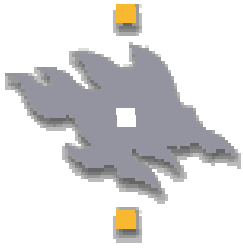


XML Schema Language

Suresh Chande



Recap

- XML Schema was introduced to address the limitation posed by DTD in defining the XML document structures and specially in data typing XML documents
- XML Schema is W3C recommendation dated: 2 May 2001
 - It consists of three specifications:
 - XML Schema structures
 - XML Schema data typing &
 - A Primer which will give an overview of XML Schemas

Root Element of XML Schema

<xsd:schema

xmlns:xsd=" <http://www.w3.org/2001/XMLSchema> "

targetNamespace=<http://www.sampledocumentschema.org>

elementFormData="qualified"

attributeFormData="qualified"

>

....

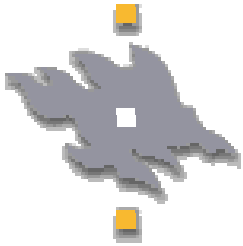
....

</xsd:schema>

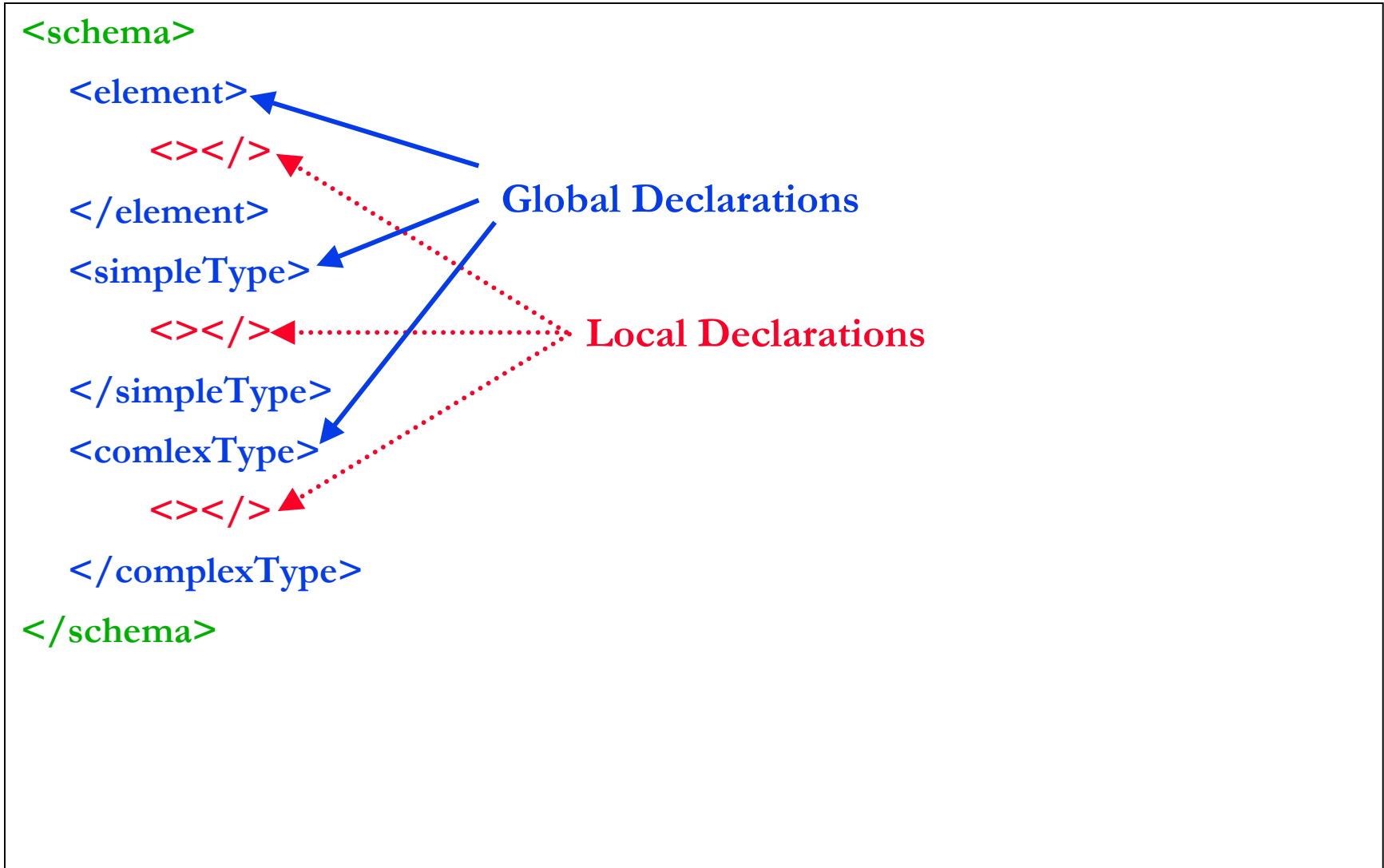
Element/Attributes names with a prefix of xsd belong to a namespace as declared here

