[MS-ODATA-Diff]:

Open Data Protocol (OData)

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
2/27/2009	0.1	Major	First Release.
4/10/2009	0.2	Minor	Clarified the meaning of the technical content.
5/22/2009	0.2.1	Editorial	Changed language and formatting in the technical content.
7/2/2009	1.0	Major	Updated and revised the technical content.
8/14/2009	1.1	Minor	Clarified the meaning of the technical content.
9/25/2009	1.2	Minor	Clarified the meaning of the technical content.
11/6/2009	1.3	Minor	Clarified the meaning of the technical content.
12/18/2009	1.3.1	Editorial	Changed language and formatting in the technical content.
1/29/2010	1.4	Minor	Clarified the meaning of the technical content.
3/12/2010	2.0	Major	Updated and revised the technical content.
4/23/2010	2.0.1	Editorial	Changed language and formatting in the technical content.
6/4/2010	3.0	Major	Updated and revised the technical content.
7/16/2010	4.0	Major	Updated and revised the technical content.
8/27/2010	5.0	Major	Updated and revised the technical content.
10/8/2010	5.1	Minor	Clarified the meaning of the technical content.
11/19/2010	6.0	Major	Updated and revised the technical content.
1/7/2011	7.0	Major	Updated and revised the technical content.
2/11/2011	8.0	Major	Updated and revised the technical content.
3/25/2011	9.0	Major	Updated and revised the technical content.
5/6/2011	10.0	Major	Updated and revised the technical content.
6/17/2011	10.1	Minor	Clarified the meaning of the technical content.
9/23/2011	11.0	Major	Updated and revised the technical content.
12/16/2011	12.0	Major	Updated and revised the technical content.
3/30/2012	13.0	Major	Updated and revised the technical content.
7/12/2012	13.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	14.0	Major	Updated and revised the technical content.
1/31/2013	15.0	Major	Updated and revised the technical content.
8/8/2013	16.0	Major	Updated and revised the technical content.
11/14/2013	17.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
2/13/2014	17.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	18.0	Major	Updated and revised the technical content.
6/30/2015	18.0	None	No changes to the meaning, language, or formatting of the technical content.
10/16/2015	18.0	None	No changes to the meaning, language, or formatting of the technical content.
5/10/2016	19.0	Major	Significantly changed the technical content.
7/14/2016	19.0	None	No changes to the meaning, language, or formatting of the technical content.
3/16/2017	20.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction 10
1.1 Glossary 10
1.2 References 12
1.2.1 Normative References 12
1.2.2 Informative References 14
1.3 Overview 14
1.4 Relationship to Other Protocols 14
1.5 Prerequisites/Preconditions 15
1.6 Applicability Statement 15
1.7 Versioning and Capability Negotiation 15
1.7.1 OData 2.0 Version-Specific Summary 17
1.7.2 OData 3.0 Version-Specific Summary 18
1.8 Vendor-Extensible Fields 23
1.9 Standards Assignments 23

2 Messages 24
2.1 Transport 24
2.2 Message Syntax 24
2.2.1 Abstract Data Model 24
2.2.1.1 Named Resource Stream 26
2.2.1.2 Named Resource Stream Instance 26
2.2.1.3 Actions 26
2.2.1.3.1 Action Metadata URL 27
2.2.1.4 Functions 28
2.2.1.4.1 Function Metadata URL 28
2.2.1.5 Service Operations 29
2.2.1.6 Containment 29
2.2.2 Abstract Type System 30
2.2.3 URI Format: Resource Addressing Rules 38
2.2.3.1 URI Syntax 39
2.2.3.2 Service Root (serviceRoot) and Path Prefix (pathPrefix) 44
2.2.3.3 Resource Path (resourcePath) 44
2.2.3.4 Resource Path: Construction Rules 44
2.2.3.5 Resource Path: Semantics 47
2.2.3.6 Query Options 56
2.2.3.6.1 System Query Options 57
2.2.3.6.1.1 Common Expression Syntax 59
2.2.3.6.1.1.1 Expression Construction and Evaluation Rules 64
2.2.3.6.1.1.2 Operator Precedence 85
2.2.3.6.1.1.3 Unary Numeric Promotions 86
2.2.3.6.1.1.4 Binary Numeric Promotions 86
2.2.3.6.1.1.5 Lifted Operators 87
2.2.3.6.1.1.6 Numeric Promotions for Method Call Parameters 88
2.2.3.6.1.1.7 Geospatial Coordinate Transformations 89
2.2.3.6.1.1.7.1 Coordinate Transformations Within a Topology 89
2.2.3.6.1.1.7.2 Arbitrary Coordinate Transformations 89
2.2.3.6.1.1.8 Geospatial Extension Methods 90
2.2.3.6.1.1.8.1 Extending Type Support for Defined Functions 90
2.2.3.6.1.1.8.2 Implementing One of the Functions Defined in [OGC-SFOLECOM] 90
2.2.3.6.1.1.8.3 Arbitrary Extensions 90
2.2.3.6.1.2 Evaluating System Query Options 91
2.2.3.6.1.3 Expand System Query Option ($expand) 91
2.2.3.6.1.4 Filter System Query Option ($filter) 92
2.2.3.6.1.5 Format System Query Option ($format) 93

2.2.3.6.1.6 OrderBy System Query Option ($orderby) 94
2.2.3.6.1.7 Skip System Query Option ($skip) 95
2.2.3.6.1.8 Top System Query Option ($top) 95
2.2.3.6.1.9 Skip Token System Query Option ($skiptoken) 96
2.2.3.6.1.10 InlineCount System Query Option ($inlinecount) 97
2.2.3.6.1.11 Select System Query Option ($select) 97
2.2.3.6.1.12 System Query Option: Additional Construction Rules 101
2.2.3.6.2 Custom Query Options 101
2.2.3.6.3 Service Operation Parameters 101
2.2.3.6.4 Function Parameters 101
2.2.3.6.5 Action Parameters 103
2.2.3.7 Data Service Metadata 104
2.2.3.7.1 Service Document 104
2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services 104
2.2.3.7.2.1 Conceptual Schema Definition Language Document Extensions for Customizable Feeds 108
2.2.3.8 URI Equivalence 111
2.2.3.9 Canonical URIs 111
2.2.4 HTTP Methods 112
2.2.4.1 PATCH/MERGE 112
2.2.5 HTTP Header Fields 113
2.2.5.1 Accept 113
2.2.5.1.1 application/atom+xml 114
2.2.5.1.2 application/json 115
2.2.5.1.3 application/json;odata=verbose 115
2.2.5.2 Content-Type 115
2.2.5.3 DataServiceVersion 115
2.2.5.4 ETag 116
2.2.5.5 If-Match 117
2.2.5.6 If-None-Match 118
2.2.5.7 MaxDataServiceVersion 118
2.2.5.8 X-HTTP-Method 119
2.2.5.9 Prefer 120
2.2.5.10 Preference-Applied 120
2.2.5.11 DataServiceId 121
2.2.6 Common Payload Syntax 121
2.2.6.1 Common Serialization Rules for XML-Based Formats 121
2.2.6.2 AtomPub Format 124
2.2.6.2.1 Entity Set (as an Atom Feed Element) 125
2.2.6.2.1.1 InlineCount Representation (for Collections of Entities) 126
2.2.6.2.1.2 Entity Set (as an Atom Feed Element) with Actions 127
2.2.6.2.1.3 Entity Set (as an Atom Feed Element) with Functions 128
2.2.6.2.2 Entity Type (as an Atom Entry Element) 129
2.2.6.2.2.1 Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping 131
2.2.6.2.2.2 Entity Type (as an Atom Entry Element) with Actions 132
2.2.6.2.2.3 Entity Type (as an Atom Entry Element) with Functions 133
2.2.6.2.3 Complex Type 133
2.2.6.2.4 Navigation Property 134
2.2.6.2.5 EDMSimpleType Property 134
2.2.6.2.6 Deferred Content 134
2.2.6.2.6.1 Inline Representation 135
2.2.6.2.7 Service Document 137
2.2.6.2.8 Additional Representations 137
2.2.6.2.9 Collection Property 137
2.2.6.2.9.1 Collection Property of Complex Type 138
2.2.6.2.9.2 Collection of EDMSimpleType 138
2.2.6.2.10 Named Resource Streams 138

