OData JSON Extensions

Ralf Handl, SAP
Susan Malaika, IBM
Michael Pizzo, Microsoft
2012-07-27, the First OASIS OData Face-to-Face
JSON Background

• JSON is widely used as a popular interchange format.
• JSON databases have emerged with native support for JSON documents such PostGres, CouchDB, and MongoDB.
• Attractions of JSON databases include:
  – Schema-less processing where developers do not need to consult database administrators when data structures change.
  – Data structure evolution without altering services data model
    • Example: define a resume as a document, rather than shredding the resume into structured entity properties
• JSON databases allow JSON collections to be defined implicitly (at first JSON insert) or explicitly, e.g. as in PostGres to create a table with a column of JSON data type:
• Common use cases for JSON databases include:
  – Logging the exchanged JSON for audit purposes
  – Examining and querying stored JSON
  – Updating stored JSON
  – Altering subsequent user experiences in accordance with what was learnt from earlier user exchanges from the stored JSON
JSON Requirements

- An OData Stream data type may be annotated to represent a JSON data type
- JSON properties may be returned separately from non-JSON properties
- Entities may be filtered based on the content of their JSON properties
- JSON values that have been derived from JSON properties may be retrieved
- Scalar values that have been derived from JSON properties may be retrieved
- Find operations may be applied to JSON properties
- The values of JSON properties may be updated
Annotation Example for JSON

```xml
Namespace="Personnel">
  <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 Name="OData.ContentType" value="application/json"/>
    </Property>
  </EntityType>
</Schema>
```
Observations on the JSON Examples

- We have avoided using curly braces {} in the OData URLs. Instead we use parentheses ()
- We use a slash / instead of the more usual dot . in JavaScript, to navigate along the JSON structure, because OData uses dot for namespaces
- Functions that return results associated with OData EDM data types have been used, e.g., find_string instead of find, to ensure that function return data type is known
Example : JSON Query

• To retrieve only those employees that have “Pizzo” as reference.lastname in their resume, one might submit:

• To retrieve resumes, 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/resume?$orderby=resume/JSON.find_string(‘()’,(state:1)’)
Example: JSON Update

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

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

resume : {
  ssn: 1234,
  lastname: "Handl",
  address: {zipcode: "10022", street: "ABC st"
  experience: excellent,
}
```
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 Directions

• The following are some technical directions for the JSON extension to OData:
  – An OData vocabulary for JSON shall be defined.
  – An annotation from a common vocabulary defining the JSON content type should be applied to a Stream property that represents JSON documents.
  – The JSON vocabulary will define functions that can be applied to JSON properties.
  – These functions will be based on common functions found in native JSON databases.
Open Questions, Issues and Work Items

- The JSON annotation may contain additional properties describing the JSON document, e.g., one or more JSON schemas
- Support may be provided for updating only a portion of a JSON property
- 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
- An alternative approach for the use of JSON in OData is to map JSON to dynamic properties of open data types. The rationale for choosing a document oriented approach is to treat the JSON as a single unit
- This paper describes functions that operate on JSON encoded documents. These functions are applicable to other encodings such as ATOM, and the technical committee could consider a set of common functions across different encodings
Useful Links

      http://www.w3.org/2011/10/integrationworkshop/p/Documentation-0.1-JSONiq-Article-en-US.pdf