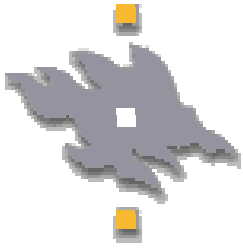
The namespace of this schema document, which any other instance document should refer to.

The qualified ensures that any instance document must contain namespace qualified element and attribute names.



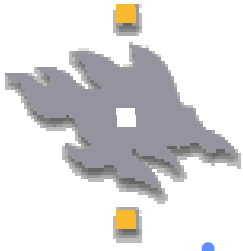
Global / Local Declarations





XML Schema Components

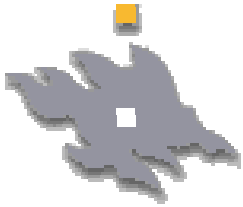
- XML Schema is built of a set of Components (Totalling to 13)
 - Simple type
 - Complex type
 - Attribute
 - Element
 - Attribute group
 - Identity-constraint
 - Model group
 - Notation
 - Annotations
 - Model groups
 - Particles
 - Wildcards
 - Attribute uses



Primitive dataTypes

- string
- base64Binary
- hexBinary
- integer
- decimal
- float
- double
- Boolean
- time
- dateTime
- duration
- date
- gMonth
- gYear
- gYearMonth
- gDay
- gMonthDay
- Name
- QName
- NCName
- anyURI
- language
- ID
- IDREF
- IDREFS
- NOTATION

19 Builtin Primitive Datatypes



Derived dataTypes

- `normalizedString` <- `string`
- `Token` <- `normalizedString`
- `Byte` <- `short`
- `unsignedByte` <- `unsigned short`
- `Integer` <- `decimal`
- `positiveInteger` <- `nonNegativeInteger`
- `negativeInteger` <- `nonPositiveInteger`
- `nonNegativeInteger` <- `integer`

- `nonPositiveInteger` <- `integer`
- `Int` <- `long`
- `unsignedInt` <- `unsignedLong`
- `Long` <- `integer`
- `unsignedLong` <- `nonNegativeInteger`
- `Short` <- `int`
- `unsignedShort` <- `unsignedInt`

- `Name` <- `Token`
- `NCName` <- `Name`
- `Language` <- `token`
- `ID` <- `NCName`
- `IDREF` <- `NCName`
- `IDREFS` <- `IDREF`
- `ENTITY` <- `NCName`
- `ENTITIES` <- `ENTITY`
- `NMTOKEN` <- `Token`
- `NMTOKENS` <- `NMTOKEN`

25 Derived datatypes

Schema Document – An Example

```
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace= "http://www.mobileservices.org/PhoneBook"
  xmlns:pb= "http://www.mobileservices.org/PhoneBook"
  elementFormDefault="qualified">

  <element name="phonebook">
    <complexType>
      <sequence minOccurs = "1" maxOccurs="unbound">
        <element ref=pb:Contact/>
      </sequence>
    </complexType>
  </element>

  <element name="Contact">
    <complexType> ON NEXT PAGE <
  </element>

</schema>
```

