

The purpose of XML (Extensible Markup Language) is to store, transport, and structure data in a format that is both software- and hardware-independent. Here's a detailed look at its primary purposes:

1. Data Storage and Transport

Storing Data: XML is designed to store data in a structured format, making it easy to manage and retrieve.

Transporting Data: XML facilitates the transport of data across different systems and networks, ensuring compatibility and consistency.

2. Self-Descriptive Data

Self-Descriptive: XML documents are self-descriptive, meaning the tags and structure provide context about the data, making it easier to understand and use.

3. Platform Independence

Software- and Hardware-Independent: XML is platform-independent, allowing it to work seamlessly across various software applications and hardware systems.

4. Separation of Data and Presentation

Separation: XML separates data from its presentation, enabling the same data to be displayed differently in various contexts without altering the data itself.

5. Interoperability

Incompatibility Solutions: XML helps solve compatibility issues between different systems by storing data in plain text format, making it easier to exchange data between incompatible or upgraded systems.

6. Extensibility

Flexible Structure: XML allows the addition or removal of data elements without breaking existing applications, providing flexibility to adapt to changing data requirements.

7. Complement to HTML

Complementary Use: XML is often used alongside HTML, where XML stores or transports data, and HTML is used to format and display the data.

8. Data Sharing and Availability

Simplifies Data Sharing: XML simplifies data sharing between different systems and applications.

Improves Data Availability: Ensures data can be accessed and used across different platforms and devices.

Examples and Features

Tree Structure: XML documents are formed as element trees, starting at a root element and branching out to child elements.

Tags and Attributes: Tags are used to define data elements, while attributes provide additional information about these elements.

XML-extensible markup language

- XML is a software- and hardware-independent tool for storing and transporting data.
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

- Example:

```
<note>
  <to>Jiya</to>
  <from>Tanisha</from>
  <heading>Meeting reminder</heading>
  <body>Hi!! I hope you remember we have a meeting this wednesday</body>
</note>
```

- The XML example below is quite self-descriptive:

- It has sender information
- It has receiver information
- It has a heading
- It has a message body

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Someone must write a piece of software to send, receive, store, or display it

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are
- The tags in the example on previous slide (like `<to>` and `<from>`) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
- HTML works with predefined tags like `<p>`, `<h1>`, `<table>`, etc.
- With XML, the author must define both the tags and the document structure

XML is Extensible

- Most XML applications will work as expected even if new data is added (or removed).
- Imagine an application designed to display the original version of note.xml (<to> <from> <heading> <body>).
- Then imagine a newer version of note.xml with added <date> and <hour> elements, and a removed <heading>.
- The way XML is constructed, older version of the application can still work:

```
<note>
  <date>2024-10-05</date>
  <hour>10:30</hour>
  <to>Jiya</to>
  <from>Tanisha</from>
  <body>Hi! I hope you remember we have a meeting this wednesday</body>
</note>
```

XML Simplifies Things

- XML simplifies data sharing
- XML simplifies data transport
- XML simplifies platform changes
- XML simplifies data availability
- Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.
- XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.
- XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

XML Separates Data from Presentation

- XML does not carry any information about how to be displayed.
- The same XML data can be used in many different presentation scenarios.
- Because of this, with XML, there is a full separation between data and presentation.

