OData JSON Format for Common Schema Definition Language (CSDL) Version 4.0

Committee Specification Draft 01 / Public Review Draft 01

03 December 2015

Specification URIs

This version:

http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/csprd01/odata-json-csdl-v4.0-csprd01.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/csprd01/odata-json-csdl-v4.0-csprd01.html
http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/csprd01/odata-json-csdl-v4.0-csprd01.pdf

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/odata-json-csdl-v4.0.docx (Authoritative)
http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/odata-json-csdl-v4.0.html
http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/odata-json-csdl-v4.0.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft
Ralf Handl (ralf.handl@sap.com), SAP AG

Editors:

Ralf Handl (ralf.handl@sap.com), SAP AG
Hubert Heijkers (hubert.heijkers@nl.ibm.com), IBM
Mike Pizzo (mikep@microsoft.com), Microsoft
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP AG

Additional artifacts:

This prose specification is one component of a Work Product that also includes:

- JSON schema: http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/csprd01/schema/

Related work:

This specification is related to:

- *OData JSON Format Version 4.0*. Latest version. http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.
- *OData Version 4.0,* a multi-part Work Product which includes:
  - *OData Version 4.0 Part 1: Protocol*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
  - *OData Version 4.0 Part 2: URL Conventions*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

- *OData Version 4.0 Part 3: Common Schema Definition Language (CSDL)*. Latest version. http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part3-csdl.html
- ABNF components: *OData ABNF Construction Rules Version 4.0* and *OData ABNF Test Cases.* 30 October 2014. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/abnf/
- Vocabulary components: *OData Core Vocabulary, OData Measures Vocabulary* and *OData Capabilities Vocabulary*. 30 October 2014. http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/vocabularies/

Abstract:

The Open Data Protocol (OData) for representing and interacting with structured content is comprised of a set of specifications. This document extends the specification OData Version 4.0 Part 3: Conceptual Schema Definition Language (CSDL) by defining a JSON format for representing OData CSDL documents. This JSON format for CSDL is based on JSON Schema.

Status:

This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/odata/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

Citation format:

When referencing this specification the following citation format should be used:

[OData-JSON-CSDL-v4.0]

OData JSON Format for Common Schema Definition Language (CSDL) Version 4.0. Edited by Ralf Handl, Hubert Heijkers, Mike Pizzo, and Martin Zurmuehl. 03 December 2015. OASIS Committee Specification Draft 01 / Public Review Draft 01. http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/csprd01/odata-json-csdl-v4.0-csprd01.html. Latest version: http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/odata-json-csdl-v4.0.html.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

Table of Examples

1 Introduction

OData services are described in terms of an Entity Data Model (EDM). **[OData-CSDL]** defines an XML representation of the entity data model exposed by an OData service. This document defines an alternative representation using the JavaScript Object Notation (JSON), see **[RFC7159]**

1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**.

OData CSDL and JSON Schema use the term “schema” with different meaning. In addition, the JSON Schema specifications use the term “JSON Schema” for the specifications as well as the media type, and use “a JSON Schema” for a JSON object that conforms to the JSON Schema specifications. To avoid confusion this document uses “JSON Schema” when referring to the JSON Schema specifications, “JSON Schema object” when referring to a JSON object that conforms to the JSON Schema specifications, and “OData schema” when referring to an OData CSDL schema.

1.2 Normative References

[JS-Core] JSON Schema: core definitions and terminology. http://tools.ietf.org/html/draft-zyp-json-schema-04.

[JS-Validation] JSON Schema: interactive and non interactive validation. http://tools.ietf.org/html/draft-fge-json-schema-validation-00.

[OData-CSDL] OData Version 4.0 Part 3: Common Schema Definition Language (CSDL). See link in “Related work” section on cover page.

[OData-JSON] *OData JSON Format Version 4.0.* See link in “Related work” section on cover page.

[OData-Protocol] *OData Version 4.0 Part 1: Protocol*. See link in “Related work” section on cover page.

[OData-URL] *OData Version 4.0 Part 2: URL Conventions*. See link in "Related work" section on cover page.

[OData-VocCore] *OData Core Vocabulary.* See link in "Related work" section on cover page.

[RFC2119] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[RFC7159] Bray, T., Ed., “The JavaScript Object Notation (JSON) Data Interchange Format”, RFC 7159, March 2014. http://tools.ietf.org/html/rfc7159.

[ECMAScript] *ECMAScript Language Specification Edition 5,1*. June 2011. Standard ECMA-262. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[XML-Schema-2] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, D. Peterson, S. Gao, C. M. Sperberg-McQueen, H. S. Thompson, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 5 April 2012, http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/. Latest version available at http://www.w3.org/TR/xmlschema11-2/.

1.3 Non-Normative References

[JS-Site] JSON Schema Site. http://json-schema.org/.

1.4 Typographical Conventions

Keywords defined by this specification use this `monospaced` font.

```
Normative source code uses this paragraph style.
```

Some sections of this specification are illustrated with non-normative examples.

Example 1: text describing an example uses this paragraph style

```
Non-normative examples use this paragraph style.
```

All examples in this document are non-normative and informative only.

All other text is normative unless otherwise labeled.

2 JSON CSDL Format Design

JSON, as described in **[RFC7159]**, defines a text format for serializing structured data. Objects are serialized as an unordered collection of name-value pairs. JSON Schema (see **[JS-Site]**, **[JS-Core]**, and **[JS-Validation]**) is an emerging standard that defines a JSON format for describing JSON formats.

JSON Schema is extensible and allows adding keywords and formats for CSDL concepts that cannot be translated into JSON Schema concepts.

2.1 Design Goals

The goals of guiding design principles are

- JSON CSDL is valid JSON Schema
- JSON CSDL can be used to by standard JSON Schema validators to validate messages from and to the service
- JSON CSDL contains the same information as the XML format for CSDL defined in **[OData-CSDL]**
- JSON CSDL uses JSON Schema concepts that correspond to CSDL concepts
- JSON CSDL uses **[OData-JSON]** concepts where it goes beyond JSON Schema
- `JSON.parse()` of JSON CSDL creates a JavaScript object graph that
  - Appeals to JavaScript programmers by following common naming conventions
  - Satisfies basic access patterns
  - Can easily be augmented with client-side post-processing to satisfy more sophisticated access patterns

2.2 Design Principles

To achieve the design goals the following principles were applied:

- Structure-describing CSDL elements (structured types, type definitions, enumerations) are translated into JSON Schema constructs
- Attributes and child elements of structure-describing CSDL elements that cannot be translated into JSON Schema constructs are added as extension keywords to the target JSON Schema constructs
- All other CSDL elements are translated into JSON with a consistent set of rules
  - Element and attribute names in UpperCamelCase are converted to lowerCamelCase, and uppercase attribute names are converted to lowercase
  - Attributes and elements that can occur at most once within a parent become name/value pairs
  - Elements that can occur more than once within a parent and can be uniquely identified within their parent (schemas, key properties, entity sets, …) became a name/value pair with pluralized name and a "dictionary" object as value containing one name/value pair per element with the identifier as name
  - Elements that can occur more than once within a parent and cannot be uniquely identified within their parent (action overloads, function overloads, …) become a name/value pair with pluralized name and an array as value containing one item per child element

3 Requesting the JSON CSDL Format

The JSON CSDL format can be requested in Metadata Document Requests (see **[OData-Protocol]**) using the `$format` query option in the request URL with the MIME type `application/schema+json`, optionally followed by format parameters.