XML Schema Namespace as
default namespace



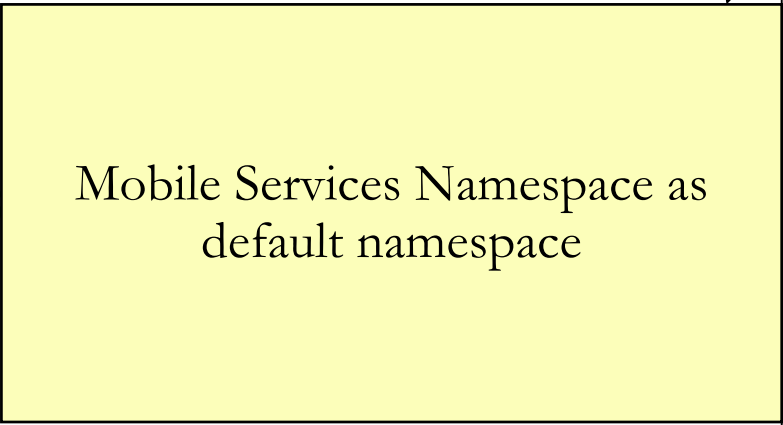
Schema Document – An Example

```
<xsd:schema xmlns:xsd = “http://www.w3.org/2001/XMLSchema”  
  targetNamespace= “http://www.mobileservices.org/PhoneBook”  
  xmlns= “http://www.mobileservices.org/PhoneBook”  
  elementFormDefault=“qualified”>
```

```
  <xsd:element name=“phonebook”>  
    <xsd:complexType>  
      <xsd:sequence minOccurs =“1” maxOccurs=“unbound”>  
        <xsd:element ref=Contact/>  
      </xsd:sequence>  
    </xsd:complexType>  
  </xsd:element>
```

```
  <xsd:element name=“Contact”>  
    <xsd:complexType> ON NEXT PAGE  
  </xsd:element>
```

```
</xsd:schema>
```



Mobile Services Namespace as
default namespace



Schema Document – An Example

```
<schema ....
```

```
  <element name="Contact">
    <complexType>
      <sequence>
        <element ref="pb:name"/>
        <element ref="pb:email"/>
        <element ref="pb:address"/>
      </sequence>
    </complexType>
  </element>
  <element name="name" type="string">
  <element name="email" type="string">
  <element name="address" type="string">
```

```
</schema>
```

We will use XML Schema Namespace as default namespace in the schema document for simplicity purpose.
NOTE: The recommendation is to use targetNamespace as the default namespace

OR It can also be defined as in next slide..



First Schema Documents

```
<schema ....  
  <element name="Contact">  
    <complexType>  
      <sequence>  
        <element name="name" type="string">  
        <element name="email" type="string">  
        <element name="address" type="string">  
      </sequence>  
    </complexType>  
  </element>  
</schema>
```

OR It can also be defined as in next slide..



First Schema Documents

```
<schema ....
```

```
  <element name="Contact" type="pb:contactType"/>
```

```
  <complexType name="contactType">
```

```
    <sequence>
```

```
      <element name="name" type="string">
```

```
      <element name="email" type="string">
```

```
      <element name="address" type="string">
```

```
    </sequence>
```

```
  </complexType>
```

```
</schema>
```



Abstract data Model of Schema Documents

- XML Schema document can be described as abstract data model made up of **13 schema components** which define a set of properties, rules for validation and constraints on the composition of the components.
- **Basic Schema components:**
 - Simple, complex definitions & Attribute, Element declarations
- **Secondary Schema Components:**
 - Attribute Group, IdentityConstraints, Model Group, Notation
- **Helper Components:**
 - Annotation, Model groups, Particles, WildCards, Attribute uses