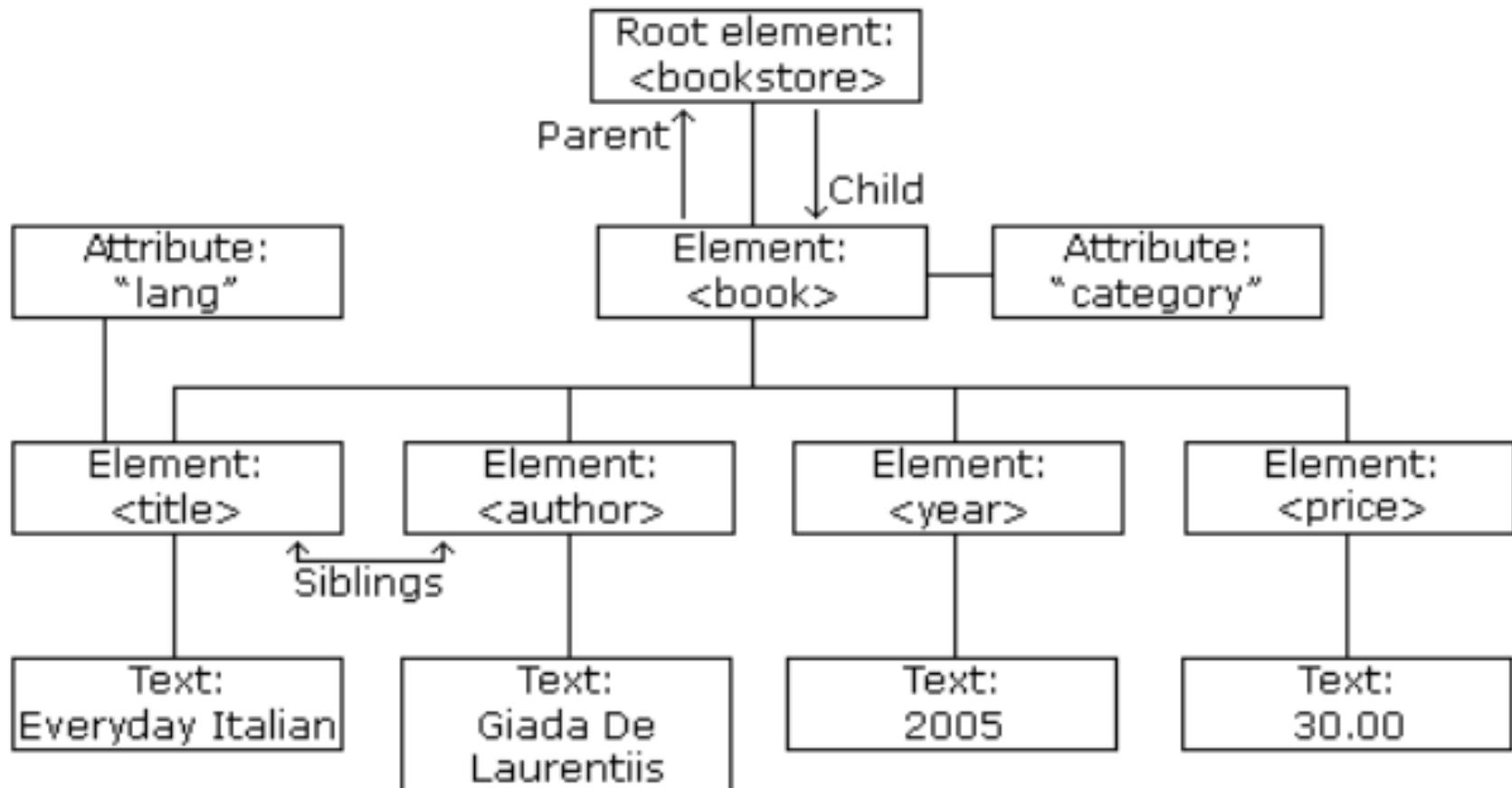
XML is Often a Complement to HTML

- In many HTML applications, XML is used to store or transport data, while HTML is used to format and display the same data.

XML Separates Data from HTML

- When displaying data in HTML, you should not have to edit the HTML file when the data changes.
- With XML, the data can be stored in separate XML files.
- With a few lines of JavaScript code, you can read an XML file and update the data content of any HTML page.

The XML Tree Structure



The XML Tree Structure

- XML documents are formed as element trees.
- An XML tree starts at a root element and branches from the root to child elements.
- All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>....</subchild>
  </child>
</root>
```

- The terms parent, child, and sibling are used to describe the relationships between elements.
- Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

XML Syntax Rules

- XML documents must contain one root element that is the parent of all other elements.
- All XML Elements Must Have a Closing Tag
- XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.
- Opening and closing tags must be written with the same case
- XML Elements Must be Properly Nested

```
<b><i>This text is bold and italic </i></b>
```

- XML Attribute Values Must Always be Quoted as XML elements can have attributes in name/value pairs just like in HTML.
- Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>salary < 1000</message>    ERROR
```

- To avoid this error, replace the "<" character with an entity reference:

<message>salary < 1000</message> CORRECT

There are 5 pre-defined entity references in XML:

- Comments in XML**

The syntax for writing comments in XML is similar to that of HTML:

<!-- This is a comment -->

- White-space is Preserved in XML**

XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

XML: Hello Tove

HTML: Hello Tove

XML Elements

- An XML element is everything from (including) the element's start tag to (including) the element's end tag.

Example: <price>29.99</price>

An element can contain:

- Text
- other elements
- attributes
- or a mix of the above

In the example above:

- <title>, <author>, <year>, and <price> have text content because they contain text (like 29.99).
- <bookstore> and <book> have element contents, because they contain elements.
- <book> has an attribute (category="children").

