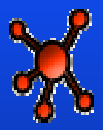
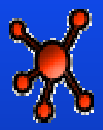


Web Services

XML Schemas

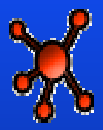
Whenever DTDs are not enough



Web Services

What are Schemas?

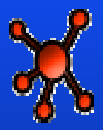
- Generically, a document that describes what a correct document may contain
- Specifically, a W3C Recommendation for an XML-document syntax that describes the permissible contents of XML documents



Web Services

What is an XML Schema?

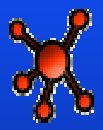
- Like a DTD, a schema defines what a given set of one or more XML documents can look like.
  - What elements, and in what order
  - Attributes and their type
  - etc.
  - etc.
- The XML document can then be validated against the designated schema.



Web Services

What Are Schemas for?

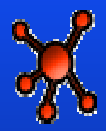
- Contracts: agreeing on formats
- Tool building: know what the data will be *before* the first instance shows up
  - Database integration
  - User interface tools
  - Programming language bindings
- Validation: make sure we got what we expected



Web Services

About Schemas

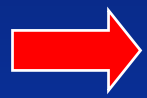
- Created by W3C XML Schema Working Group based on many different submissions
- No known patent, trademark, or other IP restrictions
- XML Schema Part 1: Structures: http://www.w3.org/TR/xmlschema-1/
- XML Schema Part 2: Datatypes: http://www.w3.org/TR/xmlschema-2/



Web Services

At First Glance

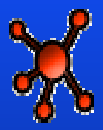
greeting.xml



greeting.xsd

```
<?xml version="1.0"?>
<GREETING>
  Hello XML!
</GREETING>
```

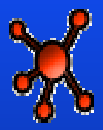
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=
"http://www.w3.org/2001/XMLSchema">
  <xsd:element
    name="GREETING"
    type="xsd:string"/>
</xsd:schema>
```



Web Services

What are the Advantages of Schemas over DTDs?

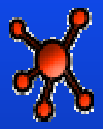
- Schemas can do everything a DTD can do, PLUS
  - They are written as well-formed XML documents
  - They offer full support for namespaces
  - Data can be validated based on built-in and user-defined data types
  - Programmers can more easily create complex and reusable content models
  - You can declare and use both local and global variables in the XML document



Web Services

Some Schema Aware Parsers

- Xerces-J 2.x: http://xml.apache.org/xerces2-j
- Xerces-J 1.4.4: http://xml.apache.org/xerces-j
- Xerces-C++ 1.7.0: http://xml.apache.org/xerces-c
- Oracle XML Parser for Java: http://technet.oracle.com/tech/xml/xdk_java/
- Oracle XML Parser for C: http://technet.oracle.com/tech/xml/xdk_c/
- Oracle XML Parser for C++: http://technet.oracle.com/tech/xml/xdk_cpp/



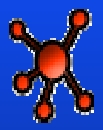
Web Services

Simple and Complex Types

- Simple types cannot have children or attributes
- Complex types can have child elements and attributes

- **Simple type** = Elements with only text
  Several built-in simple types of 'text', including:
  Date, integer, string, boolean and URL

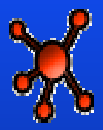
- **Complex type** = Elements with attributes and non-text elements:
  - elements that contain only other elements
  - elements that contain both elements & text
  - elements that contain only text
  - elements that are empty



Web Services

Four *Main* Schema Elements

- **`xsd:element`**
  declares an element and assigns it a type
- **`xsd:attribute`**
  declares an attribute and assigns it a type
- **`xsd:simpleType`**
  defines a new simple type
- **`xsd:complexType`**
  defines a new complex type



Web Services

Elements and Types: Example

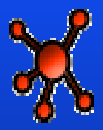
```
<xsd:element name="weight" type="xsd:string"/>
<xsd:element name="population" type="xsd:integer">
<xsd:simpleType name="zipcodeType">
  <xsd:restriction base="xsd:string">
  <xsd:pattern value="\d{5}(-\d{4})?"/>
  </xsd:restriction>
</xsd:simpleType>
```

Both simple and complex types can be

named - they can be used in other places throughout the schema

or

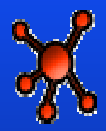
anonymous - used only within the element in which they are defined



Web Services

Local and *Global* Declarations

- Elements declared as child elements of the *xsd:schema* element are considered *global*
- Elements declared in other locations are *local* to the definition where they are declared
- When defining a complex type, you can **reference** *globally declared elements*, or declare and define new ones *locally*
- Locally declared elements can only be used in the complex type definition where they are declared.



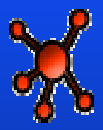
Web Services

Sample Code

