

## XQUERY

XQuery is a supportive technology, related to XPath, that extracts data from .xml files using a command language that is reminiscent of Perl. It is yet another way of extracting selected element nodes from .xml files. Unlike xslt, XQuery is intended to be processed by a computer program. There are free libraries of code, such as Saxon for Java, that take as parameters the name of the .xml file as a source, the name of the file that holds the XQuery statements and the name of the file to be created. *Note* XQuery file names are required to end in .xq (e.g., myfile.xq).

You install the saxon library (from [www.saxonica.com](http://www.saxonica.com) or other places on the net; latest version is saxon9.jar). [The specifics of where to store .jar files and how to set your CLASSPATH are not discussed here.]

Once the file has been installed, start a terminal window. In that window issue this command:  
 java net.sf.saxon.Query -s sourcedoc.xml querydoc.xq > newoutput.xml

[Replace *sourcedoc.xml* with the name of your own .xml file; *querydoc.xq* with the name of your own .xq file and, of course, *newoutput.xml* with your own file name.] Using the examples below:

```
java net.sf.saxon.Query -s trainlog.xml trainlog1.xq > newoutput.xml
```

a new file is created (newoutput.xml). Obviously, if you're writing a Java program, you can use the saxon library to parse .xml for you. In these examples, the output (results) are streamed back to the computer's monitor. In a more useful program, you'd capture the results, shape them into whatever format you want, and save the resulting file or stream the file back to the user's browser.]

In this example, there are 3 files: the .xml (trainlog.xml), a .dtd (etml.dtd) and the trainlog1.xq. **Note! At the end of this file is a sample .xsd file that is the equivalent of the etml.dtd file.** The .dtd is *not required*. The .dtd and .xsd files are provided only so you can see how these pieces go together.

### trainlog.xml

```
<?xml version="1.0"?>
<!DOCTYPE trainlog SYSTEM "etml.dtd">
<trainlog>
<!-- this is a dummy file for discussion. -->
<session date="03/15/02" type="running" heartrate="150">
  <duration units="minutes">45</duration>
  <distance units="miles">5.5</distance>
  <comments>A mid-morning run after surgery.</comments>
</session>
[and so on...]
</trainlog>
```

### etml.dtd

```
<!ELEMENT trainlog (session)+>
<!ELEMENT session (duration, distance, location, comments)>
<!ATTLIST session
  data CDATA #IMPLIED
  type (running | swimming | cycling) "running"
  heartrate CDATA #IMPLIED
>
<!ELEMENT duration (#PCDATA)>
<!ATTLIST duration
  units (seconds | minutes | hours) "minutes"
>
<!ELEMENT distance (#PCDATA)>
<!ATTLIST distance
  units (miles | kilometers | laps) "miles"
```

```
>
<!ELEMENT location (#PCDATA)>
<!ELEMENT comments (#PCDATA)>
```

Here's a sample .xq file:

### **trainlog.xq**

```
xquery version "1.0";

<runsessions>
  { for $s in //trainlog/session[@type="running"]
    order by $s/date
    return <location>{$s/@date} {data($s/location)} ({data($s/distance)} {data($s/distance/
@units)})}</location>
  }
</runsessions>
```

The xquery version "1.0" is *required*. There are several hardcoded element names (trainlog, session, data) and some attribute names (units, date)].

*How it works:*

For example, in XQuery, to extract all the color elements from this sample document:

```
<?xml version="1.0" encoding="utf-8"?>
<vehicles>
  <vehicle year="2009" make="Rolls Royce" model="Silver Ghost">
    <mileage>32</mileage>
    <color>black</color>
    <price>75000</price>
    <options>
      <option>driver</option>
    </options>
  </vehicle>
  <vehicle year="2009" make="Volvo" model="SX70">
    <mileage>16002</mileage>
    <color>silver</color>
    <price>32000</price>
    <options>
      <option>airbags</option>
    </options>
  </vehicle>
</vehicles>
```

The XQuery command is

```
for $c in //color
return $c
```

This returns the following:

```
<?xml version="1.0" encoding="utf-8"?>
<color>black</color>
<color>silver</color>
```

The // operator is used to return elements anywhere below another element. In this case, it means color elements in the document should be returned. This is the same as specifying

```
for $c in vehicles/vehicle/color
return $c
```

The \$c is a placeholder for the results of the query.