Example:

```
<bookstore>
```

```
  <book category="children">
```

```
    <title>Harry Potter</title>
```

```
    <author>J. K. Rowling</author>
```

```
    <year>2005</year>
```

```
    <price>29.99</price>
```

```
  </book>
```

```
  <book category="web">
```

```
    <title>Learning XML</title>
```

```
    <author>Erik T. Ray</author>
```

```
    <year>2003</year>
```

```
    <price>39.95</price>
```

```
  </book>
```

```
</bookstore>
```

Empty XML Elements

- An element with no content is said to be empty.
- In XML, you can indicate an empty element like this:

```
<element></element>
```

- You can also use a so called self-closing tag:

```
<element />
```

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces
- Any name can be used, no words are reserved (except xml).

1. Case-sensitive

1. Starts with letter

0. Starts with underscore

1. Contains letters, digits, hyphens, underscores, and periods

1. No spaces

1. Any name can be used

XML Elements are Extensible

- XML elements can be extended to carry more information.
- Suppose we created an application that extracted the `<to>`, `<from>`, and `<body>` elements from the XML document to produce the output.
- If the author of the XML document added some extra information to it:
- The application will not break or crash and application should still be able to find the `<to>`, `<from>`, and `<body>` elements in the XML document and produce the same output.
- This is one of the beauties of XML. It can be extended without breaking applications.

```
<note>
```

```
    <to>Jiya</to>
```

```
    <from>Tanisha</from>
```

```
    <body>Hi! I hope you remember we have a meeting this wednesday</body>
```

```
</note>
```

```
<note>
```

```
    <date>2024-10-05</date>
```

```
    <to>Jiya</to>
```

```
    <from>Tanisha</from>
```

```
    <heading>Meeting reminder</heading>
```

```
    <body>Hi! I hope you remember we have a meeting this wednesday</body>
```

```
</note>
```

XML Attributes

- Attributes are designed to contain data related to a specific element.
- Attribute values must always be quoted. Either single or double quotes can be used.
- Example:

```
<student gender="female">  
  <name>Prerna</name>  
  <marks>95</marks>  
</student>
```

XML Attributes for Metadata

- Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in HTML.
- Metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

Example:

```
<college>  
  <student sid="s1">  
    <name>Prerna</name>  
    <marks>95</marks>  
</student>  
  
<student sid="s2">  
  <name>Prerna</name>  
  <marks>95</marks>  
</student>  
</college>
```

DTD: Document Type Definition

- A DTD defines the structure and the legal elements and attributes of an XML document.
- With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
- An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)> #PCDATA means parseable character data
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Jiya</to>
  <from>Tanisha</from>
  <heading>Important</heading>
  <body>we are meeting this week.</body>
</note>
```

An External DTD Declaration

- If the DTD is declared in an external file, the `<!DOCTYPE>` definition must contain a reference to the DTD file:

```
<?xml version="1.0"?>  
<!DOCTYPE note SYSTEM "note.dtd">  
  
<note>  
  <to>Jiya</to>  
  <from>Tanisha</from>  
  <heading>Important</heading>  
  <body>we are meeting this week.</body>  
</note>
```

This is the file "note.dtd", which contains the DTD:

```
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>  
  
;-
```

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

Elements : Elements are the main building blocks of both XML and HTML documents.

Attributes : Attributes provide extra information about elements.

Entities : Some characters have a **special meaning** in XML, like the less than sign (`<`) that defines the start of an XML tag.

PCDATA : PCDATA means parsed character data.PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

CDATA : CDATA means character data.CDATA is text that will NOT be parsed by a parser

DTD Elements

- Elements with one or more children are declared with the name of the children elements inside parentheses:

Example:

```
<!ELEMENT note (to,from,heading,body)>
```

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.

The full declaration of the "note" element is:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- `<!ELEMENT element-name (child-name)>`

The example declares that the child element "message" must occur once, and only once inside the "note" element.

Example:

```
<!ELEMENT note (message)>
```

- The + sign in the example declares that the child element "message" must occur one or more times inside the "note" element.

```
<!ELEMENT note (message+)>
```

- The * sign in the example declares that the child element "message" can occur zero or more times inside the "note" element.

```
<!ELEMENT note (message*)>
```

- The ? sign declares that the child element "message" can occur zero or one time inside the "note" element.

