

An Introduction to XML

Part 2

Selim Mimaroglu

This presentation is built on:

XML Bible 1.1 by Elliotte Rusty Harold

XPath: Navigating XML with XPath 1.0 and 2.0 Kick Start by
Steven Holzner

Entities

- Logically speaking, an XML document is composed of a prolog followed by a root element that strictly contains all other elements; but physically the content of an XML document can be spread across multiple files.
- The storage units that contain particular parts of an XML document are *entities*. An entity can be a file, a database record, or any other item that contains data.

There are two types of entities:

- Internal Entities `<!ENTITY name “replacement text”>`
- External Entities

Internal Entity:

You can think of an internal general entity reference as an abbreviation for commonly used text or text that’s hard to type.

For example, instead of typing the same footer at the bottom of every page, you can simply define that text as an internal entity and use it at the bottom of each page.

```
<?xml version="1.0"?>
<!DOCTYPE DOCUMENT [
  <!ENTITY ERH "Elliote Rusty Harold">
  <!ELEMENT DOCUMENT (TITLE, SIGNATURE)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT COPYRIGHT (#PCDATA)>
  <!ELEMENT EMAIL (#PCDATA)>
  <!ELEMENT LAST_MODIFIED (#PCDATA)>
  <!ELEMENT SIGNATURE (COPYRIGHT, EMAIL, LAST_MODIFIED)>
]>
<DOCUMENT>
  <TITLE>&ERH;</TITLE>
  <SIGNATURE>
    <COPYRIGHT>2004 &ERH;</COPYRIGHT>
    <EMAIL>elharo@metalab.unc.edu</EMAIL>
    <LAST_MODIFIED>July 30, 2004</LAST_MODIFIED>
  </SIGNATURE>
</DOCUMENT>
```

You can define any entity you want; there are only five predefined entities in XML

& **&**

< **<**

> **>**

" **“**

' **‘**

External General Entities

As mentioned earlier an XML file may be composed of several entities. These entities may exist in different locations.

```
<!ENTITY name SYSTEM “URI”>
```

signature.xml

```
<COPYRIGHT>2004 Elliotte Rusty Harold</COPYRIGHT>
```

```
<EMAIL>elharo@metalab.unc.edu</EMAIL>
```

```
<LAST_MODIFIED>July 30, 2004</LAST_MODIFIED>
```

```
<HR/>
```

```
<!DOCTYPE DOCUMENT [  
  <!ELEMENT DOCUMENT  
    (TITLE, COPYRIGHT, EMAIL, LAST_MODIFIED,  
    HR?)>  
  <!ELEMENT TITLE (#PCDATA)>  
  <!ELEMENT COPYRIGHT (#PCDATA)>  
  <!ELEMENT EMAIL (#PCDATA)>  
  <!ELEMENT HR EMPTY>  
  <!ELEMENT LAST_MODIFIED (#PCDATA)>  
  <!ENTITY SIGNATURE SYSTEM  
    "http://www.cs.umb.edu/~smimarog/signature.xml"> ]>  
<DOCUMENT>  
  <TITLE>Entity references</TITLE>  
  &SIGNATURE;  
</DOCUMENT>
```

If you open it in IE that's what you see

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE DOCUMENT (View Source for full
doctype...)>
<DOCUMENT>
  <TITLE>Entity references</TITLE>
  <COPYRIGHT>
    2004 Elliotte Rusty Harold
  </COPYRIGHT>
  <EMAIL>elharo@metalab.unc.edu</EMAIL>
  <LAST_MODIFIED>
    July 30, 2004
  </LAST_MODIFIED>
  <HR />
</DOCUMENT>
```

Namespaces

When **mixing and matching** elements from different XML applications, you are likely to find the same name used for two different things.

Namespaces are solution. They allow each element and attribute in a document to be placed in a different namespace mapped to a particular URI.

Namespace Example

```
<P:PERSON xmlns:P="http://ns.cafeconleche.org/people/">  
  <P:TITLE>Mr.</P:TITLE>  
  <P:FIRST>Mikhail</P:FIRST>  
  <P:LAST>Ovitsky</P:LAST>  
  <P:COMPANY>Duplicative Artists  
  Mismanagement</P:COMPANY>  
  <P:ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </P:ADDRESS>  
</P:PERSON>
```

Another Namespace Example