Alternatively, this format can be requested using the `Accept` header with the MIME type `application/schema+json`, optionally followed by format parameters.

If specified, `$format` overrides any value specified in the `Accept` header.

Possible format parameters are:

- `IEEE754Compatible`

These are defined in **[OData-JSON]**.

4 CSDL Documents

A CSDL document in JSON is represented as a JSON Schema document with additional keywords.

It must contain name/value pairs with names `$schema` and `odata-version`, and it may contain `definitions`, `actions`, `functions`, `terms`, `entityContainer`, `schemas`, and `references`.

The value of `$schema` is a string with the canonical URL of the `edm.json` schema.

The value of `odata-version` is the string `"4.0"`.

Example 2: Structure of CSDL document

```
{
  "$schema":"http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/edm.json#",
  "odata-version":"4.0"
  "definitions": …,
  "actions": …,
  "functions": …,
  "terms": …,
  "entityContainer": …,
  "schemas": …,
  "references": …
}
```

4.1 Types

The `definitions` object contains one name/value pair per entity type, complex type, enumeration type, and type definition, using the namespace-qualified name of the type. It uses the namespace instead of the alias because these definitions can be reused by other CSDL documents, and aliases are document-local, so they are meaningless for referencing documents.

Example 3: Definitions

```
"definitions":{
  "ODataDemo.Product": …,
  "ODataDemo.Category": …,
  "ODataDemo.Supplier": …,
  "ODataDemo.Country": …,
  "ODataDemo.Address": …,
  "org.example.Employee": …,
  "org.example.Manager": …
}
```

4.1.1 Entity Types and Complex Types

Each structured type is represented as a name/value pair of the standard JSON Schema `definitions` object. The name is the namespace-qualified name of the entity type or complex type, the value is a JSON Schema object of type `object`.

The JSON Schema object may contain the standard JSON Schema name/value pairs appropriate for type `object`. It will not contain the `additionalProperties` keyword, allowing additional properties beyond the declared properties. This is necessary for inheritance as well as annotations and dynamic properties, and is in line with the model versioning rules defined in **[OData-Protocol]**.

If the structured type has a base type, the schema contains the keyword `allOf` whose value is an array with a single item: a JSON Reference to the definition of the base type.

In addition it may contain name/value pairs `abstract` and `openType`, and for entity types also `mediaEntity` and `keys`.

The `abstract`, `openType`, and `mediaEntity` name/value pairs have Boolean values. If not present, their value is `false`. They correspond to the `Abstract`, `OpenType`, and `HasStream` attributes defined in **[OData-CSDL].**

The value of `keys` is an array with one item per key property. If the key property has a key alias, the item is an object with one name/value pair, the name is the key alias and the value is the property path `name` and optionally a name/value pair `alias`. For abstract entity types that neither specify a base type nor a key the value of `keys` is an empty array. An array is used to preserve the order of the key properties.

The JSON Schema object may contain annotations.

Example 4: `Product` entity type

```
"ODataDemo.Product":{
  "type":"object",
  "mediaEntity":true,
  "keys":[
    "ID"
  ],
  "properties": …,
  …
}
```

Example 5: `Manager` entity type inheriting from `Employee`

```
"org.example.Manager":{
  "type":"object",
  "allOf":[
    {
      "$ref":"#/definitions/org.example.Employee"
    }
  ],
  …
}
```

Example 6: `Category` entity type with key alias

```
"org.example.Category18": {
  "type": "object",
  "keys": [
    {
      "EntityInfoID": "Info/ID"
    }
  ],
  …
}
```

4.1.2 Properties

Each structural property and navigation property is represented as a name/value pair of the standard JSON Schema `properties` object. The name is the property name; the value is a JSON Schema object describing the allowed values of the property.

The JSON Schema object may contain annotations.

Example 7: structural and navigation properties of `Supplier` entity type

```
"ODataDemo.Supplier":{
  …,
  "properties":{
    "ID":…,
    "Name":…,
    "Address":…,
    "Concurrency":…,
```

```
    "Products":…
  },
  …
}
```

4.1.2.1 Primitive Properties

Primitive properties of type `Edm.PrimitiveType` and any of the `Edm.Geo*` types are represented as JSON References to definitions in the `edm.json` schema.

Primitive properties of type `Edm.Stream` are represented as JSON References to an unfulfillable definition in the `edm.json` schema as they are never represented in JSON payloads.

All other primitive properties are represented with the following JSON Schema types, formats, and validation keywords:

OData Primitive Type	JSON Schema			Comment
	Type	Format	Keywords	
`Edm.Binary`	`string`	`base64url`	`maxlength` `byteLength`	OData-specific format `maxLength` is maximum length of string representation, i.e. 4*ceil(`MaxLength`/3) `byteLength` is the maximum length of the binary value in octets
`Edm.Boolean`	`boolean`			
`Edm.Byte`	`integer`	`uint8`		OData-specific format
`Edm.Date`	`string`	`date`		Swagger format
`Edm.DateTimeOffset`	`string`	`date-time`	`precision`	OData-specific keyword

`Edm.Decimal`	`number`,
	`string`
		`decimal`	`minimum`
			`maximum`
			`multipleOf`
			`precision`
			`scale`
				OData-specific format
				`string` is needed for IEEE754Compatible mode
				OData-specific keywords `precision` and `scale`

`Edm.Double`	`number` `[,string]`	`double`		Swagger format with extended meaning `string` is needed for `-INF`, `INF`, and `NaN`
`Edm.Duration`	`string`	`duration`		OData-specific format
`Edm.Guid`	`string`	`uuid`		OData-specific format
`Edm.Int16`	`integer`	`int16`		OData-specific format
`Edm.Int32`	`integer`	`int32`		Swagger format
`Edm.Int64`	`integer`, `string`	`int64`		Swagger format `string` is needed for IEEE754Compatible mode

OData Primitive Type	JSON Schema			Comment
	Type	Format	Keywords	
Edm.SByte	integer	int8		OData-specific format
Edm.Single	number [,string]	single		OData-specific format string is needed for -INF, INF, and NaN
Edm.String	string		maxlength	Sequence of UTF-8 characters
Edm.TimeOfDay	string	time	precision	OData-specific format OData-specific keyword

Properties of type Edm.Decimal and Edm.Int64 are represented as JSON strings if the format option IEEE754Compatible=true is specified, so they have to be declared with both number and string.

Properties of type Edm.Decimal use OData-specific keywords precision and scale to represent the corresponding type facets. In addition a numeric scale value is represented with the JSON Schema keyword multipleOf and a value of 10^{-scale}. The precision is represented with the maximum and minimum keywords and a value of $\pm(10^{precision-scale} - 10^{-scale})$ if the scale facet has a numeric value, and $\pm(10^{precision} - 1)$ if the scale is variable).