2.2.6.2.11 Links and Subtypes....140
2.2.6.2.12 Annotations....140
2.2.6.3 Verbose JSON Format....143
2.2.6.3.1 Common Verbose JSON Serialization Rules for All EDM Constructs....143
2.2.6.3.1.1 Modifications to GeoJSON for Use in OData....148
2.2.6.3.2 Entity Set (as a Verbose JSON Array)....148
2.2.6.3.2.1 InlineCount Representation (for Collections of Entities)....149
2.2.6.3.2.2 Entity Set (as a Verbose JSON Array) with Actions....151
2.2.6.3.2.3 Entity Set (as a Verbose JSON Array) with Functions....152
2.2.6.3.3 Entity Type (as a Verbose JSON Object)....153
2.2.6.3.3.1 Entity Type (as a Verbose JSON Object) with Actions....157
2.2.6.3.3.2 Entity Type (as a Verbose JSON Object) with Functions....157
2.2.6.3.4 Complex Type....158
2.2.6.3.5 Collection of Complex Type Instances....158
2.2.6.3.6 Navigation Property....159
2.2.6.3.7 Collection of EDMSimpleType Values....160
2.2.6.3.8 EDMSimpleType Property....160
2.2.6.3.9 Deferred Content....161
2.2.6.3.9.1 Inline Representation....161
2.2.6.3.10 Links....163
2.2.6.3.11 InlineCount Representation (for Collections of Links)....164
2.2.6.3.12 Service Document....164
2.2.6.3.13 Collection Property....165
2.2.6.3.14 Named Resource Streams....165
2.2.6.3.15 Links and Subtypes....167
2.2.6.3.16 Annotations....167
2.2.6.4 Raw Format....167
2.2.6.4.1 EDMSimpleType Property....168
2.2.6.5 XML Format....168
2.2.6.5.1 Complex Type....168
2.2.6.5.2 Collection of Complex Type Instances....168
2.2.6.5.3 EDMSimpleType Property....169
2.2.6.5.4 Collection of EDMSimpleType Values....169
2.2.6.5.5 Links....169
2.2.6.5.5.1 InlineCount Representation (for Collections of Links)....170
2.2.6.5.5.2 Next Page (for Collections of Links)....170
2.2.6.5.6 Collection of Complex Type....170
2.2.6.5.7 Collection of EDMSimpleType....171
2.2.6.6 Preferred OData 3.0 JSON Format....171
2.2.7 Request Types....171
2.2.7.1 Insert Request Types....172
2.2.7.1.1 InsertEntity Request....172
2.2.7.1.1.1 Examples....174
2.2.7.1.2 InsertLink Request....179
2.2.7.1.3 InsertMediaResource Request....180
2.2.7.2 Retrieve Request Types....182
2.2.7.2.1 RetrieveEntitySet Request....182
2.2.7.2.2 RetrieveEntity Request....183
2.2.7.2.3 RetrieveComplexType Request....185
2.2.7.2.4 RetrievePrimitiveProperty Request....186
2.2.7.2.5 RetrieveValue Request....187
2.2.7.2.6 RetrieveCollectionProperty Request....188
2.2.7.2.7 RetrieveServiceMetadata Request....189
2.2.7.2.8 RetrieveServiceDocument Request....190
2.2.7.2.9 RetrieveLink Request....191
2.2.7.2.10 RetrieveCount Request....192
2.2.7.2.11 Retrieve Request Containing a Customizable Feed Mapping....193
2.2.7.2.12 RetrieveMediaResource Request....193

2.2.7.3 Update Request Types 194
2.2.7.3.1 UpdateEntity Request 194
2.2.7.3.1.1 Example 196
2.2.7.3.2 UpdateComplexType Request 196
2.2.7.3.3 UpdatePrimitiveProperty Request 198
2.2.7.3.4 UpdateCollectionProperty Request 199
2.2.7.3.5 UpdateValue Request 200
2.2.7.3.6 UpdateLink Request 201
2.2.7.3.7 UpdateMediaResource Request 203
2.2.7.3.8 Update Request Containing a Customizable Feed Property Mapping 204
2.2.7.4 Delete Request Types 204
2.2.7.4.1 DeleteEntity Request 204
2.2.7.4.2 DeleteLink Request 205
2.2.7.4.3 DeleteValue Request 206
2.2.7.5 Invoke Request Types 207
2.2.7.5.1 Invoke Action Request 208
2.2.7.5.2 Invoke Function Request 210
2.2.7.6 Batch Request 210
2.2.7.6.1 Change Set Syntax 211
2.2.7.6.1.1 Referencing Requests in a Change Set 212
2.2.7.6.2 Query Operation Syntax 212
2.2.7.6.3 HTTP Request Restrictions 213
2.2.7.6.4 Batch Request Syntax 213
2.2.7.6.5 Example Batch Request 214
2.2.7.6.6 Batch Responses 215
2.2.7.6.7 Batch Response Syntax 216
2.2.7.6.8 Example Batch Response 217
2.2.7.7 Tunneled Requests 219
2.2.8 Response Types 219
2.2.8.1 Error Response 219
2.2.8.1.1 XML Error Response 220
2.2.8.1.2 Verbose JSON Error Response 221