```
<xsd:schema xlmns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="endangered_species" type="endType"/>
<xsd:element name="name" type="xsd:string"/>
<xsd:complexType name="endType">
   <xsd:sequence>
     <xsd:element name="animal">
     <xsd:complexType>
       <xsd:sequence>
         <xsd:element ref="name" minOccurs="1"
                                 maxOccurs="unbounded"/>
         <xsd:element name="source" type="sourceType" />
       </xsd:sequence
     </xsd:complexType>
   </xsd:sequence>
</xsd:complexType>
. . .
</xsd:schema>
```

Global

Local



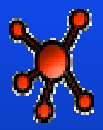
Web Services

Beginning an Embedded Simple Schema

At the top of the schema document

- Type the xml declaration
- Type the schema namespace declaration
- Your schema content will follow. . .
- Type the closing of the namespace declaration

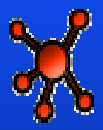
```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd=
  "http://www.w3.org/2001/XMLSchema">
  . . .
  </xsd:schema>
```



Web Services

Indicating a Simple Schema's Location

- Type: the xml document declaration
- Type: your_root_element
  Then, without closing the root element tag,
- Type: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
  Then type: xsi:noNamespaceSchemaLocation=
  Type "file.xsd" where "file.xsd" is the full url to your schema file



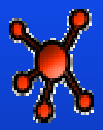
Web Services

Indicating a Schema's Location

```
<?xml version"1.0"?>
<cd
xmlns:xsi="http://ww.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation=
"http://www.iet.unipi/xml/schemas/cd.dtd">
```

- Use when you do NOT want to use the W3C schema namespace... Otherwise:

```
<stylesheet
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/1999/XSL/Transf
orm http://www.w3.org/1999/XSL/Transform.xsd
http://www.w3.org/1999/xhtml
http://www.w3.org/1999/xhtml.xsd">
```

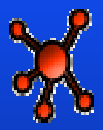


Web Services

Annotating Schemas

To add information about your schema or its elements...

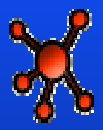
- Type: <xsd:annotation >
- Type: <xsd:documentation >
- Type: documentation_text
- Type: </xsd:documentation >
- Type: </xsd:annotation >



Web Services

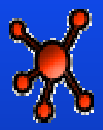
Sample Code