```
<P:PERSON xmlns:P="http://ns.cafeconleche.org/people/"  
          xmlns:CS="http://cs.umb.edu/~smimarog" >  
  <P:TITLE>Mr.</P:TITLE>  
  <P:FIRST>Mikhail</P:FIRST>  
  <P:LAST>Ovitsky</P:LAST>  
  <CS:COMPANY>Duplicative Artists  
  Mismanagement</CS:COMPANY>  
  <P:ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </P:ADDRESS>  
</P:PERSON>
```

Default Namespace Example

```
<PERSON xmlns="http://ns.cafeconleche.org/people/">  
  <TITLE>Mr.</TITLE>  
  <FIRST>Mikhail</FIRST>  
  <LAST>Ovitsky</LAST>  
  <COMPANY>Duplicative Artists  
    Mismanagement</COMPANY>  
  <ADDRESS>  
    135 Agents Row, Hollywood, CA 90123  
  </ADDRESS>  
</PERSON>
```

XML Schema

- W3C XML Schema Language
- One of the most useful characteristics of XML Schema is its ability to specify data types
- *YEAR* attribute contains a number between 1966 and 2005
- *PRICE* element must contain a number that's greater than zero with two decimal digits of precision

What's wrong with DTD

- Document type definitions (DTDs) are an outgrowth of XML's heritage in the Standardized General Markup Language (SGML). SGML was always intended for narrative-style documents: books, reports, technical manuals, brochures, web pages, and the like.
- The limitation most developers notice first is the almost complete lack of data typing, especially for element content.

- The second problem is that DTDs have an unusual non-XML syntax. The same parsers and APIs that read an XML document can't read a DTD. For example, consider this common element declaration:

```
<!ELEMENT TITLE (#PCDATA)>
```

This is not a legal XML element. You can't begin an element name with an exclamation point. TITLE is not an attribute. Neither is (#PCDATA).

- The third problem is that DTDs are only marginally extensible and don't scale very well. It's difficult to combine independent DTDs together in a sensible way.
- Perhaps most annoyingly, DTDs are only marginally compatible with namespaces.
`<!ELEMENT P:TITLE (#PCDATA)>`

You **must** include the **prefix P**.

Schemas are an attempt to solve all these problems by defining a new XML-based syntax for describing the permissible contents of XML documents that includes the following:

- Powerful data typing including range checking
- Namespace-aware validation based on namespace URIs rather than on prefixes
- Extensibility and scalability

However, schemas are not a be-all and end-all solution. In particular, *schemas do not replace DTDs!*

You can use both schemas and DTDs in the same document. DTDs can do several things that schemas cannot do, most importantly declaring entities.

And DTDs still work very well for the classic sort of narrative documents they were originally designed for. Indeed, for these types of documents, a DTD is often considerably easier to write than an equivalent schema.

Hello W3C XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema
```

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

```
  <xsd:element name="GREETING"
```

```
    type="xsd:string"/>
```

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

```
<GREETING>
```

```
  Hello XML!
```

```
</GREETING>
```

<?xml version="1.0"?>

<GREETING

xsi:noNamespaceSchemaLocation="greetin
g.xsd"

xmlns:xsi="http://www.w3.org/2001/XMLS
chema-instance">

Hello XML!

</GREETING>

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

  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="COMPOSER" type="xsd:string"/>
      <xsd:element name="PRODUCER" type="xsd:string"/>
      <xsd:element name="PUBLISHER" type="xsd:string"/>
      <xsd:element name="LENGTH" type="xsd:string"/>
      <xsd:element name="YEAR" type="xsd:string"/>
      <xsd:element name="ARTIST" type="xsd:string"/>
      <xsd:element name="PRICE" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

<?xml version="1.0"?>

<SONG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="song.xsd">

<TITLE>Yes I Am</TITLE>

<COMPOSER>Melissa Etheridge</COMPOSER>

<PRODUCER>Hugh Padgham</PRODUCER>

<PUBLISHER>Island Records</PUBLISHER>

<LENGTH>4:24</LENGTH>

<YEAR>1993</YEAR>

<ARTIST>Melissa Etheridge</ARTIST>

<PRICE>\$1.25</PRICE>

</SONG>

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="COMPOSER" type="xsd:string"
        minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string"
        minOccurs="0" maxOccurs="1"/>
      <xsd:element name="LENGTH" type="xsd:string" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="YEAR" type="xsd:string" minOccurs="1"
        maxOccurs="1"/>
      <xsd:element name="ARTIST" type="xsd:string" minOccurs="1"
        maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string" minOccurs="0"
        maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Grouping

The W3C XML Schema Language provides three grouping constructs that specify whether and how ordering of individual elements is important:

- The `xsd:all` group requires that each element in the group must occur at most once, but that order is not important.
- The `xsd:choice` group specifies that any one element from the group should appear. It can also be used to say that between N and M elements from the group should appear in any order.