3 Protocol Details 222
3.1 Client Details 222
3.1.1 Abstract Data Model 222
3.1.2 Timers 222
3.1.3 Initialization 222
3.1.4 Higher-Layer Triggered Events 222
3.1.4.1 Common Rules for All Requests 222
3.1.4.2 Request to Insert Resources 222
3.1.4.2.1 Sending an InsertEntity Request 223
3.1.4.2.2 Sending an InsertLink Request 223
3.1.4.3 Request to Retrieve Resources 223
3.1.4.3.1 Common Rules for Sending Retrieve Requests 223
3.1.4.4 Request to Update Resources 224
3.1.4.4.1 Common Rules for Sending Update Requests 224
3.1.4.5 Request to Delete Resources 225
3.1.4.5.1 Common Rules for Sending Delete Requests 225
3.1.4.6 Request to Invoke a Service Operation 225
3.1.4.7 Request to Send a Batch of Operations 226
3.1.4.8 Request to Invoke an Action 226
3.1.4.9 Request to Invoke a Function 227
3.1.5 Message Processing Events and Sequencing Rules 227
3.1.5.1 Common Rules for Receiving Responses from Data Service Requests 227
3.1.5.2 Responses from Insert Requests 228
3.1.6 Timer Events 228
3.1.7 Other Local Events 228

3.2 Server Details 228
3.2.1 Abstract Data Model 228
3.2.2 Timers 228
3.2.3 Initialization 228
3.2.4 Higher-Layer Triggered Events 228
3.2.5 Message Processing Events and Sequencing Rules 228
3.2.5.1 Common Rules for Receiving All Data Service Requests 228
3.2.5.2 Common Rules for Executing Received Insert, Update, or Delete Data Service Requests 229
3.2.5.2.1 Common Rules for Executing Requests Containing a Customizable Feeds Mapped Property 230
3.2.5.3 Executing a Received Insert Request 230
3.2.5.3.1 Executing a Received InsertEntity Request 231
3.2.5.3.2 Executing a Received InsertLink Request 231
3.2.5.3.3 Executing a Received InsertMediaResource Request 231
3.2.5.4 Executing a Received Retrieve Request 232
3.2.5.4.1 Executing a Received RetrieveEntitySet Request 232
3.2.5.4.2 Executing a Received RetrieveValue Request 232
3.2.5.4.3 Executing a Received RetrieveCount Request 233
3.2.5.5 Executing a Received Update Request 233
3.2.5.5.1 Executing a Received UpdateEntity Request 234
3.2.5.6 Executing a Received Delete Request 235
3.2.5.7 Executing a Received Invoke Request 235
3.2.5.8 Executing a Received Batch Request 235
3.2.5.9 Executing a Received Invoke Action Request 236
3.2.5.10 Executing a Received Invoke Function Request 236
3.2.6 Timer Events 237
3.2.7 Other Local Events 237
3.2.8 Common Response Codes 237

4 Protocol Examples 238
4.1 Insert a New Entity 238
4.2 Retrieve Resources 238
4.2.1 Retrieve a Collection of Entities 238
4.2.1.1 Retrieve a Collection of Entities by Using the AtomPub Format 238
4.2.1.2 Retrieve a Collection of Entities by Using the Verbose JSON Format 239
4.2.1.3 Retrieve a Partial Collection of Entities by Using the AtomPub Format 240
4.2.1.4 Retrieve a Partial Collection of Entities by Using the Verbose JSON Format 242
4.2.1.5 Retrieve a Collection of Entities with an Inline Count by Using the AtomPub Format 243
4.2.1.6 Retrieve a Collection of Entities with an Inline Count by Using the Verbose JSON Format 244
4.2.1.7 Retrieve a Collection of Entities with Named Resource Streams by Using the AtomPub Format 245
4.2.1.8 Retrieve a Collection of Entities with Named Resource Streams by Using the Verbose JSON Format 247
4.2.2 Retrieve a Single Entity by Using the AtomPub Format 248
4.2.2.1 Retrieve a Single Entity with a Mapped Property by Using the AtomPub Format 249
4.2.3 Retrieve a Single Entity by Using the Verbose JSON Format 250
4.2.4 Retrieve a Single Entity and Its Directly Related Entities by Using the AtomPub Format 251
4.2.5 Retrieve a Single Entity and Its Directly Related Entities by Using the Verbose JSON Format 253
4.2.6 Retrieve a Data Service's Metadata Document (CSDL) 254
4.2.7 Retrieve the Count of a Collection of Entities 257
4.2.8 Retrieve a Single Entity Exposing an Action by Using the AtomPub Format 257
4.2.9 Retrieve a Single Entity Exposing an Action by Using the Verbose JSON Format 258

4.2.10 Retrieve a Single Entity Exposing a Function by Using the AtomPub Format 259
4.2.11 Retrieve a Single Entity Exposing a Function by Using the Verbose JSON Format 260
4.3 Update an Existing Entity 261
4.3.1 Replace-Based Update by Using the AtomPub Format 261
4.3.2 Replace-Based Update by Using the Verbose JSON Format 262
4.3.3 Merge-Based Update by Using the AtomPub Format 263
4.3.4 Merge-Based Update by Using the Verbose JSON Format 264
4.4 Update the Relationship Between Two Entities 265
4.4.1 Update a Relationship by Using the AtomPub Format 265
4.4.2 Update a Relationship by Using the Verbose JSON Format 265
4.4.3 Delete an Existing Entity 266
4.5 Batch Requests 266
4.6 Working with Media Resources (BLOBs) 266
4.6.1 Insert a New Media Resource 266
4.6.2 Update a Media Resource 267
4.6.3 Query an Existing Media Resource 268
4.7 Working with Named Resource Streams Instances (BLOBs) 268
4.7.1 Retrieving a Named Resource Stream Instance 268
4.7.2 Updating a Named Resource Stream Instance 269
4.7.3 Unsupported Operations 269
4.7.3.1 Inserting a New Named Resource Stream Instance 269
4.7.3.2 Deleting a New Named Resource Stream Instance 269
4.8 Invoking an Action 270
4.9 Invoking a Function 270

5 Security 272
5.1 Security Considerations for Implementers 272
5.2 Index of Security Parameters 272

6 Appendix A: Sample Entity Data Model and CSDL Document 273

7 Appendix B: Product Behavior 278

8 Change Tracking 282

9 Index 283

1 Introduction

The Open Data (OData) protocol enables applications that use common web technologies, like Atom Publishing Protocol (AtomPub), JavaScript Object Notation (JSON), and XML, to expose data as a data service that can be consumed by clients within corporate networks and across the Internet.

This document defines version 1.0, version 2.0, and version 3.0 of the Open Data (OData) protocol. The OData 3.0 protocol is a superset of OData 2.0, which, in turn, is a superset of OData 1.0. OData 3.0 includes incremental additions to OData 2.0, which, in turn, includes incremental additions to OData 1.0.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

alias: A simple identifier that is typically used as a short name for a namespace.

annotation: Any custom, application-specific extension that is applied to an instance of a schema definition language through the use of custom attributes and elements that are not a part of that schema definition language.

association: A named independent relationship between two entity type definitions. Associations in the Entity Data Model (EDM) are first-class concepts and are always bidirectional. Indeed, the first-class nature of associations helps distinguish the EDM from the relational model. Every association includes exactly two association ends.

Atom Publishing Protocol (AtomPub): An application-level protocol for publishing and editing web resources, as described in [RFC5023].