```
<?xml version="1.0"?>
  <xsd:schema xmlns:xsd=
  "http://www.w3.org/2001/XMLSchema">
  <xsd:annotation >
  <xsd:documentation >
  The CD schema will be used to validate the
  'Weird Al Yankovich' CD file used in the
  Beginning XML textbook (Wrox)
  </xsd:documentation >
  </xsd:annotation >
```



Web Services

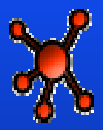
Simple Types



Web Services

Defining Simple Types

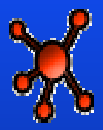
- An element with a simple type can contain only text.
  It may not contain other elements and it may not have attributes.
- Built-in simple types:
  1. String -
  2. Integer (numbers) -
  3. Boolean -
  4. Date -
  5. URL -
- Using 'facets', you can build custom simple types.



Web Services

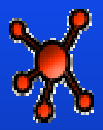
"Facets": Expressing Constraints

- Facets include:
  - length
  - minLength
  - maxLength
  - pattern
  - enumeration
  - whiteSpace
    (values:
    preserve,
    replace)
  - maxInclusive
  - maxExclusive
  - minInclusive
  - minExclusive
  - totalDigits (prev. precision)
  - fractionDigits (prev. scale)
- Not all facets apply to all types.



Declaring an Element with a Simple Type

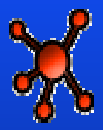
- Type <xsd:element
- Type name="label"' where label is the name of the element
- Type type="xsd:data_type" (choose a data type)
- Type /> to complete the tag
- Additional data types can be found at www.w3.com/TR/xmlschema-2/#built-in-datatypes



Simple Type Element Declarations

Web Services

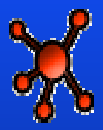
```
<xsd:element name="weight" type="xsd:string"/>
<xsd:element name="total" type="xsd:integer"/>
<xsd:element name="updated" type="xsd:date"/>
<xsd:element name="balance" type="xsd:decimal"/>
<xsd:element name="survived" type="xsd:boolean"/>
<xsd:element name="day_time" type="xsd:time"/>
<xsd:element name="id_num" type="xsd:ID"/>
<xsd:element name="EN" type="xsd:language"/>
<xsd:element name="day_time" type="custom"/>
```



Web Services

Common Date and Time Types

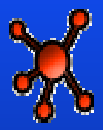
- xsd:timeDuration format=PnYnMnDTnHnMnS
- xsd:time format=hh:mm:ss.sss
- xsd:timeInstant format=CCYYMM-DDThh:mm:ss.sss
- xsd:date format=CCYY-MM-DD
- xsd:month format=CCYY-MM
- xsd:year format=CCYY
- xsd:century format=CC
- xsd:recurringDate format=--MM-DD
- xsd:recurringDay format=---DD



Web Services

Common Number Types

- xsd:decimal + or - content with finite number
- xsd:positiveInteger (1, 2, etc.)
- xsd:negativeInteger (-1, -2, etc.)
- xsd:nonPositiveInteger (0, -1, -2, etc.)
- xsd:nonNegativeInteger (0, 1, 2, etc.)
- xsd:float single precision 32 bit floating
- xsd:double double precision 64 bit floating

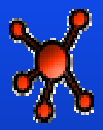


Web Services

Deriving Custom Simple Types

Custom allow you to expand on any of the built-in simple types.

- Type `<xsd:simpleType` to begin the definition
- Type `name="label">` where label will identify the new custom type (but not the element).
- Type `<xsd:restriction base="foundation">` foundation=simple type upon which it's based
- Type `<xsd:pattern value="pattern_here"/>`
- Type `</xsd:restriction>`
- Type `</xsd:simpleType>`

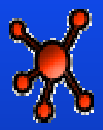


Web Services

Sample Code

```
<xsd:simpleType name="zipcodeType">
<xsd:restriction base="xsd:string">
<xsd:pattern value="\d{5}(-\d{4})?"/>
</xsd:restriction>
</xsd:simpleType>
```

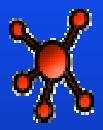
The pattern limits the content of these elements to 5 digits followed by an optional hyphen, and 4 additional digits.



Anonymous Custom Types - Example

[same as previous... but this time anonymous]

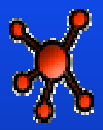
```
<xsd:element name="zipcode">
  <xsd:simpleType>
  <xsd:restriction base="xsd:string">
  <xsd:pattern value="\d{5}(-\d{4})?"/>
  </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```



Web Services

Specifying a Set of Acceptable Values

```
<xsd:element name="basicDirections">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="North"/>
      <xsd:enumeration value="South"/>
      <xsd:enumeration value="East"/>
      <xsd:enumeration value="West"/>
      <xsd:enumeration value="Up"/>
      <xsd:enumeration value="Down"/>
      </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

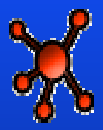


Specifying a Pattern

Web Services

You can use a regular expression (regex) to construct a pattern which content must match in order to be valid.

```
<xsd:element name="zipcode">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{5}(-\d{4})?"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```



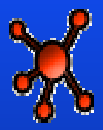
Web Services

Specifying a Range of Acceptable Values

- Type <xsd:maxInclusive (<xsd:minInclusive)
- Type value="n" (content must be less than or equal to 'n' to be valid)
- Type /> to complete the element.

OR...

- Type <xsd:maxExclusive (<xsd:minExclusive)
- Type value="n" (content must be less than but NOT equal to 'n' to be valid)
- Type /> to complete the element.