- The `xsd:sequence` group requires that each element in the group appear exactly once, in the specified order.

You may fine tune these grouping elements by using ***minOccurs*** and ***maxOccurs*** attributes (*with some restrictions*)

NAME element to have exactly one GIVEN child and one FAMILY child, but that you don't care what order they appear in.

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="NAME">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name="GIVEN"
            type="xsd:string" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="FAMILY"
            type="xsd:string" minOccurs="1" maxOccurs="1"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

The `xsd:choice` element is the schema equivalent of the `|` in DTDs.

```
<xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string"/>
    <xsd:choice>
      <xsd:element name="COMPOSER" type="PersonType"/>
      <xsd:element name="PRODUCER" type="PersonType"/>
    </xsd:choice>
    <xsd:element name="PUBLISHER" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="LENGTH" type="xsd:string"/>
    <xsd:element name="YEAR" type="xsd:string"/>
    <xsd:element name="ARTIST" type="xsd:string"
      maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The `xsd:choice` element itself can have `minOccurs` and `maxOccurs` attributes that establish exactly how many selections may be made from the choice.

```
<xsd:complexType name="SongType">
  <xsd:sequence>
    <xsd:element name="TITLE" type="xsd:string"/>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="COMPOSER" type="PersonType"/>
      <xsd:element name="PRODUCER" type="PersonType"/>
    </xsd:choice>
    <xsd:element name="PUBLISHER" type="xsd:string"
      minOccurs="0"/>
    <xsd:element name="LENGTH" type="xsd:string"/>
    <xsd:element name="YEAR" type="xsd:string"/>
    <xsd:element name="ARTIST" type="xsd:string"
      maxOccurs="unbounded"/>
    <xsd:element name="PRICE" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SONG" type="SongType"/>
  <xsd:complexType name="SongType">
    <xsd:sequence>
      <xsd:element name="TITLE" type="xsd:string"/>
      <xsd:element name="COMPOSER" type="xsd:string"
        maxOccurs="unbounded"/>
      <xsd:element name="PRODUCER" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="PUBLISHER" type="xsd:string"
        minOccurs="0"/>
      <xsd:element name="LENGTH" type="xsd:duration"/>
      <xsd:element name="YEAR" type="xsd:gYear"/>
      <xsd:element name="ARTIST" type="xsd:string"
        maxOccurs="unbounded"/>
      <xsd:element name="PRICE" type="xsd:string"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

XPath

- XPath's primary purpose is to address parts of an XML document
- To do that, XPath uses its own, non-XML, syntax. XPath operates on the abstract, logical structure of an XML document, and this logical structure is known as the data model

- Probably the most important type of XPath expressions is the location path, and for many people, this is what XPath is all about, because you use a location path to select a set of nodes (which may contain just a single node). Using a location path, you can tell XPath exactly what data you want to extract from an XML document.

`/2004/april/stocks/data`

`//stocks`

planets.xml

```
<?xml version="1.0"?>
```

```
<planets>
```

```
  <planet>
```

```
    <name>Mercury</name>
```

```
    <mass units="(Earth = 1)">.0553</mass>
```

```
    <day units="days">58.65</day>
```

```
    <radius units="miles">1516</radius>
```

```
    <density units="(Earth = 1)">.983</density>
```

```
    <distance units="million miles">43.4</distance>
```

```
    <!--At perihelion-->
```

```
  </planet>
```

```
...
```

```
</planets>
```

Here are the data types in XPath 1.0
(XPath 2.0 adds many more data types, as
we'll see in the second half of the book):

- A number— stored as a floating-point number
- A string— a sequence of characters
- A Boolean— a true or false value
- A node-set— an unordered collection of unique nodes

XPath Node Types

There are seven types of nodes in XPath 1.0:

- Root nodes
- Element nodes
- Attribute nodes
- Processing instruction nodes
- Comment nodes
- Text nodes
- Namespace nodes

Element Nodes

We're already familiar with element nodes because they correspond to the elements in an XML document—there is one element node in the XPath node tree for every element in the original XML document. You can see plenty of elements in our sample XML document, *planets.xml*, such as `<planets>`, `<planet>`, and so on.