AtomPub collection: A set of resources that can be retrieved in whole or in part.

binary large object (BLOB): A collection of binary data stored as a single entity in a database.

bind: To associate two EntityType [MC-CSDL] instances. An EntityType instance in a data service (described by using Entity Data Model (EDM) constructs) may be related to one or more other conceptual schema definition language (CSDL) instances. This relationship is represented by using an association in an EDM. The cardinality of a relationship can be determined by inspecting the EDM that describes the data service. The act of associating two EntityType instances is known as "binding" and of disassociating two instances is known as "unbinding". If two EntityType instances are already associated, they are considered to be "bound".

cardinality: The measure of the number of elements in a set.

change set: A logical group of one or more insert, update, delete, or invoke request types that may be created by using the HTTP PUT, POST, PATCH, or DELETE method. All requests within a change set must be successfully processed. If any request in the change set fails, none of the requests within the change set should be processed.

collection: A grouping of one or more EDM types that are type compatible.

conceptual schema definition language (CSDL): A language that is based on XML and that can be used to define conceptual models that are based on the Entity Data Model (EDM).

conceptual schema definition language (CSDL) document: A document that contains a conceptual model that is described by using the CSDL code.

create retrieve update delete (CRUD): The four basic functions of persistent storage. The "C" stands for create, the "R" for retrieve, the "U" for update, and the "D" for delete. CRUD is used to denote these conceptual actions and does not imply the associated meaning in a particular technology area (such as in databases, file systems, and so on) unless that associated meaning is explicitly stated.

customizable feed: A property mapping that is used to define a mapping from the properties of an EntityType to elements or attributes in any namespace (including the Atom namespace) in an AtomPub document. When a property is mapped to an element or an attribute of an element, the value for the property is equal to the value of the specified element or attribute in the AtomPub document.

data service: A server-side application that implements the OData protocol for the purpose of enabling clients to publish and edit resources. The resources exposed by data services are described by using the EDM, as specified in [MC-CSDL].

declared property: A property that is statically declared by a Property element as part of the definition of a structural type. For example, in the context of an EntityType, a declared property includes all properties of an EntityType that are represented by the Property child elements of the EntityType element that defines the EntityType.

default EntityContainer: A single EntityContainer within a CSDL document, as specified in [MC-CSDL]. Entities in the default container may be identified in a data service URI without specifying the container name.

dynamic property: A designation for an instance of an OpenEntityType that includes additional nullable properties (of a scalar type or ComplexType) beyond its declared properties. The set of additional properties, and the type of each, may vary between instances of the same OpenEntityType. Such additional properties are referred to as dynamic properties and do not have a representation in a CSDL document.

entity: An instance of an EntityType element that has a unique identity and an independent existence. An entity is an operational unit of consistency.

Entity Data Model (EDM): A set of concepts that describes the structure of data, regardless of its stored form.

Entity Data Model Extensions (EDMX): An XML-based file format that serves as the packaging format for the service metadata of a data service, as specified in [MC-EDMX].

facet: An element that provides information that specializes the usage of a type. For example, the precision (that is, accuracy) facet can be used to define the precision of a DateTime property.

Internationalized Resource Identifier (IRI): A resource identifier that conforms to the rules for Internationalized Resource Identifiers, as defined in [RFC3987].

JavaScript Object Notation (JSON): A text-based, data interchange format that is used to transmit structured data, typically in Asynchronous JavaScript + XML (AJAX) web applications, as described in [RFC4627]. The JSON format is based on the structure of ECMAScript (Jscript, JavaScript) objects.

link: A link is similar to an association, as specified in [MC-CSDL], but describes a unidirectional relationship between entity types instead of a bidirectional one. A link can be either a unidirectional relationship that occurs when two entity types are related via an association and only one of the entity types defines a NavigationProperty that is bound to the association or a reference to one direction of a bidirectional association between two entity types, as specified in [MC-CSDL].

namespace: A name that is defined on the schema and that is subsequently used to prefix identifiers to form the namespace qualified name of a structural type.

primitive property: A property of type EDMSimpleType [MC-CSDL] that is defined on an EntityType.

property: An EntityType or ComplexType can have one or more properties of the specified EDMSimpleType or ComplexType. A property of an EntityType can be a declared property or a dynamic property, as specified in [MC-CSDL]. A property of ComplexType can only be a declared property.

query operation: A logical construct that must consist of a single retrieve request type or an invoke request that uses the HTTP GET method.

resource: A network-accessible data object or service that is identified by an IRI, as defined in [RFC2616].

resource path: The path of a data service URI, starting immediately after the service root and continuing to the end of the URI's path.

schema: A container that defines a namespace that describes the scope of EDM types. All EDM types are contained within some namespace.

service operation: An operation that is exposed by the data service. A service operation is represented as a FunctionImport, as specified in [MC-CSDL]. A service operation accepts only input parameters.

service root: A URI that represents the root of a data service.

Uniform Resource Identifier (URI): A string that identifies a resource. The URI is an addressing mechanism defined in Internet Engineering Task Force (IETF) Uniform Resource Identifier (URI): Generic Syntax [RFC3986].

XML namespace: A collection of names that is used to identify elements, types, and attributes in XML documents identified in a URI reference [RFC3986]. A combination of XML namespace and local name allows XML documents to use elements, types, and attributes that have the same names but come from different sources. For more information, see [XMLNS-2ED].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ECMA-262/51] ECMA~~, "~~ International, "Standard ECMA-262 ECMAScript Language Specification", 5.1 Edition ~~5.1, ECMA-262, (~~June 2011~~,)~~, http://www.ecma-international.org/~~publications/standards/Ecma~~ecma-262~~.htm~~/5.1/index.html

[GeoJSON] Butler, H., Daly, M., Doyle, A., et al., "The GeoJSON Format Specification", June 2008, http://geojson.org/geojson-spec.html

[IANA-MMT] IANA, "~~Mime~~ Media Types", ~~September 2012,~~ http://www.iana.org/assignments/media-types/media-types.xhtml

[IEEE754-2008] IEEE, "IEEE Standard for Binary Floating-Point Arithmetic", IEEE 754-2008, August 2008, http://~~ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&isnumber=4610934&arnumber=4610935&punumber=4610933~~standards.ieee.org/findstds/standard/754-2008.html

Note There is a charge to download the specification.

[MC-CSDL] Microsoft Corporation, "Conceptual Schema Definition File Format".

[MC-EDMX] Microsoft Corporation, "Entity Data Model for Data Services Packaging Format".

[MS-ODATAJSON] Microsoft Corporation, "OData Protocol JSON Format Standards Support Document".

