OData Extension for XML Data

Andrew Eisenberg, IBM

Jan. 30, 2013

Overview

- Initial design of vocabularies for Content and XML
- Some changes to OData core are needed
- Open questions, issues, and work items

Content Vocabulary

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx
   xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx/4.0"
   Version="4.0">

   <edmx:DataServices DataServiceVersion="4.0">

      <Schema
         Namespace="Org.OData.Content"
         Alias="Content"
         xmlns="http://docs.oasis-open.org/odata/ns/csdl/4.0">

         <ValueTerm
            Name="ContentType"
            Type="Edm.String"
            MaxLength="Max"
            Nullable="false">

            <Documentation>
               <Summary>Identify the content type of a stream.</Summary>
               <LongDescription>
                  Media Types are defined by IANA:
                  http://www.iana.org/assignments/media-types
               </LongDescription>
            </Documentation>
         </ValueTerm>

      </Schema>

   </edmx:DataServices>
</edmx:Edmx>
```

Content Vocabulary

- ContentType could be included in a Common vocabulary

XML Vocabulary

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx ...>

   <edmx:Reference Url="http://vocabs.odata.org/content/v1.0" />

   <edmx:DataServices DataServiceVersion="4.0">
      <Schema
         Namespace="org.odata.vocabs.XML"
         Alias="XML"
         xmlns="http://docs.oasis-open.org/odata/ns/csdl/4.0">

         <Using Namespace="Org.OData.Content" Alias="Content" />

         <TypeDefinition
               Name="XML"
               UnderlyingType="Edm.Stream"
               MaxLength="Max"
               FixedLength="false">
            <Documentation>
               <Summary>The XML data type.</Summary>
               <LongDescription>...</LongDescription>
            </Documentation>

            <ValueAnnotation Term="Content.ContentType" String="application/xml"/>
         </TypeDefinition>
         .
         .
         .
      </Schema>
   </edmx:DataServices>
  </edmx:Edmx>
```

XML Vocabulary

```
<Schema
   Namespace="org.odata.vocabs.XML"
   Alias="XML"
   xmlns="http://docs.oasis-open.org/odata/ns/csdl/4.0">
   .
   .
   .
   <EntityContainer Name="dummy">

      <FunctionImport Name="Query" ... </FunctionImport>
      <FunctionImport Name="Exists" ... </FunctionImport>
      <FunctionImport Name="Query_String" ... </FunctionImport>
      <FunctionImport Name="Query_Boolean" ... </FunctionImport>
      <FunctionImport Name="Query_Date" ... </FunctionImport>
      <FunctionImport Name="Query_DateTimeOffset" ... </FunctionImport>
      <FunctionImport Name="Query_Decimal" ... </FunctionImport>
      <FunctionImport Name="Query_Double" ... </FunctionImport>
      <FunctionImport Name="Query_Duration" ... </FunctionImport>

   </EntityContainer>

</Schema>
```

XML Vocabulary

- <!-- Query returns an XML document, serialized as a string

  $it is bound to the document contained in an XML property
  The result of the query is serialized, with serialization parameters:

  version, set to the value of the version parameter
  omit-xml-declaration, set to the value of the omitXmlDeclaration parameter
  standalone, set to the value of the standalone parameter

-->

<FunctionImport
  Name="Query"
  ReturnType="Edm.String"
  IsBindable="true"
  IsComposable="true">
  <Parameter Name="it" Type="XML.XML" Nullable="false"/>
  <Parameter Name="query" Type="Edm.String" Nullable="false"/>
  <Parameter Name="version" Type="Edm.String" Nullable="true"/>
  <Parameter Name="omitXmlDeclaration" Type="Edm.String" Nullable="true"/>
  <Parameter Name="standalone" Type="Edm.String" Nullable="true"/>
</FunctionImport>

XML Vocabulary

- <!-- As above, with $arg1 bound to arg1 -->

```
<FunctionImport
      Name="Query"
      ReturnType="Edm.String"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="XML.XML" Nullable="false"/>
   <Parameter Name="query" Type="Edm.String" Nullable="false"/>
   <Parameter Name="version" Type="Edm.String" Nullable="true"/>
   <Parameter Name="omitXmlDeclaration" Type="Edm.String" Nullable="true"/>
   <Parameter Name="standalone" Type="Edm.String" Nullable="true"/>
   <Parameter Name="arg1" Type="Edm.String" Nullable="false"/>
</FunctionImport>
```

XML Vocabulary

```
<!-- As above, with $arg1 bound to the first member of args,
     $arg2 bound to the second member of args, etc.
-->

<FunctionImport
      Name="Query"
      ReturnType="Edm.String"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="XML.XML" Nullable="false"/>
   <Parameter Name="query" Type="Edm.String" Nullable="false"/>
   <Parameter Name="version" Type="Edm.String" Nullable="true"/>
   <Parameter Name="omitXmlDeclaration" Type="Edm.String" Nullable="true"/>
   <Parameter Name="standalone" Type="Edm.String" Nullable="true"/>
   <Parameter Name="args" Type="Collection(Edm.String)" Nullable="true"/>
</FunctionImport>
```