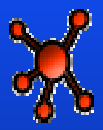


Web Services

Limiting the Length

- To specify the exact length of an element,
  Type <xsd:length value="x" />
- For minimum or maximum values:
  Type xsd:minLength value="m" />
  or xsd:maxLength value="n" />

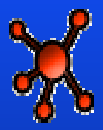
where x, m, and n are number of characters



Web Services

Limiting a Number's Digits

- Within the customType definition
  Type <xsd:totalDigits value="n" />
  n = maximum number of digits in a number
- To specify the number of digits to the right of a decimal,
  Type xsd:fractionDigits value="n" />

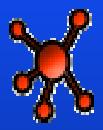


Web Services

Creating List Types

- Type <xsd:simpleType name="label"
  Where "label" is name of element declared
- Type <xsd:list itemType = "individual"
  Where 'individual' is the name of the simple type that defines each individual value of your list.
- Type </xsd:list>
- Type </xsd:simpleType>

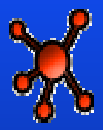
```
<xsd:simpleType name="datelist">
  <xsd:list itemType="original_list_type">
  </xsd:list>
  </xsd:simpleType>
```



Predefining an Element's Content

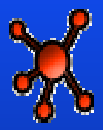
- To dictate an element's content:
  Within the element tag,
  Type fixed="value" - 'value' determines the content of the element

- To set an initial value for an element:
  Within the element tag,
  Type default="value" - 'value' determines the content of the element



Web Services

Complex Types



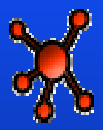
Web Services

Defining Complex Types

An element with a complex type can contain other elements and it may have attributes.

Possible "content models":

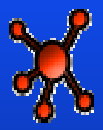
1. Empty - may contain attributes, but not elements or text
2. Text Only - may contain attributes
3. Elements Only - may contain other elements or attributes - no text
4. Mixed Content - may contain other elements or attributes or text



Content Models: Syntax

Web Services

MODEL	SYNTAX	SHORTHAND
Empty	<complexType mixed="false"> <complexContent/> </complexType>	<complexType />
Text only	<complexType> <simpleContent> </simpleContent> </complexType>	<complexType> <simpleContent> </simpleContent> </complexType>
Element only	<complexType mixed="false"> <element /> </complexType>	<complexType> <element /> </complexType>
Mixed	<complexType mixed="true"> <complexContent> </complexContent> </complexType>	<complexType> <complexContent> </complexContent> </complexType>

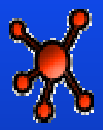


Web Services

Defining Empty Elements

'empty' means the element will have no content between the open and close tag. It may, though, contain attributes.

- Type <xsd:complexType name="label">
- Type <xsd:complexContent>
- Type <xsd:extension base="xsd:anyType">
- Declare the attributes (if any)
- Type </xsd:extension>
- Type </xsd:complexContent>
- Type </xsd:complexType>



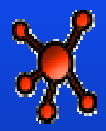
Web Services

Defining Elements to Contain Only Text

- Type <xsd:complexType name="label">
- Type <xsd:simpleContent>
- Type <xsd:restriction if the base simple type will be limited with additional facets.

  OR

- Type <xsd:extension if a simple type will be expanded.
- Type base="foundation"> 'foundation'=simple type upon which this is based.
- Type </xsd:restriction> or </xsd:extension>
- Type </xsd:simpleContent>
- Type </xsd:complexType>