[ODataJSON4.0] OASIS, "OData JSON Format Version 4.0", OASIS Standard, February 2014, http://docs.oasis-open.org/odata/odata-json-format/v4.0/os/odata-json-format-v4.0-os.doc

[OGC-SFOLECOM] Open GIS Consortium, "OpenGIS Simple Features Specification for OLE/COM Revision 1.1", 99-050, May 1999, http://portal.opengeospatial.org/files/?artifact_id=830

[RFC2045] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996, http://www.rfc-editor.org/rfc/rfc2045.txt

[RFC2046] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996, http://www.rfc-editor.org/rfc/rfc2046.txt

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., et al., "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999, http://www.rfc-editor.org/rfc/rfc2617.txt

[RFC3023] Murata, M., St.Laurent, S., and Kohn, D., "XML Media Types", RFC 3023, January 2001, http://www.~~ietf~~rfc-editor.org/rfc/rfc3023.txt

[RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004, http://www.~~ietf~~rfc-editor.org/rfc/rfc3676.txt

[RFC3986] Berners-Lee, T., Fielding, R., and Masinter, L., "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, http://www.~~ietf~~rfc-editor.org/rfc/rfc3986.txt

[RFC3987] Duerst, M., and Suignard, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005, http://www.~~ietf~~rfc-editor.org/rfc/rfc3987.txt

[RFC4287] Nottingham, M., and Sayre, R., Eds., "The Atom Syndication Format", RFC 4287, December 2005, http://www.~~ietf~~rfc-editor.org/rfc/rfc4287.txt

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006, http://www.rfc-editor.org/rfc/rfc4627.txt

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006, http://www.rfc-editor.org/rfc/rfc4648.txt

[RFC5023] Gregorio, J., and de hOra, B., Eds., "The Atom Publishing Protocol", RFC 5023, October 2007, http://www.~~ietf~~rfc-editor.org/rfc/rfc5023.txt

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.txt

[RFC5789] Dusseault, L., and Snell, J., "PATCH Method for HTTP", RFC 5789, March 2010, http://~~tools.ietf~~www.rfc-editor.org/~~html~~rfc/rfc5789.txt

[XML-BASE] Marsh, J., and Tobin, R., Eds., "XML Base (Second Edition)", W3C Recommendation, ~~December~~January 2009, http://www.w3.org/TR/2009/REC-xmlbase-20090128/

[XMLNS] Bray, T., Hollander, D., Layman, A., et al., Eds., "Namespaces in XML 1.0 (Third Edition)", W3C Recommendation, December 2009, http://www.w3.org/TR/2009/REC-xml-names-20091208/

[XMLSCHEMA1/2] Thompson, H., Beech, D., Maloney, M., and Mendelsohn, N., Eds., "XML Schema Part 1: Structures Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/

1.2.2 Informative References

[MS-NETOD] Microsoft Corporation, "Microsoft .NET Framework Protocols Overview".

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

1.3 Overview

The OData protocol is used to create Representational State Transfer (REST)-based [REST] data services which enable resources, that are identified by using Uniform Resource Identifiers (URIs) and are defined in an abstract data model, to be published and edited by web clients within corporate networks and across the Internet by using simple Hypertext Transfer Protocol (HTTP) messages.

The Atom Publishing Protocol (AtomPub) [RFC5023] does not define a URI-addressing scheme, a schema for the data content of the resources that the services expose, a format for batching requests, a concurrency policy or mechanism, or alternate data representations. The OData protocol defines a uniform, HTTP-based interface for data services that address these shortcomings of AtomPub. By using this interface, high-level, reusable, general-purpose client libraries and components can consume different services without needing to accommodate custom semantics for each.

The OData protocol depends on HTTP [RFC2616] for transfer of all protocol messages and user data and follow or extend the messaging semantics defined in AtomPub [RFC5023].

In this document, the endpoint that initiates the HTTP connection and sends HTTP request messages is referred to as the client. The entity that responds to the HTTP connection request and sends HTTP response messages is referred to as the server or data service. For the purposes of this document, the terms "server" and "data service" have the same meaning and are used interchangeably.

The use of web-based technologies, such as HTTP, make implementations of this document ideal as a mid-tier service technology for applications, such as Asynchronous JavaScript and XML (AJAX) style applications, Rich Interactive Applications (RIA), and other applications that operate against data that is stored across Internet trust boundaries.

1.4 Relationship to Other Protocols

This document defines version 1.0, version 2.0, and version 3.0 of the Open Data (OData) protocol. The OData protocol is based on the AtomPub format [RFC5023], which in turn, relies on HTTP [RFC2616]. Either HTTP 1.1 or HTTP 1.0 can be used with the OData protocol. Additionally, the OData

protocol uses HTTP headers that are defined in the HTTP specification but that are not referenced in the AtomPub specification.

The OData protocol also uses message formats that are defined by other industry standard specifications, such as the Multipurpose Internet Mail Extensions (MIME) format [RFC2046] and the JavaScript Object Notation (JSON) format [RFC4627].

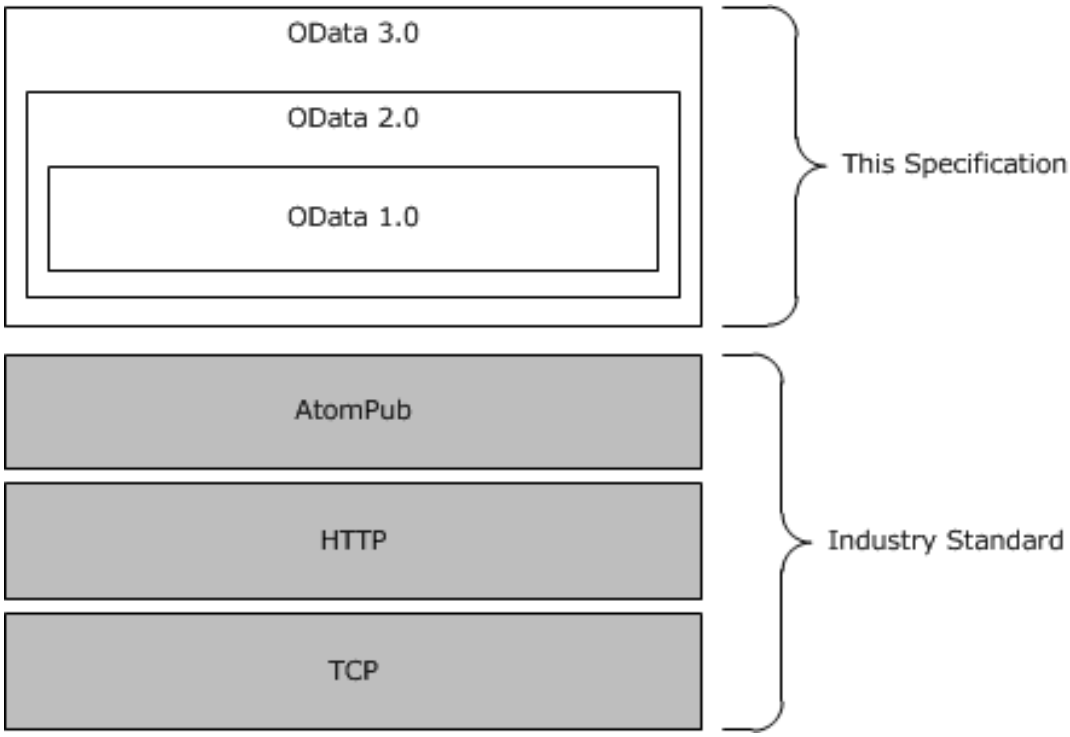

Figure 1: OData ~~Relationship~~relationship to ~~Other Protocols~~other protocols

1.5 Prerequisites/Preconditions

The OData protocol does not provide a mechanism for a client to discover the existence and location of arbitrary data services (of the server). It is a prerequisite that the client obtain a URI to the server before the protocol can be used.