Attribute Nodes

- In XPath, you can refer to attributes using the attribute axis or its shorthand version, `@`.
- To match all attributes in a document, you can use the XPath expression `//@*`
- In XPath terms, the element is the parent of each of its attribute nodes—however, an attribute node is not considered a child of its parent element. Note that this is different from the W3C XML Document Object Model (DOM), which does not treat the element with an attribute as the parent of the attribute. *(just a definition)*

Processing Instruction Nodes

- There is a processing instruction node for every XML processing instruction. For example:

```
<?xml version="1.0"?>
```

```
<?xml-stylesheet type="text/xsl" href="ch01_02.xsl"?>
```

```
<planets> ...
```

- From an XPath 1.0 point of view, the value of a processing instruction is everything following the processing instruction's target (xml-stylesheet here) up to the final ?. For example, the value of `<?xml-stylesheet type="text/xsl" href="ch01_02.xsl"?>` is `type="text/xsl" href="ch01_02.xsl"`.

Comment Nodes

- As you'd expect, comment nodes in XPath correspond to comments in XML documents, which are delimited with `<!--` and `-->`. As far as XPath is concerned, the value of a comment node is the text between `<!--` and `-->`. In an XPath document tree, there is a comment node for every comment (except for any comment that occurs in a DTD or schema).
- In XPath, you can match comments with the comment node test, which means that the expression `//comment()` matches all comment nodes in a document.

Text Nodes

- XPath also gives you the means of handling text data in elements as text nodes. For example, the value of the text node in the <name> element here is "Mercury":

```
<?xml version="1.0"?>  
<planets>  
  <planet>  
    <name>Mercury</name>
```

- In XPath, you can match text nodes with the text node function, which means that you can match all text nodes throughout a document with the expression //text(),

Sample XML

```
<?xml version="1.0"?>
```

```
<library>
```

```
  <book>
```

```
    <title>
```

```
    I Love XPath
```

```
  </title>
```

```
    <title>
```

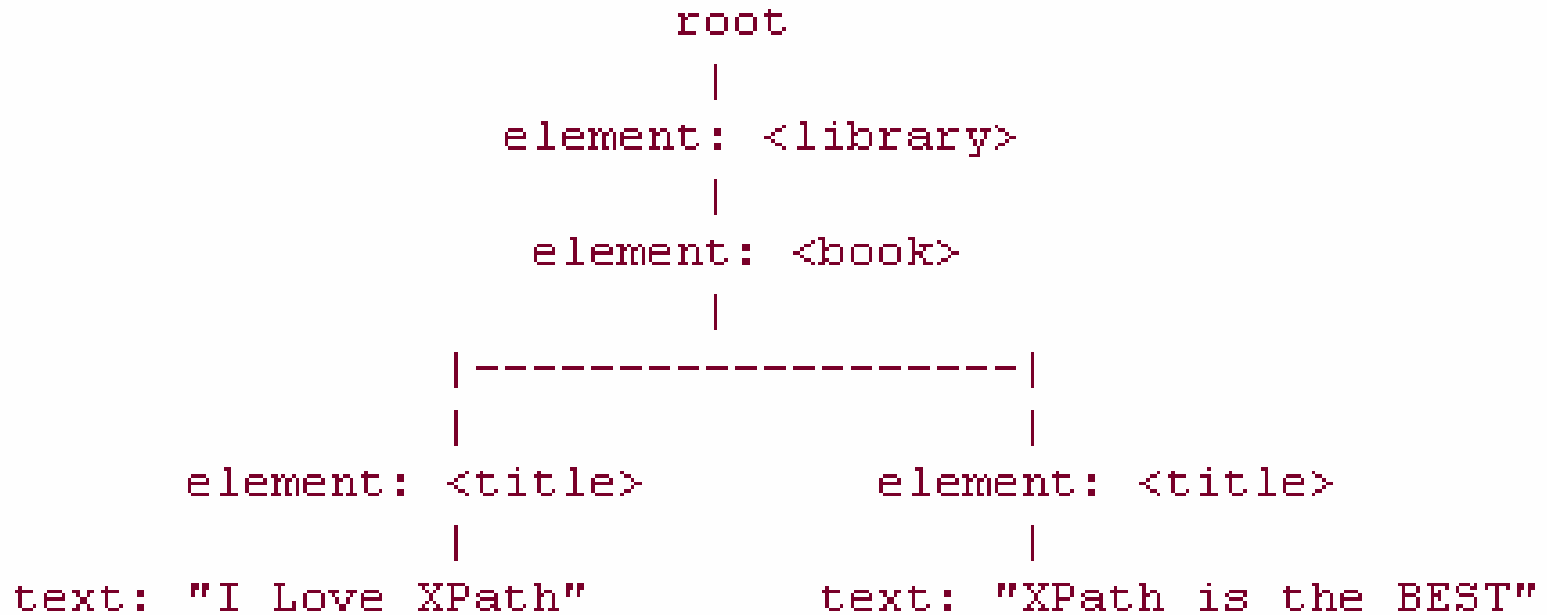
```
    XPath is the BEST
```

```
  </title>
```

