OData Extension for XML Data

Andrew Eisenberg, IBM
Ralf Handl, SAP
Michael Pizzo, Microsoft

July 27, 2012
SQL/XML Background

- Support for XML was added to SQL in 2003 and then extended in 2008.
- SQL/XML allows XML columns to be defined in tables:

```
CREATE TABLE employees
    (empid INTEGER PRIMARY KEY,
     lastname VARCHAR(30) NOT NULL,
     resume XML)
```

- The XMLQUERY function can be used to evaluate an XQuery expression on an XML document and return an XML document as a result:

```
SELECT XMLQUERY ('<jobs> {$r/resume/jobHistory} </jobs>'
    PASSING resume AS "r")
FROM   employees
WHERE  empid='166549'
```

- XMLCAST can be used to cast a value of an XML type into an SQL type:

```
SELECT XMLCAST (XMLQUERY ('($r/resume/jobHistory)[1]/jobTitle'
    PASSING resume AS "r")
    AS VARCHAR(30))
FROM   employees
WHERE  empid='166549'
```
SQL/XML Background

- XML\textbf{E}XISTS can be used to evaluate an XQuery expression, returning true if the result contains any nodes (elements, attributes, etc.).

\begin{verbatim}
SELECT empid
FROM employees
WHERE XML\textbf{E}XISTS ('$r/resume/jobHistory[contains(., "Marketing")]' 
   PASSING resume AS "r")
\end{verbatim}
Requirements

- The following capabilities must be supported in this extension to OData:
  - An OData Stream data type may be annotated to represent an XML data type
  - XML properties may be returned separately from non-XML properties
  - Entities may be filtered based on the content of their XML properties
  - XML values that have been derived from XML properties may be retrieved
  - Scalar values that have been derived from XML properties may be retrieved
  - XMLQUERY, XMLEXISTS, and XMLCAST operations may be applied to XML properties
  - The values of XML properties may be updated
The Employees entity set might now be published as:

```xml
<Schema
  <EntityContainer Name="MyCompany">
    <EntitySet Name="Employees" EntityType="Employee"/>
  </EntityContainer>
  <EntityType Name="Employee">
    <Key>
      <PropertyRef Name="empid"/>
    </Key>
    <Property Name="empid" Type="Edm.Int32" Nullable="false"/>
    <Property Name="lastname" Type="Edm.String" Nullable="false" MaxLength="30" FixedLength="false" Unicode="true"/>
    <Property Name="resume" Type="Edm.Stream" Nullable="true" MaxLength="Max" FixedLength="false">
      <ValueAnnotation Term="OData.ContentType" String="text/xml"/>
    </Property>
  </EntityType>
</Schema>
```
Examples

- To retrieve a single employee, one might submit:

  http://www.example.com/mycompany/Employees(166549)

  This might return:

  ```xml
  <entry xml:base=http://www.example.com/mycompany/Employees ...>
    <id>http://www.example.com/mycompany/Employees(166549)</id>
    <link
type="application/xml" title="resume" href="resume166549.xml"/>
    <link
type="application/xml" title="resume" href="resume166549.xml"/>
    <m:properties>
      <d:empid>166549</d:empid>
      <d:lastname>Heinlein</d:lastname>
    </m:properties>
  </entry>
  ```
Examples

- To retrieve only those employees that have “Marketing” in the job history located in their resume, one might submit:

  http://www.example.com/mycompany/Employees
  ?$filter=resume/XML.XMLX EXISTS
  (query = '$r//jobHistory[contains(.,"Marketing")]', vname= 'r')

- Or, if the variable name can be defaulted to “it”:

  http://www.example.com/mycompany/Employees
  ?$filter=resume/XML.XMLX EXISTS
  (query = '$it//jobHistory [contains(., "Marketing")])

- To return every employee, ordering the result based on the state in which they live, where that state is located in their resume, one might submit:

  http://www.example.com/mycompany/Employees
  ?$orderby=resume/XML.XMLQUERY_STRING
  (query = '$r/resume/address/state', vname = 'r')
Examples

- To return the name and phone number of every employee, where the phone number has been taken from his or her resume, one might submit:

  http://www.example.com/mycompany/Employees
  ?$select=lastname,
  resume/XML.XMLQUERY_STRING (query = '$r/resume/phone', vname = 'r')

Note: This query would require an extension to OData, allowing an expression to appear in a $select query option.
Update

- To replace an old resume with a new resume for a specific employee, one might submit:

```
PUT /resume166549.xml HTTP/1.1
Host: host
Content-Type: application/xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
If-Match: ...Etag...
Content-Length: ####

<resume>
    <ssn>1234</ssn>
    <lastname>Handl</lastname>
    <address>
        <street>ABC St.</street>
        <zipcode>10022</zipcode>
    </address>
    <experience>excellent</experience>
</resume>
```
Design Principles

- The design principles of OData extensions are to:
  - Ensure extensions do not violate the core semantics of OData
  - Avoid defining different representations for common concepts across extensions
  - Ensure independent extensions compose well
  - Ensure clients can ignore extended functionality and still query and consume data correctly
Technical Direction

- The design of this extension to OData should take the following direction:
  - An OData vocabulary for XML shall be defined.
  - An annotation from a common vocabulary defining the content type should be applied to a Stream property that represents XML documents.
  - The XML vocabulary will define functions that can be applied to XML properties.
  - These functions will be based on functions found in SQL/XML:2011, and by extension will be based on XQuery 1.0.
Open questions, issues and work items

- Support may be provided for transforming an XML property by applying an XSLT stylesheet.
- The XML annotation may contain additional properties describing the XML document. These properties might include the location of the schema used to validate the document.
- Support may be provided for updating only a portion of an XML property.
- XML operations could define a default variable name for the document being processed. The document could be assigned as the context item for that query.
- OData could be extended to allow expressions in the $select query option, allowing derived values to be returned along with the properties of an entity.
- OData could be extended with an operator that returns the content of a Stream as either a String or Binary value.
- The OData.ContentType value annotation could be defined to allow multiple content types as its value.
- Support for a function similar to XSLTRANSFORM could be added.
References