XML Vocabulary

```
<!-- As above, but with $it bound to the XML document in string argument "it".
     An error will be returned if argument "it" is not a well-formed XML
     Document.
-->

<FunctionImport
      Name="Query"
      ReturnType="Edm.String"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="Edm.String" Nullable="false"/>
   <Parameter Name="query" Type="Edm.String" Nullable="false"/>
   <Parameter Name="version" Type="Edm.String" Nullable="true"/>
   <Parameter Name="omitXmlDeclaration" Type="Edm.String" Nullable="true"/>
   <Parameter Name="standalone" Type="Edm.String" Nullable="true"/>
</FunctionImport>

<!-- and with arg1 -->

<!-- and wth args →

<!-- and with string it parameter -->
```

XML Vocabulary

```
<!-- Exists

     returns false if the query result is the empty sequence,
     otherwise true
-->

<FunctionImport
      Name="Exists"
      ReturnType="Edm.Boolean"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="XML.XML" Nullable="false"/>
   <Parameter Name="query" Type="Edm.String" Nullable="false"/>
</FunctionImport>


<!-- and with arg1 -->

<!-- and wth args -->

<!-- and with string it parameter -->
```

XML Vocabulary

```
<!-- Query_String

     Returns the result of the query, after fn:string() has been invoked on it.

-->

<FunctionImport Name="Query_String"
      ReturnType="Edm.String"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="XML.XML" Nullable="false"/>
   <Parameter Name="query" Type="Edm.String" Nullable="false"/>
</FunctionImport>


<!-- and with arg1 -->

<!-- and wth args -->

<!-- and with string it parameter →

<!-- and with Query_Date, Query_DateTimeOffset, Query_Decimal, Query_Double,
     Query_Duration -->
```

Employees Entity Set

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<edmx:Edmx
   xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx/4.0"
   Version="4.0">

   <edmx:Reference Url="http://vocabs.odata.org/XML/v1.0" />

   <edmx:DataServices DataServiceVersion="4.0">
      <Schema
         Namespace="Personnel"
         xmlns="http://docs.oasis-open.org/odata/ns/csdl/4.0">

         <Using Namespace="org.odata.vocabs.XML" Alias="XML" />

         <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="XML.XML"/>
         </EntityType>
      </Schema>
   </edmx:DataServices>

</edmx:Edmx>
```

Examples

- To retrieve an employee summary:

```
http://www.example.com/mycompany/Employees('011223')
/XML.dummy.Query (query = '<summary> {$it//name, $it//jobHistory/} </summary>',
                  version = '1.0')
```

- This may return:

```
<?xml version="1.0"?>
<summary>
   <name>Ken</name>
   <jobHistory>
      <job Company="Digtal Equipmenmt Corp." … />
   </jobHistory>
</summary>
```

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
?$filter=resume/XML.dummy.Exists
                  (query = '$it//jobHistory [contains(., "Marketing")]')
```

- Or:

```
http://www.example.com/mycompany/Employees
?$filter=resume/XML.dummy.Exists
                  (query = '$it//jobHistory [contains(., $arg1)]'
                   arg1 = 'Marketing')
```

- Or:

```
http://www.example.com/mycompany/Employees
?$filter=resume/XML.dummy.Exists
                  (query = '$it//jobHistory [contains(., $arg1)]'
                   args = ['Marketing'])
```

Examples

- Or:

```
http://www.example.com/mycompany/Employees
?$filter=resume/XML.dummy.Exists
                  (query = '$it//jobHistory [contains(., $arg1)]'
                   arg1 = @a1)
 &@a1='Marketing'
```

Examples

- To retrieve only those employees that have “Marketing” in the job history located in their resume and zip code 33442, one might submit:

```
http://www.example.com/mycompany/Employees
?$filter=resume/XML.dummy.Exists
                  (query = '$it//jobHistory [contains(., $arg1)]
                            and $it//zip eq $arg2'
                   args = ['Marketing', '33442'])
```

Examples

- To order employees based on the number of phones they have, one might submit:

```
http://www.example.com/mycompany/Employees
?$orderby=resume/XML.dummy.Query_Decimal(query = 'count($it//phone])')
```

Changes to OData Core

- Relax rule for isBindable attribute in FunctionImport:

  “If the value of the IsBindable attribute is set to true, the function import MUST contain at least one edm:Parameter element, and the first such parameter must represent an entity or collection of entities.”

```
<FunctionImport
      Name="Query"
      ReturnType="Edm.String"
      IsBindable="true"
      IsComposable="true">
   <Parameter Name="it" Type="XML.XML" Nullable="false"/>
   ...
</FunctionImport>
```

- Relax requirement that FunctionImport occur in an EntityContainer

  `XML.dummy.Exists(...)` becomes `XML.Exists(...)`

- Relax restriction that disallows parameter alias in the query portion of the URL

```
?$filter=resume/XML.dummy.Exists(query = '...', arg1 = @a1)&@a1='OData'
```

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

- OData Extension for XML Data - A Directional White Paper, Andrew Eisenberg, Ralf Handl, Michael Pizzo, May 18, 2012, http://www.oasis-open.org/committees/download.php/46087/ODataExtensionforXMLDatav1.0.pdf.