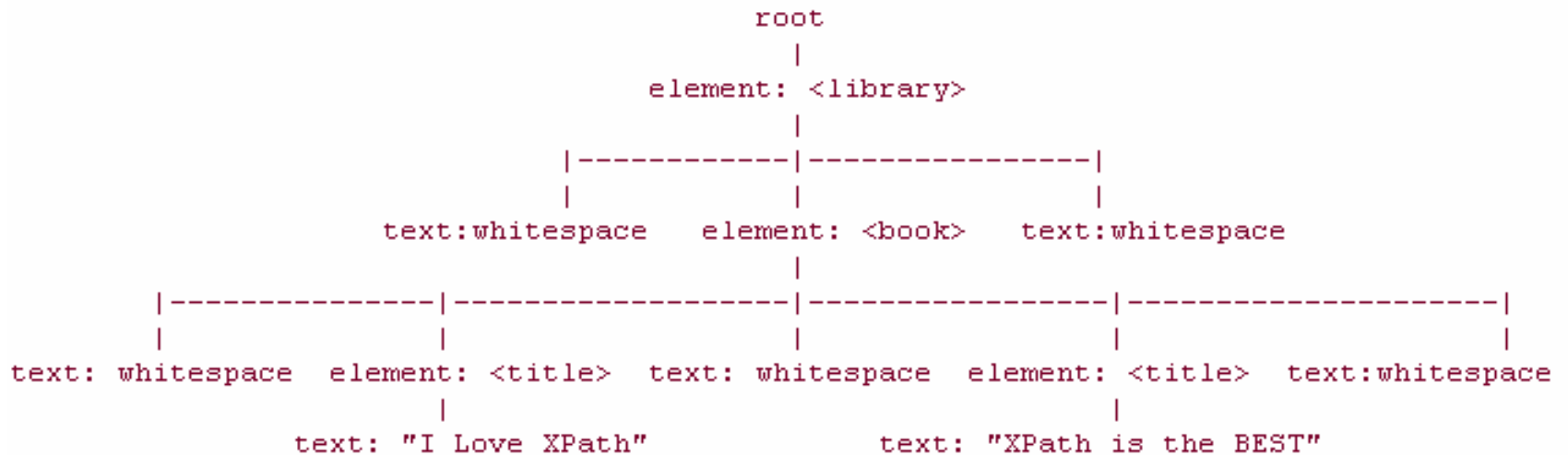
```
  </book>
```

```
</library>
```

XPath Node Tree



Another XPath Node Tree



/planets/planet/name

XPPath Visualiser Ver. 1.4 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address

XPath expression:

