetNamespace

- Each schema has one target namespace and possibly many source namespaces

# Example 1: Sample Schema

```
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:ACC="http://www.SampleStore.com/Account">  
  <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>  
  <xsd:element name="Product" type="ACC:ProductCode"/>  
  <xsd:simpleType name="ProductCode">  
    <xsd:restriction base="xsd:string">  
      <xsd:pattern value="[A-Z]{1}d{6}"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:schema>
```

# Example 1: Explanation

- The targetNamespace name is <http://www.SampleStore.com/Account>, which contains the *InvoiceNo*, *ProductID*, and *ProductCode* names in its namespace
- The *schema*, *element*, *simpleType*, *pattern*, *string*, and *positiveInteger* belong to source namespace <http://www.w3.org/2001/XMLSchema>
- The targetNamespace also happens to be one of the source namespaces because the name *ProductCode* is used in defining other names.

# Importing a Schema with `schemaLocation`

# Example 2: Sample Schema with no schema location

```
<!-- This is the same schema you saw in Example1 -->  
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:ACC="http://www.SampleStore.com/Account">  
  <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>  
  <xsd:element name="Product" type="ACC:ProductCode"/>  
  <xsd:simpleType name="ProductCode">  
    <xsd:restriction base="xsd:string">  
      <xsd:pattern value="[A-Z]{1}d{6}"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:schema>
```

# Why schemaLocation?

- The schema file uses several source namespaces but how does it know where to get the schema files of the source namespaces?

## Example 2: Explanation

- Example 2 (same schema as Example 1) does not need to specify locations of source schema files
  - > For the overall "schema of schemas," *http://www.w3.org/2001/XMLSchema*, you need not specify a location because it is well known
  - > For the source namespace *http://www.SampleStore.com/Account*, you do not need to specify a location since it also happens to be the name of the target namespace that is being defined in this file.

# Example 3: Schema with schema location

```
<xsd:schema targetNamespace="http://www.SampleStore.com/Account"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ACC="http://www.SampleStore.com/Account"
  xmlns:PART="http://www.PartnerStore.com/PartsCatalog">
  <xsd:import namespace="http://www.PartnerStore.com/PartsCatalog"
    schemaLocation="http://www.ProductsStandards.org/repository/alpha.xsd"/>
  <xsd:element name="InvoiceNo" type="xsd:positiveInteger"/>
  <xsd:element name="Product" type="ACC:ProductCode"/>
  <xsd:simpleType name="ProductCode">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="[A-Z]{1}d{6}"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name="stickyGlue" type="PART:SuperGlueType"/>
</xsd:schema>
```



# Example 3: Explanation

- The PART namespace needs to be imported using the *import* declaration element whose *schemaLocation* attribute specifies the location of the file that contains the schema because
  - > Is not a well-known namespace
  - > Is not a targetNamespace

**Learn with Passion!**  
**JPassion.com**