Neither the Atom Publishing Protocol (AtomPub) nor the OData protocol defines an authentication or authorization scheme. Implementers of the protocol ought to review the recommended security prerequisites in Security Considerations for Implementers (section 5.1) of this document and in [RFC5023] section 15.

1.6 Applicability Statement

AtomPub, as specified in [RFC5023], in combination with the OData protocol, is appropriate for use in Web services that need a uniform, flexible, general purpose interface for exposing create retrieve update delete (CRUD) operations on a data model to clients. It is less suited to Web services that are primarily method-oriented or in which data operations are constrained to certain prescribed patterns.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas: <1>

Supported Transports: This document can be implemented on top of the Atom Publishing Protocol (AtomPub) described in Transport (section 2.1).

Protocol Versions: Clients specify the protocol version by using the DataServiceVersion (section 2.2.5.3) and MaxDataServiceVersion (section 2.2.5.7) request headers. Servers specify the protocol version by using the DataServiceVersion (section 2.2.5.3) response header.

Security and Authentication Methods: This document supports (but does not require) any authentication scheme that can be supported by using HTTP request and response headers. An example of such an authentication protocol is HTTP Basic Access Authentication described in [RFC2617].

Localization: This document does not specify any localization-dependent behavior.

Capability Negotiation: The OData protocol that is defined in this document enables limited capability negotiation using the DataServiceVersion (section 2.2.5.3) and MaxDataServiceVersion (section 2.2.5.7) version request headers and the DataServiceVersion (section 2.2.5.3) response header. These headers provide a way to version the OData protocol and do not act as a versioning scheme for the AtomPub in general.

In a request from the client to data service, the DataServiceVersion (section 2.2.5.3) and MaxDataServiceVersion (section 2.2.5.7) version headers can be specified.

If present in the request, the DataServiceVersion (section 2.2.5.3) header value states the version of the protocol used by the client to generate the request. If no DataServiceVersion (section 2.2.5.3) header is provided, the server assumes a value equal to the maximum version number the server supports.

If present in the request, the MaxDataServiceVersion (section 2.2.5.7) header value specifies the maximum version number the client can accept in a response. The client sets this value to the maximum version number of the protocol it is able to interpret. If the header is not present in a request, the server assumes the same version number as that specified by the DataServiceVersion (section 2.2.5.3) header. If a DataServiceVersion (section 2.2.5.3) header is not present, the server assumes the client can interpret the maximum version number the server can interpret.

When the server receives a request, it validates that the version number specified in the DataServiceVersion (section 2.2.5.3) header (or derived value if the header is not present) is less than or equal to the maximum version number it supports. If it is not, the server returns a response with a 4*xx* response code. The server also returns a description of the error by using the error format defined in Error Response (section 2.2.8.1).

In addition, a server validates that the version number specified in the MaxDataServiceVersion (section 2.2.5.7) header (or derived value if the header is not present) is greater than or equal to the minimum version number the server needs to use to generate the response. If it is not, the server returns an error response, described in Error Response (section 2.2.8.1).

In a response from the server to the client, the DataServiceVersion (section 2.2.5.3) header is specified. The value states the version of the protocol that the server used in the request to generate the response and that is used by the client to determine if it can correctly interpret the response (that is, the value is not larger than the value of the MaxDataServiceVersion (section 2.2.5.7) header sent in the associated request). The value of the headeris the lowest version of the protocol the server can use to fulfill the request.

1.7.1 OData 2.0 Version-Specific Summary

Following is a summary of the protocol constructs that are defined in this document that apply to OData 2.0 and OData 3.0. This section is structured by protocol feature, which is described and lists the sections that include content that is specific to that feature. Any constructs or semantics that exist only in OData 2.0 and OData 3.0 are explicitly denoted in that content.

Partial sets of entities: Servers responds to RetrieveEntitySet (section 2.2.7.2.1) GET requests with a response body containing a representation of a partial list of the entities that are identified by the request URI and a link to the next partial list.

- 2.2.3.6.1 System Query Options
- 2.2.3.6.1.9 Skip Token System Query Option ($skiptoken)
- 2.2.6.2.1 Entity Set (as an Atom Feed Element)
- 2.2.6.3.2 Entity Set (as a JSON Array)
- 3.2.5.4.1 Executing a Received RetrieveEntitySet Request

RetrieveCount request: The purpose of the RetrieveCount Request (section 2.2.7.2.10) is to enable the count of a collection of EntityType ([MC-CSDL] section 2.1.2) instances to be retrieved by the client.

- 2.2.3.1 URI Syntax
- 2.2.3.5 Resource Path: Semantics
- 2.2.3.6.1 System Query Options
- 2.2.7 Request Types
- 2.2.7.2.10 RetrieveCount Request
- 3.1.4.3.1 Common Rules for Sending Retrieve Requests
- 3.2.5.4.3 Executing a Received RetrieveCount Request

InlineCount system query option: A data service URI with an InlineCount system query option specifies that the response to the request includes the count N of the total number of entities in the **EntitySet** ([MC-CSDL] section 2.1.18) that are identified by the resource path section of the URI.

- 2.2.3.1 URI Syntax
- 2.2.3.5 Resource Path: Semantics
- 2.2.3.6.1 System Query Options
- 2.2.3.6.1.2 Evaluating System Query Options
- 2.2.3.6.1.10 InlineCount System Query Option ($inlinecount)
- 2.2.6.2.1.1 InlineCount Representation (for Collections of Entities)
- 2.2.6.3.2.1 InlineCount Representation (for Collections of Entities)
- 2.2.6.3.11 InlineCount Representation (for Collections of Links)
- 3.2.5.4 Executing a Received Retrieve Request

Select system query option: A data service URI with a **$select** system query option identifies the same set of entities as a URI without a **$select** query option. However, the presence of a **$select** query option specifies that a response from the data service returns a subset, as identified by the value of the **$select** query option, of the properties that would have been returned had the URI not included a **$select** query option.

- 2.2.3.1 URI Syntax
- 2.2.3.6.1 System Query Options
- 2.2.3.6.1.2 Evaluating System Query Options
- 2.2.3.6.1.11 Select System Query Option ($select)

Customizable feeds: Customizable feed property mappings can be used to override an entity type's default AtomPub representation and specify how one or more properties of an entity type are to be represented within an AtomPub **atom:entry** element. This feature of the protocol specifies a set of data service metadata document (see section 2.2.3.7.2) annotations, which enable a property of an entity type to be mapped to a child element of an **atom:entry** element, or an XML attribute on the **atom:entry** element, or one of its child elements. When a property is mapped to an element, the value for the property is used as the value of the mapped-to element or attribute.