Properties of type Edm.Double and Edm.Single have special values for -INF, INF, and NaN that are represented as JSON strings, so they also have to be declared with both number and string. Services that do not support the special values -INF, INF, and NaN can omit the string keyword.

The default value of a property is represented with the JSON Schema keyword default.

Example 8: non-nullable Boolean property with default value

```
"BooleanValue":{
  "type":"boolean",
  "default":false
}
```

Example 9: non-nullable binary property with both maxLength and byteLength

```
"BinaryValue":{
  "type":"string",
  "format":"base64url",
  "maxLength":44,
  "byteLength":31,
  "default":"T0RhdGE"
}
```

Example 10: non-nullable integer property

```
"IntegerValue":{
  "type":"integer",
  "format":"int32",
  "default":-128
}
```

Example 11: non-nullable floating-point properties: string representation for -INF, INF, and NaN,

```
"DoubleValue":{
  "type":[
    "number",
    "string"
  ],
```



```
  "format":"double",
  "default":3.1415926535897931
},
"SingleValue":{
  "type":[
    "number",
    "string"
  ],
  "format":"single"
}
```

Example 12: non-nullable decimal property with unspecified precision: no minimum and maximum

```
"DecimalValue":{
  "type":[
    "number",
    "string"
  ],
  "format":"decimal",
  "scale":"variable",
  "default":34.95
}
```

Example 13: non-nullable decimal property with specified precision, minimum and maximum

```
"FixedDecimalValue":{
  "type":[
    "number",
    "string"
  ],
  "format":"decimal",
  "precision":12,
  "scale":2,
  "multipleOf":0.01,
  "minimum":-999999999.99,
  "maximum":999999999.99
}
```

Example 14: non-nullable string property with maximum length of 40 characters

```
"StringValue":{
  "type":"string",
  "maxLength":40
  "default":"Say \"Hello\",\nthen go"
}
```

Example 15: non-nullable date property

```
"DateValue":{
  "type":"string",
  "format":"date",
  "default":"2012-12-03"
}
```

Example 16: non-nullable timestamp property with 7 fractional digits precision

```
"DateTimeOffsetValue":{
  "type":"string",
  "format":"date-time",
  "precision":7,
  "default":"2012-12-03T07:16:23:00.0000000Z"
}
```

Example 17: non-nullable timestamp property with 12 fractional digits precision


```
"DurationValue":{
  "type":"string",
  "format":"duration",
  "precision":12,
  "default":"P12DT23H59M59.999999999999S"
}
```

Example 18: non-nullable time property with 3 fractional digits precision

```
"TimeOfDayValue":{
  "type":"string",
  "format":"time",
  "precision":3,
  "default":"07:59:59.999"
}
```

Example 19: non-nullable guid property with default value

```
"GuidValue":{
  "type":"string",
  "format":"uuid",
  "default":"1234567-89ab-cdef-0123-456789abcdef"
}
```

Example 20: non-nullable 8-byte integer property, allowing for string representation in IEEE754Compatible mode

```
"Int64Value":{
  "type":[
    "integer",
    "string"
  ],
  "format":"int64",
  "default":0
}
```

Example 21: non-nullable enumeration property

```
"ColorEnumValue":{
  "anyOf":[
    {
      "$ref":"#/definitions/Model1.Color"
    }
  ],
  "default":"yellow"
},
```

Example 22: non-nullable geography-point property

```
"GeographyPoint":{
  "anyOf":[
    {
      "$ref":"http://docs.oasis-open.org/odata/odata-json-
csdl/v4.0/edm.json#/definitions/Edm.GeographyPoint"
    }
  ],
  "default":{
    "type":"Point",
    "coordinates":[
      142.1,
      64.1
    ]
  }
}
```

Example 23: non-nullable stream property: not part of payload in version 4.0


```
"StreamValue":{
  "$ref":"http://docs.oasis-open.org/odata/odata-json-
csdl/v4.0/edm.json#/definitions/Edm.Stream"
}
```

Example 24: non-nullable property typed with a type definition

```
"TypeDefValue":{
  "anyOf":[
    {
      "$ref":"#/definitions/Model1.IntegerDecimal"
    }
  ],
  "default":42
}
```

Example 25: non-nullable primitive property with abstract type, e.g. in term definition

```
"PrimitiveValue":{
  "$ref":"http://docs.oasis-open.org/odata/odata-json-
csdl/v4.0/edm.json#/definitions/Edm.PrimitiveType"
}
```

4.1.2.2 Complex Properties

Complex properties are represented as JSON References to the definition of the complex type, either as local references for types directly defined in the CSDL document, or as external references for types defined in referenced CSDL documents.

Example 26: structural properties of `Supplier` entity type: a string property, a nullable string property, a complex property, and an integer property

```
"properties":{
  "ID":{
    "type":"string"
  },
  "Name":{
    "type":[
      "string",
      "null"
    ]
  },
  "Address":{
    "$ref":"#/definitions/ODataDemo.Address"
  },
  "Concurrency":{
    "type":"integer",
    "format":"int32"
  },
  …
}
```

4.1.2.3 Navigation Properties

Navigation properties are represented similar to complex properties so that a standard JSON Schema validator can validate the expanded representation of the navigation property.

Navigation properties contain a `relationship` name/value pair whose value is an object that may contain name/value pairs `partner`, `onDelete`, and `referentialConstraints`.

The value of `partner` is the name of the partner navigation property. The value of `onDelete` is an object with a single name/value pair `action` whose value is one of the values `Cascade`, `None`, `SetDefault`, or `SetNull` defined in **[OData-CSDL]**, section 7.3.1.

The value of `referentialConstraints` is an object with one name/value pair per dependent property, using the dependend property name as name and an object as value. This object contains the name/value pair `referencedProperty` whose value is the name of the principal property.

In addition this object may contain annotations.

Example 27: multi-valued navigation property `Products` with partner and on-delete constraint

```
"Products":{
  "type":"array",
  "items":{
    "$ref":"#/definitions/ODataDemo.Product"
  },
  "relationship":{
    "partner":"Category",
    "onDelete":{
      "action":"Cascade"
    }
  }
}
```

Example 28: required single-valued navigation property `Category`

```
"Category":{
  "anyOf":[
    {
      "$ref":"#/definitions/ODataDemo.Category"
    }
  ],
  "relationship":{}
}
```

Example 29: nullable single-valued navigation property `Country` with referential constraint

```
"Country":{
  "anyOf":[
    {
      "$ref":"#/definitions/ODataDemo.Country"
    },
    {
      "type":"null"
    }
  ],
  "relationship":{
    "referentialConstraints":{
      "CountryName":{
        "referencedProperty":"Name"
      }
    }
  }
}
```

4.1.2.4 Collection-Valued Properties

Collection-valued structural and navigation properties are represented as JSON Schema objects of type `array`. The value of the `items` keyword is a JSON Schema object specifying the type of the items.