Element Declaration

- This is the core of the Schema document which defines the element's content model, its attributes and as well as the datatypes and behaviour of the data types
- It is globally defined if it is a child element of the XML Schema root element "*schema*" and locally as child of a complexType
- An element declaration can contain : complexType, simpleType, Identity constraints(unique, key or keyref), annotations

SchemaDocument

```
<schema ...
  xmlns:ab="....." >
  <element name = "contact" type="string" minOccurs="1" maxOccurs="unbound">
<element name="phonebook">
  <complexType>
    <choice>
      <element ref="ab:email"/>
      <element name="snail_mail">
        <complexType>
          <sequence>
            <element name="street" type="string"/>
            <element name="city_and_pincode" type="string"/>
            <element name="country" type="string"/>
          </sequence>
        </complexType>
      </element>
    </choice>
  </complexType>
</element>
```

CONTINUED ON NEXT SLIDE...

NOTE: We could have multiple choices by defining minOccurs and maxOccurs attributes to Choice element

Some examples

Element

Attribute

SimpleType

ComplexType

```
<element name="email" type="ab:emailType">
  <simpleType name="emailType">
    <restriction base="string">
      <pattern value="*@*\.(com|org|net|gov|edu)">
    </restriction>
  </simpleType>
</schema>
```

Email addresses of the format as:

any characters@any characters.com
any characters@any characters.org
any characters@any characters.net
any characters@any characters.gov
any characters@any characters.edu



Element Declarations

Defaulting:

```
<element name="initialNumber" type="int" default="0"/>
```

Fixed:

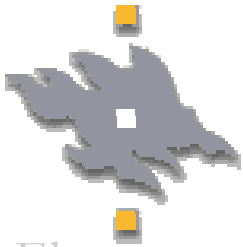
```
<element name="maximumSMSTextSize" type="int"  
  fixed="160"/>
```

Attribute declarations

- This can be defined as a child of *schema* or part of any other schema component
- Attribute defines the name, data type, element association, behaviour and a default or fixed value.
- It can contain: simpleType, Annotation

```
<attribute ...>
---
</attribute>
```
- Attribute group

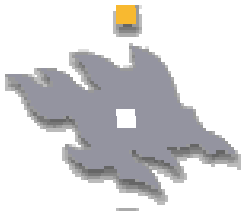
```
<attributeGroup name="someGroupName">
  att1, att2, att3 declarations
</attributeGroup>
```



Element
Attribute
SimpleType
ComplexType

Attribute declarations - Example

```
<schema ...  
<element name="unified_message">  
  <complexType>  
    <sequence>  
      <element name="to" type="prefix:addressType"/>  
      <element name="from" type="prefix:addressType"/>  
      <element name="priority" type="prefix:priorityType"/>  
    </sequence>  
  </complexType/>  
  <attribute ref="prefix:message_type">  
  <attributeGroup ref="session_attributes">  
</element> .....next slide
```

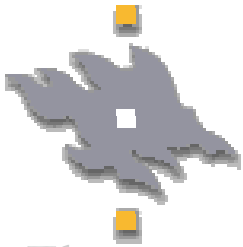


Element
Attribute
SimpleType
ComplexType

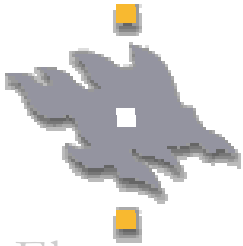
Attribute declarations - Example

```
<attribute name="message_type" use="required">
  <simpleType>
    <restriction base="string">
      <enumeration value="plain_text"/>
      <enumeration value="mms"/>
      <enumeration value="sms"/>
      <enumeration value="voice"/>
      <enumeration value="video"/>
      <enumeration value="animation"/>
    </restriction>
  </simpleType>
</attribute>

<attributeGroup name="sessionAttributes">
  <attribute name="id" type="ID" use="required"/>
  <attribute name="title" type="string" use="required"/>
  <attribute name="messageNumber" type="int" />
</attributeGroup>
</schema>
```