- 2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services
- 2.2.3.7.2.1 Conceptual Schema Definition Language Document Extensions for Customizable Feeds
- 2.2.6.2.2 Entity Type (as an Atom Entry Element)
- 2.2.6.2.2.1 Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping
- 2.2.7.1.1 InsertEntity Request
- 2.2.7.2.11 Retrieve Request Containing a Customizable Feed Mapping
- 2.2.7.3.1 UpdateEntity Request
- 2.2.7.3.8 Update Request Containing a Customizable Feed Mapping
- 3.2.5.2.1 Common Rules for Executing Requests Containing a Customizable Feeds Mapped Property

Revised Verbose JSON response format: The Verbose JSON representation for collections has been enhanced to allow for the representation of additional collection-level metadata.

- 2.2.6.3 JavaScript Object Notation (JSON) Format

1.7.2 OData 3.0 Version-Specific Summary

Following is a summary of the protocol constructs that are defined in this document that apply to OData 3.0. This section is structured by protocol feature, which is briefly described, and lists the sections that include content that is specific to that feature. Any constructs or semantics that exist only in OData 3.0 are explicitly denoted in that content.

Collection properties: A collection property is a property that represents a non-nullable, unordered, homogenous set of **EDMSimpleType** or **ComplexType**, as specified in [MC-CSDL].

- 2.2.1 Abstract Data Model

- 2.2.3.1 URI Syntax
- 2.2.3.4 Resource Path: Construction Rules
- 2.2.3.6.1 System Query Options
- 2.2.6.2.2 Entity Type (as an Atom Entry Element)
- 2.2.6.2.9 Collection Property
- 2.2.6.2.9.1 Collection Property of Complex Type
- 2.2.6.2.9.2 Collection of EDMSimpleType
- 2.2.6.3.13 Collection Property
- 2.2.6.2.9.1 Collection Property of Complex Type
- 2.2.6.2.9.2 Collection of EDMSimpleType
- 2.2.7.2.6 RetrieveCollectionProperty Request
- 2.2.7.3.4 UpdateCollectionProperty Request

Geospatial properties: A geospatial property is a property of a geospatial **EDMSimpleType**, as specified in [MC-CSDL] (section 2.2.1).

- 2.2.2 Abstract Type System
- 2.2.3.6.1.1 Common Expression Syntax
- 2.2.3.6.1.1.1 Expression Construction and Evaluation Rules
- 2.2.3.6.1.1.7 Geospatial Coordinate Transformations
- 2.2.3.6.1.1.7.1 Coordinate Transformations Within a Topology
- 2.2.3.6.1.1.7.2 Arbitrary Coordinate Transformations
- 2.2.3.6.1.1.8 Geospatial Extension Methods
- 2.2.3.6.1.1.8.1 Extending Type Support for Defined Functions
- 2.2.3.6.1.1.8.2 Implementing One of the Functions Defined in [OGC-SFOLECOM]
- 2.2.3.6.1.1.8.3 Arbitrary Extensions
- 2.2.6.1 Common Serialization Rules for XML-based Formats
- 2.2.6.2 AtomPub Format
- 2.2.6.3.1 Common Verbose JSON Serialization Rules for All EDM Constructs
- 2.2.6.3.1.1 Modifications to GeoJSON [GeoJSON] for use in OData
- 2.2.6.3.9.1 Inline Representation

Relationship links: Relationship links can be used to represent the association between related entities.

- 2.2.6.2.1.1 InlineCount Representation (for Collections of Entities)
- 2.2.6.2.2 Entity Type (as an Atom Entry Element)

- 2.2.6.2.4 Navigation Property
- 2.2.6.2.6.1 Inline Representation
- 2.2.6.3.2.1 InlineCount Representation (for collections of entities)
- 2.2.6.3.3 Entity Type (as a JSON object)
- 2.2.6.3.6 Navigation Property
- 2.2.6.3.9 Deferred Content
- 2.2.6.3.9.1 Inline Representation
- 2.2.7.1.1.1 Examples

PATCH method: The HTTP PATCH method is defined by [RFC5789] and defines behavior that is equivalent to the HTTP MERGE method (section 2.2.4.1) that is defined in this specification.

- 2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services
- 2.2.4.1 PATCH/MERGE
- 2.2.5.5 If-Match
- 2.2.5.6 If-None-Match
- 2.2.5.8 X-HTTP-Method
- 2.2.7.3.1 UpdateEntity Request
- 2.2.7.3.2 UpdateComplexType Request
- 2.2.7.3.3 UpdatePrimitiveProperty Request
- 2.2.7.3.4 UpdateCollectionProperty Request
- 2.2.7.3.5 UpdateValue Request
- 2.2.7.3.6 UpdateLink Request
- 2.2.7.3.7 UpdateMediaResource Request
- 3.2.5.2 Common Rules for Executing Received Insert, Update, or Delete Data Service Requests
- 3.2.5.5 Executing a Received Update Request

Prefer header: The prefer header allows a client to request that the server include a representation of the resource that was updated during a HTTP POST, MERGE, PUT, or PATCH operation in the body of the response.

- 2.2.5.9 Prefer
- 2.2.5.10 Preference-Applied
- 2.2.5.11 DataServiceId
- 2.2.7.1.1 InsertEntity Request
- 2.2.7.1.2 InsertLink Request
- 2.2.7.1.3 InsertMediaResource Request

- 2.2.7.3.1 UpdateEntity Request
- 2.2.7.3.2 UpdateComplexType Request
- 2.2.7.3.3 UpdatePrimitiveProperty Request
- 2.2.7.3.4 UpdateCollectionProperty Request
- 2.2.7.3.5 UpdateValue Request
- 2.2.7.3.6 UpdateLink Request
- 2.2.7.3.7 UpdateMediaResource Request

Named resource streams: Named resource streams extend an **EntityType** definition to include an optional set of named streams (or byte arrays).

- 2.2.1 Abstract Data Model
- 2.2.3.1 URI Syntax
- 2.2.3.3 Resource Path (resourcePath)
- 2.2.3.4 Resource Path: Construction Rules
- 2.2.3.5 Resource Path: Semantics
- 2.2.3.6.1.11 Select System Query Option ($select)
- 2.2.3.6.1.12 System Query Option: Additional Construction Rules
- 2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services
- 2.2.3.7.2.1 Conceptual Schema Definition Language Document Extensions for Customizable Feeds
- 2.2.6.2.10 Named Resource Streams
- 2.2.6.3.14 Named Resource Streams

Any/All method support: This feature adds support for the Any method and for the All method.

- 2.2.3.6.1.1 Common Expression Syntax
- 2.2.3.6.1.1.1 Expression Construction and Evaluation Rules
- 2.2.3.6.1.4 Filter System Query Option ($filter)

Derived types: This feature adds support for indicating a more specific subtype in various contexts, such as navigation and filtering. Derived types also allow references to members of a given subtype.