Example 30: collection-valued nullable string property `Tags`

```
"Tags":{
  "type":"array",
  "items":{
    "type":[
      "string",
```

```
      "null"
    ]
  }
}
```

Example 31: collection-valued navigation property `Products` of `Supplier` entity type

```
"Products":{
  "type":"array",
  "items":{
    "$ref":"#/definitions/ODataDemo.Product"
  },
  "relationship":{
    "partner":"Supplier"
  }
}
```

4.1.2.5 Nullable Properties

Nullable properties of primitive types except `Edm.Stream` and `Edm.Geo*` are represented as an array-valued JSON Schema `type` that consists of the corresponding JSON Schema primitive type(s) and the JSON Schema `null` type.

Other nullable structural and navigation properties are represented as a JSON Schema object using the `anyOf` keyword followed by a two-element array with a JSON Schema object for the non-null values and a JSON Schema object for the JSON Schema `null` type.

Example 32: nullable property `Price` of type `Edm.Decimal` with precision 15 and scale 3

```
"Price":{
  "type":["number","string","null"],
  "precision":15,
  "scale":3,
  "multipleOf":1e-3,
  "minimum":-999999999999.999,
  "maximum":999999999999.999
}
```

Example 33: nullable property `Created` of type `Edm.DateTimeOffset` with precision 6

```
"Created":{
  "type":["string","null"],
  "format":"date-time",
  "precision":6
}
```

Example 34: nullable collection-valued property `Dates`

```
"Dates":{
  "type":"array",
  "items":{
    "type":["string","null"],
    "format":"date"
  }
},
```

Example 35: nullable navigation property `Supplier`

```
"Supplier":{
  "anyOf":[{
      "$ref":"#/definitions/ODataDemo.Supplier"
    }, {
      "type":"null"
    }
```



```
  ],
  "relationship":{
    "partner":"Products"
  }
}
```

4.1.3 Enumeration Types

Each enumeration type is represented as a name/value pair of the standard JSON Schema `definitions` object. The name is the namespace-qualified name of the type definition; the value is a JSON Schema object describing the allowed values.

If the enumeration type does not allow multiple members to be selected simultaneously, the JSON Schema object uses the `enum` keyword to list all defined values. The value of the `enum` keyword is an array that contains a string with the member name for each enumeration member.

If the enumeration type allows multiple members to be selected simultaneously, the JSON Schema object uses the `anyOf` keyword with an array value containing two JSON Schema objects: one JSON Schema object using the `enum` keyword listing all explicitly defined member names, and one JSON Schema object of type string using the `pattern` keyword with a regular expression for a comma-separated list of member names or nonnegative integer values.

The numeric value of each enumeration member is represented as an annotation on the members with the term `odata.value`.

The outer JSON Schema object may contain annotations. Annotations on enumeration members are represented similar to instance annotations on properties as name/value pairs whose name is the member name, followed by an at (`@`) sign, followed by the namespace-qualified term name, and optionally followed by a hash (#) sign and the qualifier. The annotation value is represented according to the rules defined in this specification.

Example 36: enumeration type with exclusive members and annotations on members and on the type

```
"org.example.ShippingMethod":{
  "enum":[
    "FirstClass",
    "TwoDay",
    "Overnight"
  ],
  "FirstClass@Core.Description":"Shipped with highest priority",
  "TwoDay@Core.Description":"Shipped within two days",
  "Overnight@Core.Description":"Shipped overnight"
  "description":"Method of shipping"
}
```

Example 37: enumeration type with flag values

```
"org.example.Pattern":{
  "anyOf":[
    {
      "enum":[
        "Plain",
        "Red",
        "Blue",
        "Yellow",
        "Solid",
        "Striped",
        "SolidRed",
        "SolidBlue",
        "SolidYellow",
        "RedBlueStriped",
        "RedYellowStriped",
        "BlueYellowStriped"
      ],
```

```
    },
    {
      "type":"string",

"pattern":"^(Plain|Red|Blue|Yellow|Solid|Striped|SolidRed|SolidBlue|SolidYello
w|RedBlueStriped|RedYellowStriped|BlueYellowStriped|[1-9][0-
9]*)(,(Plain|Red|Blue|Yellow|Solid|Striped|SolidRed|SolidBlue|SolidYellow|RedB
lueStriped|RedYellowStriped|BlueYellowStriped|[1-9][0-9]*))*$"
    }
  ],
  "Plain@odata.value":0,
  "Red@odata.value":1,
  "Blue@odata.value":2,
  "Yellow@odata.value":4,
  "Solid@odata.value":8,
  "Striped@odata.value":16,
  "SolidRed@odata.value":9,
  "SolidBlue@odata.value":10,
  "SolidYellow@odata.value":12,
  "RedBlueStriped@odata.value":19,
  "RedYellowStriped@odata.value":21,
  "BlueYellowStriped@odata.value":22
}
```

4.1.4 Type Definitions

Each type definition is represented as a name/value pair of the standard JSON Schema `definitions` object. The name is the namespace-qualified name of the type definition; the value is a JSON Schema object describing the allowed values of the type definition using the same rules as primitive properties.

The JSON Schema object may contain annotations.

Example 38: type definitions based on `Edm.String`, `Edm.Decimal` and `Edm.DateTimeOffset`

```
"Model1.Text50":{
  "type":"string",
  "maxLength":50
},
"Model1.VariableDecimal":{
  "type":"number",
  "description":"A type definition"
},
"Model1.ExactTimestamp":{
  "type":"string",
  "format":"date-time",
  "precision":12
}
```

4.2 Actions and Functions

The `actions` and `functions` objects contain one name/value pair for each action/function name defined in the CSDL document. The name is the namespace-qualified action/function name, the value is an array with one action/function description object for each overload for this name. An action/function description object has name/value pairs `entitySetPath`, `isBound`, `parameters`, and `returnType`. Objects representing functions in addition may have an `isComposable` name/value pair with a Boolean value.

The value of `entitySetPath` is a string.

The values of `isBound` and `isComposable` are Boolean.

The value of `parameters` is an array with one object per parameter. It has a name/value `name` for the parameter name and a name/value pair `parameterType` whose value is a schema describing the allowed parameter values. It has the same structure as the schema for a property.

The value of `returnType` is a schema describing the allowed return values. It has the same structure as the schema for a property.

All objects may contain annotations.

Example 39: action `Rejection` with two overloads and function `Foo` with one overload an no parameters

```
"actions":{
  "Model1.Rejection":[
    {
      "isBound":true,
      "parameters":[
        {
          "name":"foo",
          "parameterType":{
            "$ref":"#/definitions/Model.One.Waldo"
          }
        }
      ]
    },
    {
      "isBound":true,
      "parameters":[
        {
          "name":"bar",
          "parameterType":{
            "$ref":"#/definitions/Model.One.Waldo"
          }
        },
        {
          "name":"Reason",
          "parameterType":{
            "type":"string"
          }
        }
      ]
    }
        ]
},
"functions":{
  "Model1.Foo":[
    {
      "parameters":[ ],
      "returnType":{
        "type":"string",
        "maxLength":42
      }
    }
  ]
}
```

4.3 Entity Container

The `entityContainer` object may contain name/value pairs `entitySets`, `singletons`, `actionImports`, and `functionImports`. The values of these pairs are objects with one name/value pair per container child of that type. The name of each pair is the child's unqualified name, the value is an object.