Element
Attribute
SimpleType
ComplexType



Element
Attribute
ComplexType
SimpleType

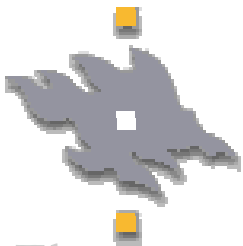
Complex Types Definitions

- **If an element should contain sub-elements , attributes.** It is used to define the content model
- **A ComplexType can either be a child of an element or the *schema*** (global type declarations)
- The Element can utilise the contentType definition to define its internal content model
- **A complex type can contain :**
 - simpleContent
 - complexContent
 - attribute
 - attributeGroup
 - anyAttribute
 - compositors(all, sequence or choice)
 - group
 - annotation

An Example

```
<complexType name="addressType">
  <sequence>
    <element name="street" type="prefix:streetType">
    <element name="city" type="string">
    <element name="pincode" type="prefix:pinCodeType">
    <element name="country" type="prefix:countryType">
  </sequence>
</complexType>

<element name="address" type="ab:addressType" use="required">
```

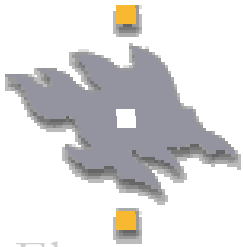


Element
Attribute
ComplexType
SimpleType

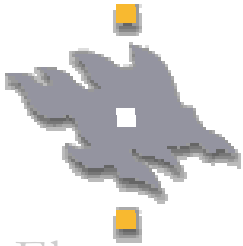
simple/complexContent

- complexContent is used to extend or restrict complexTypes, while the simpleContent is used for extending/restricting simpleTypes
- The Complex type can contain both of these content models
- For example :

```
<xsd:element name = "Designation">  
  <xsd:complexType>  
    <xsd:simpleContent>  
      <xsd:extension base = "xsd:string">  
        <xsd:attribute name = "title" use = "required" type = "xsd:string"/>  
      </xsd:extension>  
    </xsd:simpleContent>  
  </xsd:complexType>  
</xsd:element>
```



Element
Attribute
ComplexType
SimpleType



Element

Attribute

ComplexType

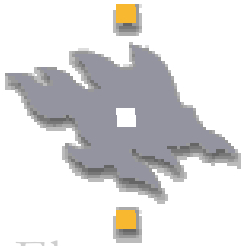
SimpleType

simple/complexContent

```
<xsd:complexType name = “shippingAddressType”>  
  <xsd:complexContent>  
    <xsd:extension base=“xsd:addressType”>  
      <xsd:element name=“state” use=“required” type=“xsd:string”/>  
      <xsd:attribute name=“priority” use =“required” type =“xsd:string”/>  
    </xsd:extension>  
  </xsd:complexContent>  
</xsd:complexType>
```

simpleType Definitions

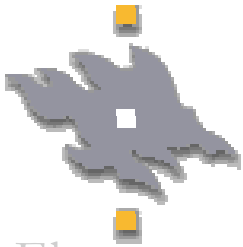
- These are used to define the datatypes for text contents in both elements and attributes.
- The simpleType can use the 44 data types to represent the data types or then can even derive data types from these base 44 data types.
- The derivation is supported by :
 - restriction
 - list
 - union



Element
Attribute
ComplexType
SimpleType

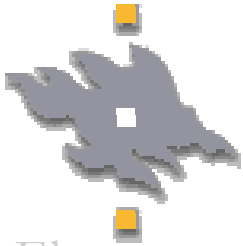
simpleType - Restriction

- Restriction is used to define a specific format of a primitive data type.
- Restrictions applies on top of base types and specifies a set of defined rigid rules on subsetting the base type. The restriction can apply 12 possible means of restricting the content model of the simpleType, namely:
 - *enumeration, fractionDigits, length, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, pattern, totalDigits, whitespace*