0 of 3/3 matches

```
<planets>
  <planet>
    <name>Mercury</name>
    <mass units="(Earth = 1)">.0553</mass>
    <day units="days">58.65</day>
    <radius units="miles">1516</radius>
    <density units="(Earth = 1)">.983</density>
    <distance units="million miles">43.4</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Venus</name>
    <mass units="(Earth = 1)">.815</mass>
    <day units="days">116.75</day>
    <radius units="miles">3716</radius>
    <density units="(Earth = 1)">.943</density>
    <distance units="million miles">66.8</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Earth</name>
    <mass units="(Earth = 1)">1</mass>
    <day units="days">1</day>
    <radius units="miles">2107</radius>
    <density units="(Earth = 1)">1</density>
    <distance units="million miles">128.4</distance>
    <!--At perihelion-->
  </planet>
</planets>
```

My Computer

start

10:13 AM

//name

The screenshot displays the XPath Visualiser application within a Microsoft Internet Explorer browser window. The address bar shows the file path: `D:\research\KDD_seminar\xpathvisualiserseptember\XPathMain.htm`. The main interface includes a file selection area with the file `D:\research\KDD_seminar\planets.xml` and a text input field for the XPath expression, which contains `//name`. Below the input field, it indicates `0 of 3/3 matches`. The main content area displays the XML document structure with the following content:

```
<planets>
  <planet>
    <name>Mercury</name>
    <mass units="(Earth = 1)">.0553</mass>
    <day units="days">58.65</day>
    <radius units="miles">1516</radius>
    <density units="(Earth = 1)">.983</density>
    <distance units="million miles">43.4</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Venus</name>
    <mass units="(Earth = 1)">.815</mass>
    <day units="days">116.75</day>
    <radius units="miles">3716</radius>
    <density units="(Earth = 1)">.943</density>
    <distance units="million miles">66.8</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Earth</name>
    <mass units="(Earth = 1)">1</mass>
    <day units="days">1</day>
    <radius units="miles">2107</radius>
    <density units="(Earth = 1)">1</density>
    <distance units="million miles">128.4</distance>
    <!--At perihelion-->
  </planet>
</planets>
```

The taskbar at the bottom shows the Windows Start button, several open applications, and the system clock displaying 10:14 AM.

/planets/planet[position()=2]/density

The screenshot shows the XPath Visualiser application running in Microsoft Internet Explorer. The address bar shows the file path: D:\research\KDD_seminar\xpathvisualiserseptember\XPathMain.htm. The XPath expression entered is /planets/planet[position()=2]/density. The application has found 1 match. The XML document content is displayed below, showing the planets Mercury, Venus, and Earth. The density of Venus is highlighted in yellow.

```
<planets>
  <planet>
    <name>Mercury</name>
    <mass units="(Earth = 1)">.0553</mass>
    <day units="days">58.65</day>
    <radius units="miles">1516</radius>
    <density units="(Earth = 1)">.983</density>
    <distance units="million miles">43.4</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Venus</name>
    <mass units="(Earth = 1)">.815</mass>
    <day units="days">116.75</day>
    <radius units="miles">3716</radius>
    <density units="(Earth = 1)">.943</density>
    <distance units="million miles">66.8</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Earth</name>
    <mass units="(Earth = 1)">1</mass>
    <day units="days">1</day>
    <radius units="miles">2107</radius>
    <density units="(Earth = 1)">1</density>
    <distance units="million miles">128.4</distance>
    <!--At perihelion-->
  </planet>
</planets>
```

/planets/planet//@*

The screenshot shows the XPath Visualiser application running in Microsoft Internet Explorer. The address bar displays the file path: D:\research\KDD_seminar\xpathvisualiserseptember\xPathMain.htm. The XPath expression entered is `/planets/planet//@*`. The application shows 0 of 15/15 matches. The XML content is displayed as follows:

```
<planets>
  <planet>
    <name>Mercury</name>
    <mass units="Earth = 1">.0553</mass>
    <day units="days">58.65</day>
    <radius units="miles">1516</radius>
    <density units="Earth = 1">.983</density>
    <distance units="million miles">43.4</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Venus</name>
    <mass units="Earth = 1">.815</mass>
    <day units="days">116.75</day>
    <radius units="miles">3716</radius>
    <density units="Earth = 1">.943</density>
    <distance units="million miles">66.8</distance>
    <!--At perihelion-->
  </planet>
  <planet>
    <name>Earth</name>
    <mass units="Earth = 1">1</mass>
    <day units="days">1</day>
    <radius units="miles">2107</radius>
    <density units="Earth = 1">1</density>
    <distance units="million miles">128.4</distance>
    <!--At perihelion-->
  </planet>
</planets>
```