An object describing an entity set must have an `entityType` name/value pair whose value is a JSON Reference to the entity type that is the base type of all entites in this set. It may have a

`navigationPropertyBindings` name/value pair. Its value is an object with one name/value pair per navigation property that has a binding. The name is the path to the navigation property; the value is an object with a name/value pair target whose value is the name of the target entity set.

An object describing a singleton must have a `type` name/value pair whose value is a JSON Reference to the entity type of the singleton. It may have a `navigationPropertyBindings` name/value pair with the same structure as in objects describing an entity set.

An object describing an action import must have an `action` name/value pair whose value is a JSON Reference to the action triggered by this action import. It may have an `entitySet` name/value pair whose value is the name of the entity set containing the entity or entities returned by the action.

An object describing a function import must have a `function` name/value pair whose value is a JSON Reference to the function triggered by this function import. It may have an `entitySet` name/value pair whose value is the name of the entity set containing the entity or entities returned by the function. If the function has no parameters, it also may have an `includeInServiceDocument` name/value pair with a Boolean value.

All objects may contain annotations.

Example 40: entity container

```
"entityContainer":{
  "name":"DemoService",
  "entitySets":{
    "Products":{
      "entityType":{
        "$ref":"#/definitions/ODataDemo.Product"
      },
      "navigationPropertyBindings":{
        "Category":{
          "target":"Categories"
        }
      }
    },
    "Categories":{
      "entityType":{
        "$ref":"#/definitions/ODataDemo.Category"
      },
      "navigationPropertyBindings":{
        "Products":{
          "target":"Products"
        }
      }
    },
    "Suppliers":{
      "entityType":{
        "$ref":"#/definitions/ODataDemo.Supplier"
      },
      "navigationPropertyBindings":{
        "Products":{
          "target":"Products"
        },
        "Address/Country":{
          "target":"Countries"
        }
      },
      "@Core.OptimisticConcurrency":[
        {
          "@odata.type":"#PropertyPath",
          "value":"Concurrency"
        }
      ]
    },
    "Countries":{
```



```
      "entityType":{
        "$ref":"#/definitions/ODataDemo.Country"
      }
    }
  },
  "singletons":{
    "Contoso":{
      "type":{
        "$ref":"#/definitions/Self.Supplier"
      },
      "navigationPropertyBindings":{
        "Products":{
          "target":"Products"
        }
      }
    }
  },
  "functionImports":{
    "ProductsByRating":{
      "entitySet":"Products",
      "function":{
        "$ref":"#/schemas/ODataDemo/functions/ProductsByRating"
      }
    }
  }
}
```

4.4 Terms

The `terms` object contains one name/value pair per term defined within the CSDL document. The name of each pair is the term's namespace-qualified name, the value is a JSON Schema object describing the type of the term. It has the same structure as the schema for a property, and in addition may have a name/value pair `baseTerm` whose value is a JSON Reference to the base term, and a name/value pair `appliesTo` whose value is either a string or an array of strings specifying the model element(s) the term can be applied to.

All term definition objects may contain annotations.

Example 41: term definition

```
"terms":{
  "Core.IsURL": {
    "anyOf": [
      {
        "$ref": "#/definitions/Org.OData.Core.V1.Tag"
      },
      {
        "type": "null"
      }
    ],
    "default": true,
    "appliesTo": [
      "Property",
      "Term"
    ],
    "description": "Properties and terms annotated with this term MUST contain
a valid URL",
    "@Core.RequiresType": "Edm.String"
  },
  "Core.OptimisticConcurrency": {
    "type": "array",
    "items": {
```

```
      "$ref": "http://docs.oasis-open.org/odata/odata-json-
csdl/v4.0/edm.json#/definitions/Edm.PropertyPath"
    },
    "appliesTo": "EntitySet",
    "description": "Data modification requires the use of Etags. A non-empty
collection contains the set of properties that are used to compute the ETag"
  },
  "Y.Developer":{
    "baseTerm":{
      "$ref":"#/terms/X.Person"
    },
    "anyOf":[
      {
        "$ref":"#/definitions/Y.DeveloperType"
      }
    ]
  }
}
```

4.5 Schemas

The `schemas` object contains one name/value pair per OData schema defined or included in the CSDL document, and one name-value pair per defined alias. The name is either the namespace of the OData schema or the alias assigned to an OData schema. The value of an alias or an included OData schema is a JSON Reference. The value of an OData schema defined in the document is an object that may contain the name/value pair `annotations`.

It also may contain annotations.

Example 42: schemas

```
"schemas":{
  "SomeAlias":{
    "$ref":"#/schemas/Some.Model"
  },
  "Some.Model":{
    "annotations": …,
    "@Annotation.With.Some.Term": …
  }
}
```

4.5.1 Included Schemas and Aliases

OData schemas that are included via a reference to a separate CSDL document as well as aliases for OData schemas are represented as JSON References. Aliases for OData schemas defined in the same document are local references whose URL value consists of `#/schemas/` followed by the namespace of the schema.

Example 43: Alias for schema defined in the same document

```
"SomeAlias":{
  "$ref":"#/schemas/Some.Model"
},
```

Included OData schemas as well as aliases for included OData schemas are represented as JSON References with an absolute or relative URL that locates the document defining the included OData schema.

Example 44: Included schema and alias for the included schema

```
"Org.OData.Core.V1":{
  "$ref":"http://vocabularies.odata.org/Org.OData.Core.V1.json#/schemas/Org.OD
ata.Core.V1"
```

```
},
"Core":{
  "$ref":"http://vocabularies.odata.org/Org.OData.Core.V1.json#/schemas/Org.OD
ata.Core.V1"
},
```

4.5.2 Annotations with External Targeting

Annotations can appear inline within a model element, or externally as a group that targets a model element. Annotations with external targeting are represented as an `annotations` name/value pair whose value is an array of JSON objects. Each of these objects contains a `target` name/value pair whose value is a string with a path expression identifying the annotated model element. In addition, each object contains at least one annotation in the same format that is used for inline annotations.

Example 45: Annotations with external targeting

```
"annotations":[
  {
    "target":"Some.EntityType/SomeProperty",
    "@X.Y":…,
    …
  },
  {
    "target":"Another.EntityType",
    "@X.Y":…,
    …
  },
  …
]
```

4.5.3 Inline Annotations

Annotations are represented similar to instance annotations in **[OData-JSON]**, chapter 18.

Annotations for JSON objects are name/value pairs placed within the object, the name is an at (`@`) sign followed by the namespace-qualified name of the term, optionally followed by a hash (`#`) sign and the qualifier of the annotation.

Annotations for JSON arrays or primitives are name/value pairs placed next to the name/value pair whose value is the annotated array or primitive value. The name is the name of the annotated name/value pair followed by an at (`@`) sign, followed by the namespace-qualified name of the term, optionally followed by a hash (`#`) sign and the qualifier of the annotation.