Element
Attribute
ComplexType
SimpleType

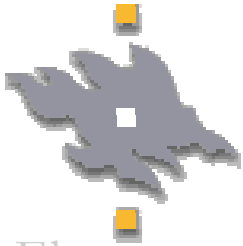
simpleType Restriction-Examples



Element
Attribute
ComplexType
SimpleType

```
<simpleType name="title">  
  <restriction base="string">  
    <enumeration value="Mr">  
    <enumeration value="Mrs">  
    <enumeration value="Ms">  
    <enumeration value="Dr">  
    <enumeration value="Master">  
  </restriction>  
</simpleType>  
  
<simpleType name="temperature">  
  <restriction base="decimal">  
    <totalDigits value="3"/>  
    <fractionDigits value="1"/>  
  </restriction>  
</simpleType>
```

simpleType Restriction-Examples

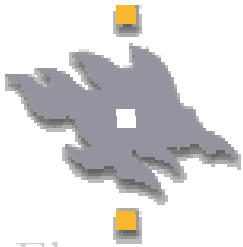


Element
Attribute
ComplexType
SimpleType

```
<simpleType name="smsText">  
  <restriction base="string">  
    <length value="160" fixed="true"/>  
  </restriction>  
</simpleType>
```

```
<simpleType name=" safeDrivingSpeed">  
  <restriction base="decimal">  
    <minInclusive value="20"/>  
    <maxExclusive value="121"/>  
  </restriction>  
</simpleType>
```

minLength, minExclusive, MaxInclusive, totaldigits & whiteSpace



Element
Attribute
ComplexType
SimpleType

simpleType Restriction-Examples

Pattern : For data formatting and is based on regular expression(similar to that of perl)

Few examples(W3C):

Chapter \d => Chapter 0, Chapter 1, Chapter 2

a*x => x, ax, aax, aaax

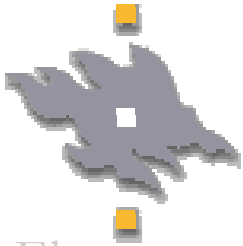
a?x => ax, x

a+x => ax, aax, aaax

(a | b)+x => ax, bx, aax, abx, bax, bbx, aaax, aabx, abax, abbx,
baax, babx, bbax, bbbx, aaaax

[abcde]x => ax, bx, cx, dx, ex

(ab){2}x => ababx



Element
Attribute
ComplexType
SimpleType

simpleType Restriction-Examples

```
<simpleType name="emailType">  
    <restriction base="string">  
        <pattern value="*@*\.(com|org|net|gov|edu)">  
    </restriction>  
</simpleType>
```

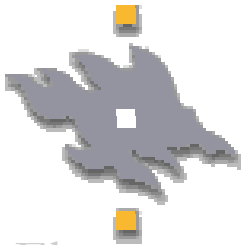
simpleType List & Union

- LIST contains a whitespace separated list of values of the type of base types as declared.

```
<simpleType name="participantsAges">  
  <list itemType="decimal"/>  
</simpleType>
```

- UNION helps in deriving a datatype based on of multiple datatypes

```
<simpleType name="conferenceParticipants">  
  <union memberTypes="projectManagerDLType,  
    softwareDeveloperDLType, softwareArchitectDLType"/>  
</simpleType>
```



Element
Attribute
ComplexType
SimpleType



XML Schema elements

- **Particles:** allows definition of occurrence behaviours (minOccurs, maxOccurs)
- **Wildcards:** If the instance document authors require to extend and introduce additional document structures then it can be accomodated by the schema definition within pre-determined portion of the documents by utilising *Wildcards* such as by using: "any".

For example:

```
<element name="diary">
  <complexType>
    <sequence>
      <element name="title">
        <element name="description">
          <any/>
        </element>
      </sequence>
    </complexType>
  </element>
```



XML Schema elements Contd..