```
<!ELEMENT note (message?)>
```

- The example declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

```
<!ELEMENT note (to,from,header,(message|body))>
```

- The example declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

DTD Attributes

In a DTD, attributes are declared with an ATTLIST declaration.

An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>
```

- The attribute-type examples : CDATA:The value is character data , ID:The value is a unique id
- The attribute-value can be: #REQUIRED:The attribute is required , #IMPLIED:The attribute is optional , #FIXED value:The attribute value is fixed

A Default Attribute Value

In the example , the "square" element is defined to be an empty element with a "width" attribute of type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present

DTD:

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

Valid XML:

```
<square width="100" />
```

DTD:

```
<!ATTLIST person age CDATA #REQUIRED>
```

Valid XML:

```
<person age="22" />
```

Invalid XML:

```
<person />
```

#IMPLIED

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

DTD:

```
<!ATTLIST person age CDATA #IMPLIED>
```

Valid XML:

```
<person age="22" />
```

Valid XML:

```
<person />
```

#FIXED

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

DTD:

```
<!ATTLIST person company CDATA #FIXED "Google">
```

Valid XML:

```
<person company="Google" />
```

Invalid XML:

```
<person company="Amazon" />
```

Enumerated Attribute Values

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

Syntax

```
<!ATTLIST element-name attribute-name (en1|en2|..) default-value>
```

Example

DTD:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check" />
```

or

```
<payment type="cash" />
```

XML Schema

An XML document with correct syntax is called "Well Formed".

An XML document validated against an XML Schema is both "Well Formed" and "Valid".

The XML Schema language is also referred to as XML Schema Definition (XSD)

Learning XML Schema is important because:

- In the XML world, hundreds of standardized XML formats are in daily use.
- Many of these XML standards are defined by XML Schemas.
- XML Schema is an XML-based (and more powerful) alternative to DTD.
- One of the greatest strength of XML Schemas is the support for data types.

XML Schemas use XML Syntax

- Another great strength about XML Schemas is that they are written in XML.
 - You don't have to learn a new language
 - You can use your XML editor to edit your Schema files
 - You can use your XML parser to parse your Schema files
 - You can manipulate your Schema with the XML DOM
 - You can transform your Schema with XSLT
-
- An XML Schema file called "note.xsd" defines the elements of the XML document.

XSD Schema

- The <schema> element is the root element of every XML Schema.
- The <schema> element may contain some attributes. A schema declaration often looks something like this:

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

- This indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs
- The syntax for defining a simple element is:

```
<xsd:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

- XML Schema has a lot of built-in data types. The most common types are:

xs:string,

xs:decimal

xs:integer

xs:boolean

xs:date,

xs:time

- Example : <xs:element name="age" type="xs:integer"/>

- A default value is automatically assigned to the element when no other value is specified.

Example : <xs:element name="color" type="xs:string" default="green"/>

- A fixed value is also automatically assigned to the element, and you cannot specify another value.

Example: <xs:element name="color" type="xs:string" fixed="green"/>

XSD Attributes

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

- The syntax for defining an attribute is:

```
<xs:attribute name="xxx" type="yyy"/>
```

where xxx is the name of the attribute and yyy specifies the data type of the attribute.

- XML Schema has a lot of built-in data types. The most common types are:

xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time

- An XML element with an attribute: <lastname lang="EN">Ananya</lastname>

And this is the corresponding attribute definition: <xs:attribute name="lang" type="xs:string"/>

- <xs:attribute name="lang" type="xs:string" default="EN"/>

- <xs:attribute name="lang" type="xs:string" fixed="EN"/>

- <xs:attribute name="lang" type="xs:string" use="required"/>

Restrictions on Content

- When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.
- If an XML element is of type "xs:date" and contains a string like "Tanya", the element will not validate.
- With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets.

Complex Element

- A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

- A complex XML element, "product", which is empty:

```
<employee eid="12"/>
```

- A complex XML element, "employee", which contains only other elements:

```
<employee>
  <firstname>Rajvi</firstname>
</employee>
```

- A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

- A complex XML element, "description", which contains both elements and text:

```
<description>
  It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

XML DATA

```
<class>  
  <student>  
    <firstname>Shruti</firstname>  
    <age>20</age>  
  </student>  
</class>
```

XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  <xs:element name="class">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="student">  
          <xs:complexType>  
            <xs:sequence>  
              <xs:element type="xs:string" name="firstname"/>  
              <xs:element type="xs:byte" name="age"/>  
            </xs:sequence>  
          </xs:complexType>  
        </xs:element>  
      </xs:sequence>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

XML DATA

```
<class>
  <student>
    <firstname>Shruti</firstname>
    <age>20</age>
  </student>

  <student>
    <firstname>Varnika</firstname>
    <age>20</age>
  </student>
</class>
```

XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="class">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="student" maxOccurs="unbounded"
minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="firstname"/>
              <xs:element type="xs:integer" name="age"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```