The value of the annotation is either a plain JSON value or a JSON object.

Example 46: annotation within an object, annotation of a non-object value, and annotation of an annotation

```
"@Some.Term" : …,
"Hugo@Some.Term" : …,
"@Some.Term#SomeQualifier@Some.Term": …
```

Annotations always specify an explicit value, even if the term definition specifies a default value. This is consistent with the representation of instance annotations in JSON payloads and an intentional difference to the XML representation of annotations.

4.5.3.1 Constant Expressions

Constant expressions `edm:Bool` and `edm:String` are represented as plain JSON values as defined in **[OData-JSON]**, section 7.1.

Example 47: a string-valued annotation, a Boolean-valued annotation, a numeric float-valued annotation, an infinity-valued annotation, and an integer annotation

```
"@A.Binary":"T0RhdGE",
"@A.Boolean" : true,
"@A.Date":"2013-10-09",
"@A.DateTimeOffset":"2000-01-01T16:00:00.000Z",
"@A.Decimal":12.34,
"@A.Duration":"P7D",
"@An.EnumMember":"Red,Striped",
"@A.Float":1.23e4,
"@A.Float#inf": "INF",
"@A.Float#minusInf":"-INF",
"@A.Float#nan":"NaN",
"@A.Guid":"86a96539-871b-45cf-b96b-93dbc235105e",
"@An.Int": 42
"@A.String":"plain text",
"@A.String#withAmp":"Fast&Furious",
"@A.String#ToBeEscaped":"A/\"good\"\r\nstory\\for\tkids",
"@A.TimeOfDay":"21:45:00",
```

4.5.3.2 Path Expressions

The expressions `edm:AnnotationPath`, `edm:NavigationPropertyPath`, `edm:Path`, and `edm:PropertyPath` are represented similar to individual properties or operation responses in **[OData-JSON]**, chapter 11, as a JSON object with a name/value pair `@odata.annotationPath`, `@odata.path`, or `@odata.propertyPath` whose value is a string containing the path expression.

Example 48: annotation with `edm:Path` dynamic expression

```
"@Org.OData.Measures.V1.ISOCurrency":{
  "@odata.path":"Currency"
}
```

4.5.3.3 Collection Expressions

The dynamic expression `edm:Collection` is represented as a JSON array. Its items are representations of its child expressions according to the rules defined in this specification.

4.5.3.4 Record Expressions

The dynamic expression `edm:Record` is represented as a JSON object. The `Type` attribute of the `edm:Record` expression is represented as an `@odata.type` annotation. Each `edm:PropertyValue` child element is represented as a name/value pair with the value of the `Property` attribute as name. Its value expression is represented according to the rules defined in this specification.

It may also contain annotations.

Example 49: annotation with `edm:Record` dynamic expression, one Boolean `edm:PropertyValue` and one with an `edm:Collection` value

```
"@Capabilities.UpdateRestrictions":{
  "Updatable":true,
  "NonUpdatableNavigationProperties":[
    {
      "@odata.navigationPropertyPath":"Supplier"
    },
    {
      "@odata.navigationPropertyPath":"Category"
    }
  ]
}
```

4.5.3.5 Comparison and Logical Operators and `If` Expression

The dynamic expression `edm:If` and the logical expressions `edm:Eq`, `edm:Ne`, `edm:Ge`, `edm:Gt`, `edm:Le`, `edm:Lt`, `edm:And`, and `edm:Or` are represented are represented as a JSON object with a name/value pair `@odata.if`, `@odata.eq` etc. whose value is a JSON array with items that are representations of the child expressions according to the rules defined in this specification.

It may also contain annotations.

Example 50: `edm:If` expression using an `edm:Path` expression as its condition and evaluating to one of two `edm:String` expressions

```
"@org.example.display.DisplayName":{
  "@odata.if":[
    {
      "@odata.path":"IsFemale"
    },
    "Female",
    "Male"
  ]
}
```

4.5.3.6 Expression `Apply`

The dynamic expression `edm:Apply` is represented as a JSON object with an `@odata.apply` name/value pair whose value is the function name as a string value. The child expressions are represented as a `parameterValues` name/value pair whose value is an array with items that are representations of the child expressions according to the rules defined in this specification.

It may also contain annotations.

Example 51: `edm:Apply` expression with two edm:String expressions and one `edm:Path` expression as parameter values

```
"@Some.Computed.Url":{
  "@odata.apply":{
    "function":"odata.concat",
    "parameterValues":[
      "Products(",
      {
        "@odata.path":"ID"
      },
      ")"
    ]
  }
}
```

4.5.3.7 Expressions `Cast` and `IsOf`

The dynamic expressions `edm:Cast` and `edm:IsOf` are represented as JSON objects with a name/value pair `@odata.cast` or `@odata.isOf` whose value is a string with a qualified type name. The facet attributes are represented as name/value pairs `maxLength`, `precision`, `scale`, and `srid`. The child expression is represented as the value of a `value` name/value pair according to the rules defined in this specification.

It may also contain annotations.

Example 52: `edm:IsOf` expression using an `edm:Path` expression as its parameter

```
"@For.Testing":{
  "@odata.isOf":"Edm.Binary",
  "value":{
    "@odata.path":"ImageData"
```

```
  }
}
```

4.5.3.8 Expression `LabeledElement`

The dynamic expression `edm:LabeledElement` is represented as a JSON object with an `@odata.labeledElement` name/value pair whose value is a string with the qualified name of the labeled element. Its single child expression is represented as a `value` name/value pair whose value is the representation of the child expression according to the rules defined in this specification.

It may also contain annotations.

Example 53: `edm:LabeledElement` expression

```
{
  "@odata.labeledElement":"Model.MyReusableAnnotation",
  "value":…,
}
```

4.5.3.9 Expression `LabeledElementReference`

The dynamic expression `edm:LabeledElementReference` is represented as a JSON object with an `@odata.labeledElementReference` name/value pair whose value is a string with the qualified name of the referenced labeled element.

Example 54: `edm:LabeledElementReference` expression

```
{
  "@odata.labeledElementReference":"Model.MyReusableAnnotation"
}
```

4.5.3.10 Expression `Not`

The dynamic expression `edm:Not` is represented as a JSON object with an `@odata.not` name/value pair whose value is the representation of the child expression according to the rules defined in this specification.

Example 55: `edm:Not` expression

```
"@Some.Term": {
  "@odata.not":{
    "@odata.path":"IsHappy"
  }
}
```

4.5.3.11 Expression `Null`

If the dynamic expression `edm:Null` contains annotations, it is represented as a JSON object with an `@odata.null` name/value pair whose value is an object that may contain annotations.

Example 56: `edm:Null` expression with nested annotations

```
"@Some.Term": {
  "@odata.null":{
    "@Within.Null": true
  }
}
```

Example 57: `edm:Null` expression without nested annotations

```
"@Some.Term": {
  "@odata.null":{
```

```
  }
}
```

4.5.3.12 Expression `UrlRef`

The dynamic expression `edm:UrlRef` is represented as a JSON object with an `@odata.type` name/value pair whose value is `#UrlRef`. Its single child expression is represented as a `value` name/value pair whose value is the representation of the child expression according to the rules defined in this specification.