- **NOTATION:** This is similar to that of DTD, but with the XML Schema syntax, this is used to define the location of the external non XML Data and the association of the external application which can handle such non-XML Data

```
<notation name="jad", public="application/midp2.0" system="j2me.jar">
```

- **Annotation:** This is used to provide additional information to the current data model for humans(documentation) and applications to utilise(appInfo)

```
<annotation> <documentation> Human readable text </documentation>  
</annotation>
```

```
<annotation> <appInfo>sessionId=2312; JsessionId=HSIWOO&%PXQ21sq</  
appInfo > </annotation>
```



Some Schema Guidelines

- Reuse as many declarations as possible, by declaring them as globals and referencing them as seen necessary. Utilise Include, Import and Redefine to reuse already defined schemas.
- Qualifying Namespace : Make the `targetNamespace` the default namespace of the schema document when specially the schema document is not referencing many different namespaces
- Global vs Local : Global when reuse is higher, Local when you need to hide the elements/structures definitions
- Namespaces can be depicted as a URN / URL(preffered by the web community)
- Versioning: Use the version in the root element “*schema*”, introduce a “*schemaVersion*” attribute for the root element, change the “*targetNameSpace*”

Best Practices Guidelines further reading: <http://www.xfront.com/BestPracticesHomepage.html>



Reuse of declarations

- The declarations and definitions of elements, types and structures can be reused by utilising the ”*ref*” attribute within elements.
- A Set of such element declarations can also be reused using Group definitions

- For example:

```
<group name=“collection”>  
  <sequence> | <choice> | <all>  
</group>
```

```
<element name=“playlist”>  
<complexType>  
  <group ref=“prefix:collection”/>
```

```
.....  
</complexType>  
</element>
```

- Other means of reuse are by Include, Import, Redefinition

Schema Re-use - IMPORT

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaA">...
</schema>
```

TargetNamespace A

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaB">...
</schema>
```

TargetNamespace B

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaC"
  xmlns:sa="http://www.sample.com/SchemaA">
  xmlns:sb="http://www.sample.com/SchemaB">
  <import namespace="http://www.sample.com/SchemaA"
    schemaLocation="SchemaA.xsd"/>
  <import namespace="http://www.sample.com/SchemaB"
    schemaLocation="SchemaB.xsd"/>
</schema>
```

TargetNamespace C

NOTE: Assembling from multiple schemas into a single schema.

Department of Computer Science

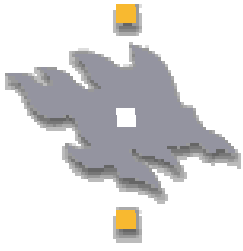
Schema Re-use - INCLUDE

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaC">...
</schema>
```

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaC">...
</schema>
```

```
<schema
  xmlns="http://www-w3.org/2001/XMLSchema" >
  targetNamespace=http://www.sample.com/SchemaC
  xmlns:sa="http://www.sample.com/SchemaA">
  xmlns:sb="http://www.sample.com/SchemaB">
  <include schemaLocation="SchemaA.xsd"/>
  <include schemaLocation="SchemaB.xsd"/>
</schema>
```

TargetNamespace C



Schema Reuse - redefinition

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" >
  targetNamespace="http://www.sample.com/SchemaA">...

  <element name="diary" type="diaryEntryType"/>
  ...
  <complexType name="diaryEntryType">
    <sequence>
      <element name="title">
        <element name="description">
          <any/>
        </element>
      </sequence>
    </complexType>
  ....
</schema>
```