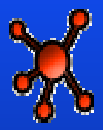


Web Services

Sample Code

```
<xsd:complexType>
  <xsd:simpleContent>
    <xsd:extension base="xsd:integer">
      <xsd:attribute name="year"
            type="xsd:year"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Use this if an element will contain only text, but might contain an attribute.

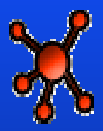


Web Services

Defining Elements to Contain Only Elements

- Type <xsd:complexType name="label"/>
  'label' is the name of the complexType
- Declare a sequence, choice, or unordered group, or reference a named group (we'll see in a while).
- Then declare or reference the attributes.
- Type </xsd:complexType> to end.

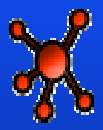
```
<xsd:complexType name="address_bookType">
  <xsd:sequence>
  <xsd:element name="record" type="recordType"/>
  </xsd:sequence>
</xsd:complexType>
```



Web Services

Requiring Elements to Appear in Sequence

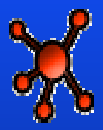
- Type <xsd:sequence
- If desired, specify how many times the sequence of elements itself can appear by setting minOccurs and maxOccurs attributes.
- Type > to close the tag.
- Declare or reference desired components in the order you want them to appear.
- Type </xsd:sequence > to end.



Web Services

Sequence: Example

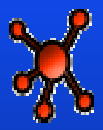
```
<xsd:complexType name="recordType">
  <xsd:sequence>
  <xsd:complexType element= "name" type="nameType" />
  <xsd:complexType name="address" type="addressType" />
  <xsd:complexType name="contact" type="contactType" />
  </xsd:sequence>
</xsd:complexType>
```



Web Services

Creating a Set of Choices

- Type <xsd:choice
- If desired, specify how many times the set of choices can appear by setting minOccurs and maxOccurs attributes.
- Type > to close the tag.
- Declare or reference desired elements that will make up the choices in the set.
- Type </xsd:choice > to end.

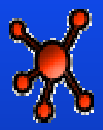


Web Services

Allowing Elements to Appear in Any Order

When you want to have an element be able to contain other elements in any order.

- Type <xsd:all> to begin the group.
- If desired, use minOccurs or maxOccurs attributes, may be used to set "how many".
- Only values of 0 or 1 may be used.
- An "all" group must be the sole child of a complex type definition or named group.

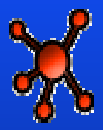


Web Services

Defining Named Groups

If a collection of elements appear together throughout an XML document, you can group them together to simplify the declaration.

- Type <xsd:group name="label"
  Where 'label' will identify this group
- Type > to close the group tag
- Declare the sequences, sets of choices, or unordered group that will make up the named group.
- Type </xsd:group> to end.



Web Services

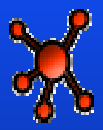
Referencing a Named Group

Once you've created a group, you can reference it in other groups or in complex type definitions.

- Type <xsd:group ref="label"
  Where 'label' matches the name of the group you created.
- Type > to close the group tag

```
<xsd:group ref="physical_traits"/>
<xsd:group ref="physical_traits"/>
```

The same reference can now be used as other element names are declared and defined.



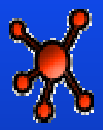
Web Services

Referencing Already Named Elements

Elements of both simple and complex type that are declared globally (just inside the xsd:schema element) must be called or *referenced* to appear in the XML document.

- Type <xsd:ement name="label"
- Specify minOccurs or maxOccurs if needed,
- Type /> to end the reference.

```
<xsd:element ref="name" minOccurs="2"/>
<xsd:element ref="name" minOccurs="1"/>
<xsd:element ref="name" maxOccurs="5"/>
```

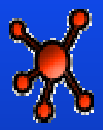


Controlling How Many

- In the element's or group's opening tag, Type minOccurs="n" to indicate the fewest number of times it must occur.

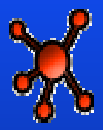
  OR

- Type maxOccurs="n" to indicate the maximum number of times it may occur.
- Type maxOccurs="unbounded" to indicate that the element may occur any number of times.
- Default value for both is 1.



Defining Elements with Mixed Content

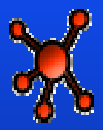
- Type <xsd:complexType name="label"
- Type mixed="true" to indicate the element can contain elements, attributes and may possibly contain text.
- Type > to close the opening tag.
- Declare a sequence, choice, etc...
- Declare or reference the attributes...
- Type </xsd:complexType>



Web Services

Sample Code

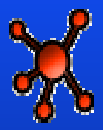
```
<xsd:complexType name="paragraph" mixed="true">
<xsd:sequence>
<xsd:element name="name" type="nameType"/>
</xsd:sequence>
<xsd:attribute name="length" type="xsd:string"/>
</xsd:complexType>
```



Web Services

Building Complex Types upon Complex Types

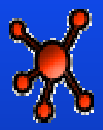
- Type <xsd:complexType name="label">
- Type xsd:complexContent>
- Type xsd:extension or xsd:restriction
- Type base="existing" (the name of the existing complex type)
- Type > to close the tag.
- Declare the changes you will make...
- Declare or reference any attributes...
- Type </xsd:extension> or </xsd:restriction>
- Type </xsd:complexContent>
- Type </xsd:complexType>



Complex from Complex: Example (I)

This is a dedinition of a complex type,
that will be used later within another complex type