Example 58: `edm:UrlRef` expressions with `edm:String` value and with `edm:Path` value

```
"@Good.Reference#one":{
  "@odata.urlRef":"http://www.odata.org"
},
"@Good.Reference#two":{
  "@odata.urlRef":{
    "@odata.path":"DocumentationUrl"
  }
}
```

4.5.3.13 Annotation `Core.Description`

The annotation `Core.Description` (see **[OData-VocCore]**) semantically corresponds to the JSON Schema keyword `description`, so unqualified annotations with `Core.Description` that have static content are represented with this keyword. Qualified annotations and annotations with dynamic content are represented as other annotations.

Example 59: unqualified static `Core.Description` as `description`

```
"org.example.Size": {
  "enum": [
    "S",
    "M",
    "L"
  ],
  "S@Core.Description": "Small",
  "M@Core.Description": "Medium",
  "L@Core.Description": "Large",
  "description": "T-Shirt Size",
  "@Core.Description#alt": "Size (S, M, L)",
  "@Core.LongDescription": "Size, expressed with letters familiar from e.g. T-
Shirt sizes",
 },
```

4.6 References

The value of `references` is an object with one name/value pair per referenced CSDL document. The name is the URI of the CSDL document. Its value is an object that may contain a name/value pair `includeAnnotations`.

It may contain annotations.

Example 60: references

```
"references": {
  "http://tinyurl.com/Org-OData-Measures-V1": {
    "@Some.Term": …
  },
  "http://somewhere/ExternalAnnotations": {
    "includeAnnotations": …
```

```
  }
}
```

4.6.1 IncludeAnnotations

The value `includeAnnotations` is an array of objects. Each object has a `termNamespace` name/value pair and may have name/value pairs `targetNamespace` and `qualifier`. The values of these name/value pairs are strings.

Example 61: includeAnnotations

```
"includeAnnotations": [ {
    "termNamespace": "Name.Space",
    "targetNamespace": "Target.Space"
  }, {
    "termNamespace": "Name.Space",
    "targetNamespace": "Target.Space",
    "qualifier": "SomeName"
  }, {
    "termNamespace": "NameSpace",
    "qualifier": "SomeName"
  }, {
    "termNamespace": "Name.Space"
  }
]
```

5 Extensions to JSON Schema

5.1 The edm.json Schema

The edm.json schema is an extension of JSON Schema Draft 04, see **[JS-Core]**. It defines reuse types for JSON CSDL documents as well as additional keywords.

The definitions object contains one name/value pair per OData primitive type, and one for the standard OData error response.

For each OData primitive type the corresponding schema states the JSON Schema primitive type (string, number, or integer) used to represent the OData primitive type, and additional restrictions on the values: pattern for strings, minimum and maximum for integers. In addition, some types specify a custom format.

A special case is the schema for Edm.Stream: it specifies an unfulfillable constraint on the values as stream properties don't have an inline representation in OData 4.0.

5.2 Keywords

OData CSDL contains many concepts that cannot be translated into JSON Schema, these are represented using the custom keywords. On the document root level these are

- actions
- entityContainer
- functions
- odata-version
- references
- schemas
- terms

JSON Schema objects of type object use the keywords

- abstract
- keys
- mediaEntity
- openType
- relationship

JSON Schema objects for primitive types may use the keywords

- precision
- scale

5.3 Formats

Not all constraints on values of OData primitive types can be expressed with standard JSON Schema means, and the format keyword of JSON Schema allows defining new values. CSDL JSON documents use the following custom formats:

Format	OData Type	Comment
base64url	Edm.Binary	OData-specific format
date	Edm.Date	Swagger format, was part of JSON Schema Draft 03
decimal	Edm.Decimal	OData-specific format

Format	OData Type	Comment
double	Edm.Double	Swagger format extended with -INF, INF, NaN
duration	Edm.Duration	OData-specific format
int16	Edm.Int16	OData-specific format
int32	Edm.Int32	Swagger format
int64	Edm.Int64	Swagger format
int8	Edm.SByte	OData-specific format
single	Edm.Single	OData-specific format
time	Edm.TimeOfDay	OData-specific format, was part of JSON Schema Draft 03
uint8	Edm.Byte	OData-specific format
uuid	Edm.Guid	OData-specific format

6 Validation

A JSON CSDL `$metadata` document contains definitions that can be used to validate request and response messages. Depending on the context URL a small wrapper schema has to be used that refers to the corresponding definition in the JSON `$metadata` document.

Example 62: a schema for validating messages containing a single `Product` entity

```
{
  "$schema":"http://json-schema.org/draft-04/schema#",
  "anyOf":[
    {
      "$ref": "csdl-16.1.json#/definitions/ODataDemo.Product"
    }
  ]
}
```

Example 63: a schema for validating messages containing a collection of `Product` entities

```
{
  "$schema":"http://json-schema.org/draft-04/schema#",
  "type":"object",
  "properties":{
    "value":{
      "type":"array",
      "items": {
        "$ref": "csdl-16.1.json#/definitions/ODataDemo.Product"
      }
    }
  }
}
```

7 Extensibility

Vocabularies and annotations already allow defining additional characteristics or capabilities of metadata elements, such as a service, entity type, property, function, action or parameter, and **[OData-CSDL]** defines which model elements can be annotated. This document specifies how these metadata annotations are represented in JSON CSDL documents.

8 CSDL Examples

Following are two basic examples of valid OData models as represented in JSON CSDL. These examples demonstrate many of the topics covered above. They represent the same documents as the XML examples in chapter 16 of **[OData-CSDL]**.