Schema Reuse - redefinition

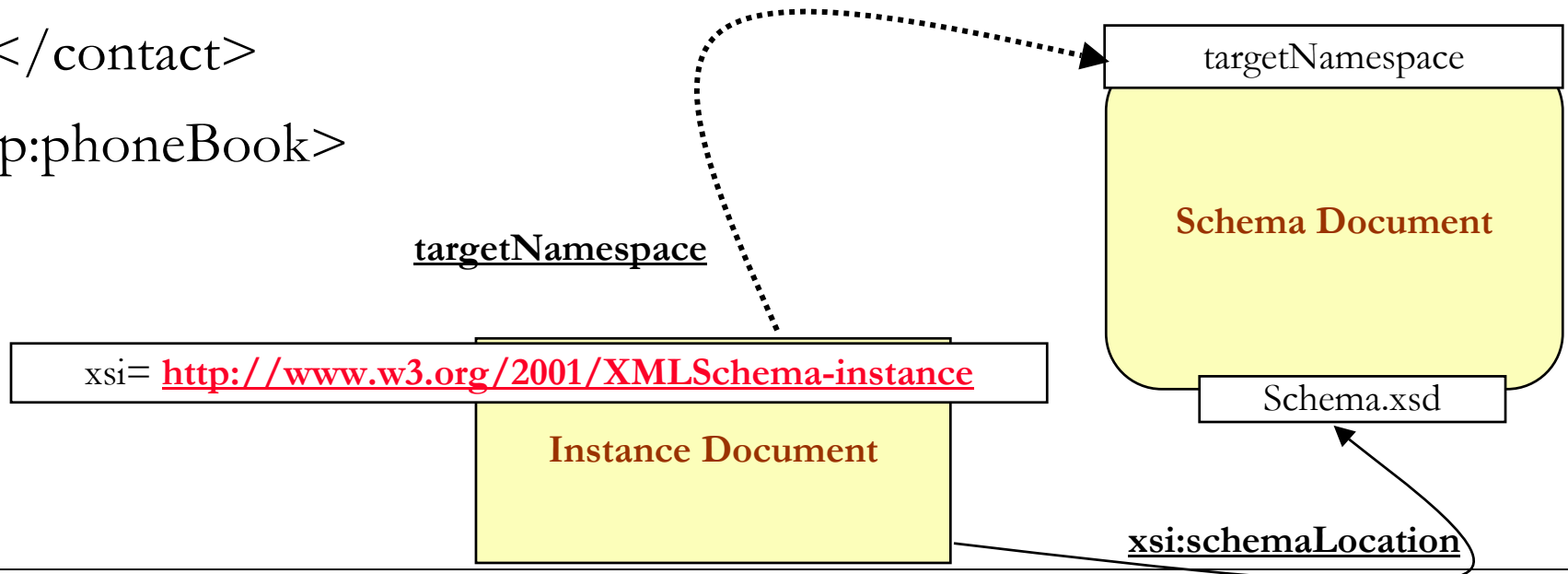
```
<schema xmlns="http://www.w3.org/2001/XMLSchema" >  
    targetNamespace="http://www.sample.com/SchemaB">
```

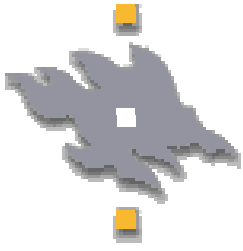
..

```
    <redefine schemaLocation="SchemaA.xsd">  
        <complexType name="diaryEntryType">  
            <complexContent>  
                <extension base="diaryEntryType">  
                    <sequence>  
                        <element name="startDate", type="date">  
                        <element name="endDate", type="date">  
                    </sequence>  
                </extension>  
            </complexContent>  
        </complexType>  
    </redefine>  
</schema>
```

Instance Document

```
<? xml version="1.0" ?>
<p:phoneBook xmlns:p="http://www.mobileservices.org/phoneBook"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.mobileservices.org/phoneBook.xsd">
  <contact ....
  </contact>
</p:phoneBook>
```





Further Reading

- Xml Schema Primer : <http://www.w3.org/TR/xmlschema-0/>
- *XML Schema Part 1: Structures* : <http://www.w3.org/TR/xmlschema-1/>
- *XML Schema Part 2: Datatypes* : <http://www.w3.org/TR/xmlschema-2/>
- A very good source for XML Schema Tutorial;
<http://www.xfront.com/xml-schema.html>
- Best Practices Guidelines further reading:
<http://www.xfront.com/BestPracticesHomepage.html>