```
<xsd:complexType name="characteristicsType">
  <xsd:sequence>
    <xsd:element name="weight" type="xsd:string"/>
    <xsd:element name="length" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="kind" type="xsd:string"/>
</xsd:complexType>
```

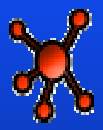


Web Services

Complex from Complex: Example (II)

```
<xsd:complexType name="birthType">
 <xsd:complexContent>
   <xsd:extension base="characteristicsType">
    <xsd:sequence>
    <xsd:element name="mother" type="xsd:string">
    <xsd:element name="birthdate" type="xsd:date">
    </xsd:sequence>
   </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

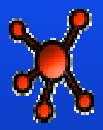
When you base a complex type on another complex type, you begin with all the information from the existing type, then add or remove features.
In this example, characteristicsType was previously defined.



Declaring an Element of Complex Type

Once a complex type is defined, it can be assigned to an element so that the element can then be used in the XML document.

- Type <xsd:element type="label"
  'label' must match the identifying word used when the complex type was defined.
- Type /> to close the tag.



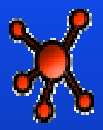
Web Services

Elements with Anonymous Complex Type

If you don't need to reuse a complex type, it may be faster to create an anonymous complex type within the element declaration.

- Type <xsd:element name="label">
- Type <xsd:complexType>
- Declare a sequence, choice, group, etc..
- Declare or reference any attributes...
- Type </xsd:complexType>
- Type </xsd:element>

```
<xsd:element name="characteristics">
   <xsd:complexType>
   <xsd:sequence>
   <xsd:element name="weight" type="xsd:string"/>
   <xsd:element name="length" type="xsd:date"/>
   <xsd:attribute name="kind" type="xsd:date"/>
   </xsd:sequence>
   </xsd:complexType>
   </xsd:element>
```

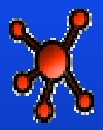


Web Services

Declaring Attributes

- Type <xsd:attribute name="label"
- Type type="simple" (simple type to which the attribute belongs)
  OR
- Type ref="label" (label = already declared global attribute)
- Add any restraining facets...
- Type /> to close the tag

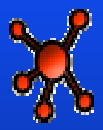
```
<xsd:element name="source">
<xsd:complexType>
<xsd:complexContent>
<xsd:extension base="xsd:anyType">
<xsd:attribute name="sectionid" type="xsd:string"/>
<xsd:attribute name="newspaperid" type="xsd:string"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:element>
```



Requiring an Attribute

Attributes default to optional. Add this to require that they be used.

- Type <xsd:attribute name="label"
- Type use="required"
- Type value="must" if you want to require a specific value.
- Add any other attribute declarations...
- Type /> to close the tag.
- Other possible values:
  Type use="prohibited" if you want the document to validate only if the attribute is NOT present.



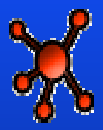
Web Services

Predefining an Attribute's Content

- Type <xsd:attribute name="label"
- Type type="xsd:string"
  (or other valid type)
- Type use="fixed" value="content"
  To set a fixed value,

  OR

- Type use="default" value="content"
  To set an initial value, which can be typed over.
- Type /> to close the tag.



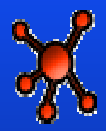
Web Services

Defining Attribute Groups

This allows you to use the same set of attributes in more than one element.

- Type <xsd:attributeGroup name="label">
- Declare or reference each attribute that belongs to the group.
- Type </xsd:attributeGroup> to end the definition.

```
<xsd:attributeGroup name="imageAtts">
<xsd:attribute name="filename" type="xsd:uri-
reference"/>
<xsd:attribute name="x" type="xsd:integer"/>
<xsd:attribute name="y" type="xsd:integer"/>
</xsd:attributeGroup>
```



Web Services

Referencing Attribute Groups

Once an attribute list has been declared, it is easy and very efficient way to reference the group from within the corresponding complex type