The / at the beginning of a query string refers to the root level of the document or a relative folder level separation (just as it does in DOS and Windows). Note: if you used /color in this example, nothing is returned because color is not the root level.

*Qualifying which elements to select:*

**Square Brackets:** square brackets are used for subqueries. Subqueries don't retrieve elements themselves but are used to qualify the elements that are retrieved. For example: //vehicle/color retrieves color elements that are children of vehicle elements. Another example: //vehicle[color] returns *only* vehicles that have a color element as a child.

*Unknown elements or using \* as a wildcard:*

**Wildcards:** To find all particular elements, e.g. options in the above example, that are *grandchildren* of an element, e.g., vehicle, you could use vehicles/vehicle/options/option or a shorthand vehicles/vehicle/\*/option. The benefit here is that if you don't know the elements between vehicle and option, it doesn't matter! The wildcard \* matches any element. Use it, also, at the end of a query to match all the children of a particular element.

You'll note, too, that the name of the variable placeholder doesn't matter (before we used \$c, now \$o):

```
for $o in vehicles/vehicle/*/option
return $o
```

Results:

```
<?xml version="1.0" encoding="utf-8"?>
  <option>driver</option>
  <option>airbags</option>
```

*Specific nodes by filtering - specific data:*

So far, we've looked solely for elements. Now we can search for particular data. The [ ] operator indicate that the expression within the square brackets should be searched but that the element listed to the left of the square brace should be returned. For instance, to look for "any vehicle element that contains a color element with the value of silver...":

```
for $v in //vehicle[color='green']
return $v
```

Results:

```
<?xml version="1.0" encoding="UTF-8"?>
<vehicle year="2009" make="Volvo" model="SX70">
  <mileage>16002</mileage>
  <color>silver</color>
  <price>30000</price>
  <options>
    <option>airbag</option>
  </options>
</vehicle>
```

The full vehicle element is returned because it appears on the left of the search expression enclosed in the square braces.

*Using Boolean operators and, or and not:*

Say you want to find all vehicles that are silver and have a price of less than 20000, use this query:

```
for $v in //vehicle[color='silver' or price<'20000']  
return $v
```

The idea of negation (not) is expressed as "not", so you can select vehicles that are not green or blue and which were produced in 2009. Note that *attributes* are referenced by the @ symbol.

```
for $v in //vehicle[not(color='blue' or color='green') and @year='2009']  
return $v
```

*Selecting Attributes:*

The @ sign refers to attributes. To return all particular attributes of a given element (e.g., all the year attributes of the vehicle element, do this: //vehicle/@year

To retrieve *all* of the vehicle elements that have the year attribute as well, use: //vehicle[@year]

An example complete query looks like this:

```
for $v in //vehicle[@year="2009"]  
return $v
```

---

There is quite a bit more to XQuery and XPath. This is enough for now.

*Appendix*

etml.xsd

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="trainlog">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="session" type="sessionType"
          minOccurs="0" maxOccurs="unbounded" />
      </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="sessionType">
      <xsd:sequence>
        <xsd:element name="duration" type="xsd:duration"/>
        <xsd:element name="distance" type="distanceType"/>
        <xsd:element name="location" type="xsd:string"/>
        <xsd:element name="comments" type="xsd:string"/>
      </xsd:sequence>

      <xsd:attribute name="date" type="xsd:date" use="required"/>
      <xsd:attribute name="type" type="typeType" use="required"/>
      <xsd:attribute name="heartrate" type="xsd:positiveInteger" />
    </xsd:complexType>

    <xsd:complexType name="distanceType">
      <xsd:simpleContent>
        <xsd:extension base="xsd:decimal">
          <xsd:attribute name="units" type="unitsType" use="required"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>

    <xsd:simpleType name="typeType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="running" />
        <xsd:enumeration value="swimming" />
        <xsd:enumeration value="cycling" />
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="unitsType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="miles"/>
        <xsd:enumeration value="kilometers"/>
        <xsd:enumeration value="laps"/>
      </xsd:restriction>
    </xsd:simpleType>

  </xsd:schema>
```