8.1 Products and Categories Example

Example 64:

```
{
 "$schema":"http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/edm.json#",
 "odata-version":"4.0",
 "definitions":{
  "ODataDemo.Product":{
   "type":"object",
   "mediaEntity":true,
   "keys":[
    "ID"
   ],
   "properties":{
    "ID":{
     "type":"string"
    },
    "Description":{
     "type":[
      "string",
      "null"
     ],
     "@Core.IsLanguageDependent":true
    },
    "ReleaseDate":{
     "type":[
      "string",
      "null"
     ],
     "format":"date"
    },
    "DiscontinuedDate":{
     "type":[
      "string",
      "null"
     ],
     "format":"date"
    },
    "Rating":{
     "type":[
      "integer",
      "null"
     ],
     "format":"int32"
    },
    "Price":{
     "type":[
      "number",
      "string",
      "null"
     ],
     "format":"decimal",
     "multipleOf":1,
```

```
    "@Org.OData.Measures.V1.ISOCurrency":{
     "@odata.path":"Currency"
    }
   },
   "Currency":{
    "type":[
     "string",
     "null"
    ],
    "maxLength":3
   },
   "Category":{
    "anyOf":[
     {
      "$ref":"#/definitions/ODataDemo.Category"
     }
    ],
    "relationship":{
     "partner":"Products"
    }
   },
   "Supplier":{
    "anyOf":[
     {
      "$ref":"#/definitions/ODataDemo.Supplier"
     },
     {
      "type":"null"
     }
    ],
    "relationship":{
     "partner":"Products"
    }
   }
  }
 },
 "ODataDemo.Category":{
  "type":"object",
  "keys":[
   "ID"
  ],
  "properties":{
   "ID":{
    "type":"integer",
    "format":"int32"
   },
   "Name":{
    "type":"string",
    "@Core.IsLanguageDependent":true
   },
   "Products":{
    "type":"array",
    "items":{
     "$ref":"#/definitions/ODataDemo.Product"
    },
    "relationship":{
     "partner":"Category",
     "onDelete":{
      "action":"Cascade"
     }
    }
   }
  }
 },
```



```
 "ODataDemo.Supplier":{
  "type":"object",
  "keys":[
   "ID"
  ],
  "properties":{
   "ID":{
    "type":"string"
   },
   "Name":{
    "type":[
     "string",
     "null"
    ]
   },
   "Address":{
    "$ref":"#/definitions/ODataDemo.Address"
   },
   "Concurrency":{
    "type":"integer",
    "format":"int32"
   },
   "Products":{
    "type":"array",
    "items":{
     "$ref":"#/definitions/ODataDemo.Product"
    },
    "relationship":{
     "partner":"Supplier"
    }
   }
  }
 },
 "ODataDemo.Country":{
  "type":"object",
  "keys":[
   "Code"
  ],
  "properties":{
   "Code":{
    "type":"string",
    "maxLength":2
   },
   "Name":{
    "type":[
     "string",
     "null"
    ]
   }
  }
 },
 "ODataDemo.Address":{
  "type":"object",
  "properties":{
   "Street":{
    "type":[
     "string",
     "null"
    ]
   },
   "City":{
    "type":[
     "string",
     "null"
```



```
     ]
    },
    "State":{
     "type":[
      "string",
      "null"
     ]
    },
    "ZipCode":{
     "type":[
      "string",
      "null"
     ]
    },
    "CountryName":{
     "type":[
      "string",
      "null"
     ]
    },
    "Country":{
     "anyOf":[
      {
       "$ref":"#/definitions/ODataDemo.Country"
      },
      {
       "type":"null"
      }
     ],
     "relationship":{
      "referentialConstraints":{
       "CountryName":{
        "referencedProperty":"Name"
       }
      }
     }
    }
   }
  }
 },
 "schemas":{
  "Org.OData.Core.V1":{
   "$ref":"http://docs.oasis-
open.org/odata/odata/v4.0/Org.OData.Core.V1.json#/schemas/Org.OData.Core.V1"
  },
  "Core":{
   "$ref":"http://docs.oasis-
open.org/odata/odata/v4.0/Org.OData.Core.V1.json#/schemas/Org.OData.Core.V1"
  },
  "Org.OData.Measures.V1":{
   "$ref":"http://docs.oasis-
open.org/odata/odata/v4.0/Org.OData.Measures.V1.json#/schemas/Org.OData.Measur
es.V1"
  },
   "UoM":{
   "$ref":"http://docs.oasis-
open.org/odata/odata/v4.0/Org.OData.Measures.V1.json#/schemas/Org.OData.Measur
es.V1"
  },
  "ODataDemo":{
  }
 },
 "functions":{
  "ODataDemo.ProductsByRating":[
```



```
   {
    "parameters":[
     {
      "name":"Rating",
      "parameterType":{
       "type":[
        "integer",
        "null"
       ],
       "format":"int32"
      }
     }
    ],
    "returnType":{
     "type":"array",
     "items":{
      "$ref":"#/definitions/ODataDemo.Product"
     }
    }
   }
  ]
 },
 "entityContainer":{
  "name":"DemoService",
  "entitySets":{
   "Products":{
    "entityType":{
     "$ref":"#/definitions/ODataDemo.Product"
    },
    "navigationPropertyBindings":{
     "Category":{
      "target":"Categories"
     }
    }
   },
   "Categories":{
    "entityType":{
     "$ref":"#/definitions/ODataDemo.Category"
    },
    "navigationPropertyBindings":{
     "Products":{
      "target":"Products"
     }
    }
   },
   "Suppliers":{
    "entityType":{
     "$ref":"#/definitions/ODataDemo.Supplier"
    },
    "navigationPropertyBindings":{
     "Products":{
      "target":"Products"
     },
     "Address/Country":{
      "target":"Countries"
     }
    },
    "@Core.OptimisticConcurrency":[
     {
      "@odata.propertyPath":"Concurrency"
     }
    ]
   },
   "Countries":{
```



```
    "entityType":{
     "$ref":"#/definitions/ODataDemo.Country"
    }
   }
  },
  "singletons":{
   "Contoso":{
    "type":{
     "$ref":"#/definitions/ODataDemo.Supplier"
    },
    "navigationPropertyBindings":{
     "Products":{
      "target":"Products"
     }
    }
   }
  },
  "functionImports":{
   "ProductsByRating":{
    "entitySet":"Products",
    "function":{
     "$ref":"#/functions/ODataDemo.ProductsByRating"
    }
   }
  }
 }
}
```

8.2 Annotations for Products and Categories Example

Example 65: schema `Annotations` contains annotations for referenced schema `ODataDemo` with terms from vocabulary `Some.Vocabulary.V1`

```
{
  "$schema":"http://docs.oasis-open.org/odata/odata-json-csdl/v4.0/edm.json#",
  "odata-version":"4.0",
  "schemas":{
    "ODataDemo":{
      "$ref":"http://host/service/$metadata#/schemas/ODataDemo"
    },
    "Some.Vocabulary.V1":{
      "$ref":"http://somewhere/Vocabulary/V1#/schemas/Some.Vocabulary.V1"
    },
    "Vocabulary1":{
      "$ref":"http://somewhere/Vocabulary/V1#/schemas/Some.Vocabulary.V1"
    },
    "Annotations":{
      "annotations":[
        {
          "target":"ODataDemo.Supplier",
          "@Vocabulary1.EMail":{
            "@odata.null":{}
          },
          "@Vocabulary1.AccountID":{
            "@odata.path":"ID"
          },
          "@Vocabulary1.Title":"Supplier Info",
          "@Vocabulary1.DisplayName":{
            "@odata.apply":"odata.concat",
            "parameterValues":[
              {
                "@odata.path":"Name"
              },
```

```
              " in ",
              {
                "@odata.path":"Address/CountryName"
              }
            ]
          }
        },
        {
          "target":"ODataDemo.Product",
          "@Self.Tags":[
             "MasterData"
          ]
        }
      ]
    }
  }
}
```


9 Conformance

Conforming services MUST follow all rules of this specification document for the types, sets, functions, actions, containers and annotations they expose.

Conforming clients MUST be prepared to consume a model that uses any or all of the constructs defined in this specification, including custom annotations, and MUST ignore any elements or attributes not defined in this version of the specification.

Appendix A. Acknowledgments

The contributions of the OASIS OData Technical Committee members, enumerated in **[OData-Protocol]**, are gratefully acknowledged.

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2015-11-20	Ralf Handl	Initial version