- Type <xsd:attributeGroup ref="label"/>

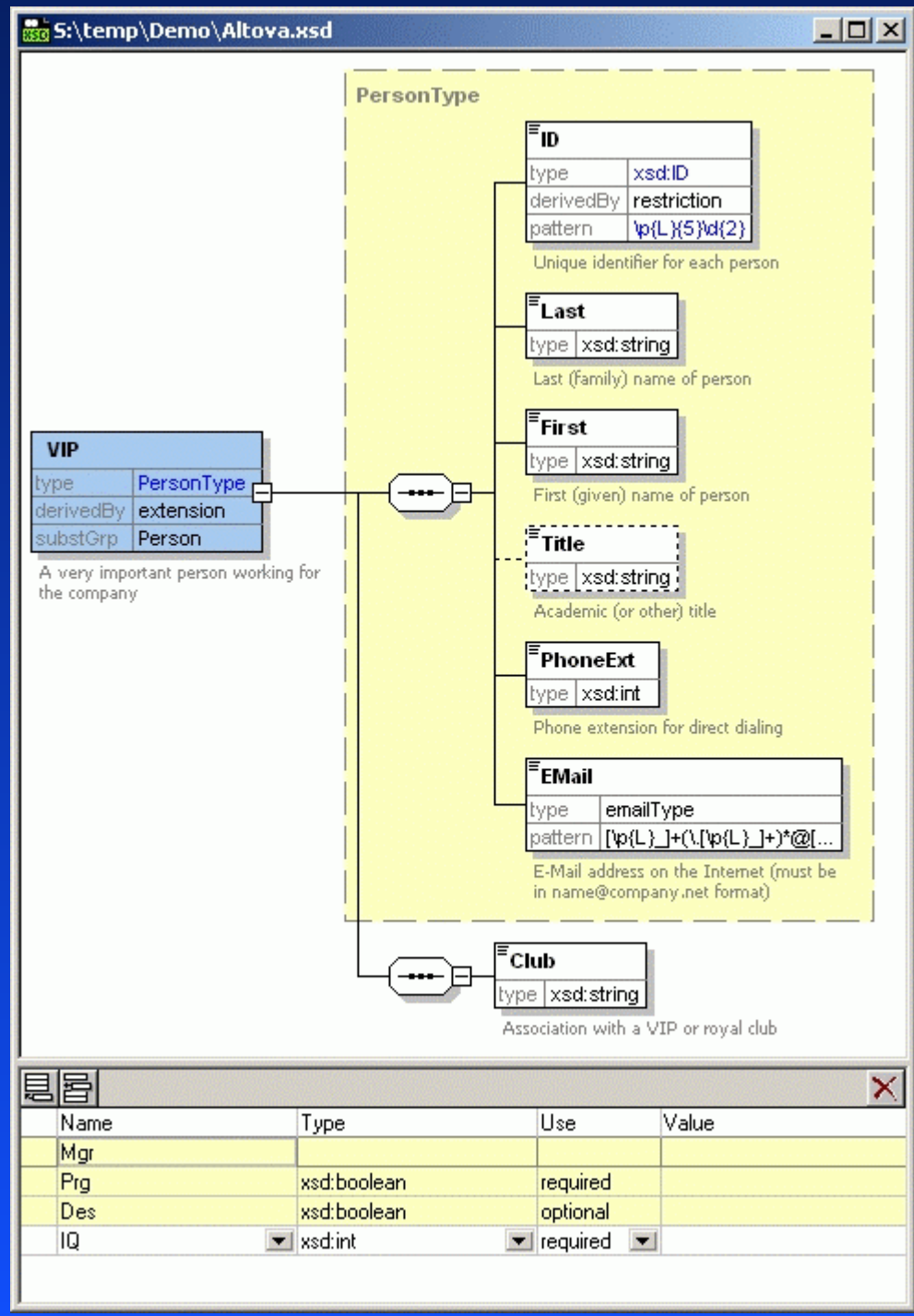
```
<xsd:element name="picture">
<xsd:complexType>
<xsd:complexContent>
<xsd:extension base="xsd:anyType">
<xsd:attributeGroup ref="imageAtts"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:element>
```

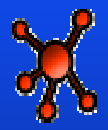
Web Services

Graphical Schema Editors

A number of graphical tools exist to help write XML schema;

Here it's shown a screenshot from XMLSPY, one of the most popular ones.





Web Services

Graphical Schema Editors

- Details
- Facets