- 2.2.3 URI Format: Resource Addressing Rules
- 2.2.3.1 URI Syntax
- 2.2.3.5 Resource Path: Semantics
- 2.2.3.6.1 System Query Options
- 2.2.3.6.1.1 Common Expression Syntax
- 2.2.3.6.1.3 Expand System Query Option ($expand)

- 2.2.3.6.1.4 Filter System Query Option ($filter)
- 2.2.3.6.1.6 OrderBy System Query Option ($orderby)
- 2.2.3.6.1.11 Select System Query Option ($select)
- 2.2.6.2.11 Links and Subtypes

Actions and functions: Actions provide a way to define and invoke side effecting operations that are associated with an entity or a collection of entities. Functions provide a way to define and invoke operations that are free of side effects.

- 2.2.1.3 Actions
- 2.2.1.3.1 Action Metadata URL
- 2.2.1.4 Functions
- 2.2.1.4.1 Function Metadata URL
- 2.2.1.5 Service Operations
- 2.2.3.1 URI Syntax
- 2.2.3.4 Resource Path: Construction Rules
- 2.2.3.5 Resource Path: Semantics
- 2.2.3.6.1.1 Common Expression Syntax
- 2.2.3.6.1.1.1 Expression Construction and Evaluation Rules
- 2.2.3.6.1.3 Expand System Query Option ($expand)
- 2.2.3.6.1.11 Select System Query Option ($select)
- 2.2.3.6.4 Function Parameters
- 2.2.3.6.5 Action Parameters
- 2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services
- 2.2.5.5 If-Match
- 2.2.6.2.1.2 Entity Set (as an Atom Feed Element) with Actions
- 2.2.6.2.1.3 Entity Set (as an Atom Feed Element) with Functions
- 2.2.6.2.2.2 Entity Type (as an Atom Entry Element) with Actions
- 2.2.6.2.2.3 Entity Type (as an Atom Entry Element) with Functions
- 2.2.6.3.2.2 Entity Set (as a JSON array) with Actions
- 2.2.6.3.2.3 Entity Set (as a JSON array) with Functions
- 2.2.6.3.3.1 Entity Type (as a JSON Object) with Actions
- 2.2.6.3.3.2 Entity Type (as a JSON Object) with Functions
- 2.2.7.5.1 Invoke Action Request
- 2.2.7.5.2 Invoke Function Request

- 3.1.4.8 Request to Invoke an Action
- 3.1.4.9 Request to Invoke a Function
- 3.2.5.9 Executing a Received Invoke Action Request
- 3.2.5.10 Executing a Received Invoke Function Request

Containment: Containment provides a way to model situations in which an **EntityType** is contained by another **EntityType**. This implies constraints on how to access, create, and update the contained **EntityType**.

- 2.2.1.6 Containment
- 2.2.3.4 Resource Path: Construction Rules
- 2.2.3.7.1 Service Document
- 2.2.3.9 Canonical URIs

New JSON format: A new JSON format provides a preferred JSON with optional metadata that more closely resembles custom JSON formats.

- 2.2.5.1.2 application/json
- 2.2.5.2 Content-Type
- 2.2.6 Common Payload Syntax

1.8 Vendor-Extensible Fields

The AtomPub-based messages defined in Messages (section 2) can be extended by adding additional elements or attributes. Such extensions cannot be in any of the namespaces that are listed in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

Additional extensibility rules are defined in [RFC5023] section 6.2.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The OData protocol that is defined in this specification and the Atom Publishing Protocol (AtomPub) [RFC5023] both use HTTP (as specified in [RFC2616]) as the transport layer. HTTP operations are performed on resources identified by a URI. URI Format: Resource Addressing Rules (section 2.2.3) describes the resource addressing rules defined by this specification which extend the addressing rules used by AtomPub and HTTP.

A TCP port has not been reserved for this protocol. TCP port 80 is commonly used because many HTTP proxy servers forward only HTTP traffic that use port 80.

This specification does not prescribe a mechanism to secure (authenticate, encrypt, and so on) AtomPub communications. For security recommendations which relate to the protocol's transport layer, see [RFC5023] section 15.

2.2 Message Syntax

This section includes the following:

Abstract Data Model (section 2.2.1) specifies the key concepts of the abstract data model that serve as the basis for the OData protocol. The subsequent sections each define mappings from the data model to the OData protocol.

Abstract Type System (section 2.2.2) specifies the abstract type system that is used to define the primitive types (such as **String**, **Boolean**, and so on) that are used by the OData protocol.

URI Format: Resource Addressing Rules (section 2.2.3) specifies a set of rules that are used to construct the URIs that identify each of the constructs in the data model, described in Abstract Data Model (section 2.2.1), and that are relevant to the OData protocol.

HTTP Methods (section 2.2.4) specifies how HTTP PATCH and MERGE are used to do partial updates in the OData protocol.

HTTP Header Fields (section 2.2.5) specifies the syntax for the HTTP headers that are defined in this document.

The HTTP payload format syntax in Common Payload Syntax (section 2.2.6) specifies how data that is described by using the abstract data model in Abstract Data Model (section 2.2.1) is mapped to the AtomPub, JSON, and Verbose JSON serialization formats for use in the payloads of the HTTP request types that are described in Request Types (section 2.2.7).

Request Types (section 2.2.7) specifies the types of requests that are defined by this document and how each request type is mapped to AtomPub request types as well as to constructs in the data model that is described in Abstract Data Model (section 2.2.1).

Response Types (section 2.2.8) specifies how error responses are mapped to XML and Verbose JSON formats.

Note All the example URIs and message payloads used in this section and throughout the remainder of this document are based on the sample conceptual schema definition language (CSDL) document in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

2.2.1 Abstract Data Model

This section describes a data modeling vocabulary that a server uses to describe the data it exposes. This modeling vocabulary is also used in subsequent sections of this document to describe data as

exchanged by this document. The use of this modeling vocabulary does not mandate a particular data persistence format or implementation on the server, as long as the server's interface is consistent with the OData protocol.

The OData protocol uses the Entity Data Model (EDM) as its data modeling vocabulary. Data models can be described in EDM terms using a conceptual schema definition language (CSDL) document [MC-CSDL]. The remainder of this section provides a brief description of the EDM and defines how EDM constructs are mapped to the resource types defined in the AtomPub specification.

Entity Data Model: The central concepts in the EDM are entities and associations. Entities are instances of **EntityType** (such as Customer, Employee, and so on) that are structured records that consist of named and typed properties and that include a key.

A **ComplexType** ([MC-CSDL] section 2.1.7) is a structured type that also consists of a list of properties. However, a **ComplexType** does not have a key and thus can exist only as a property of a containing entity or as a temporary value.

A service operation represents a **FunctionImport**, as specified in [MC-CSDL], that accepts only input parameters.

A **Collection** type is a non-nullable, unordered, homogenous set of types **EDMSimpleType** or **ComplexType**.

An **EntityKey** ([MC-CSDL] section 2.1.5) is formed from a subset of properties of the **EntityType**. The **EntityKey** (such as CustomerId, OrderId, and so on) is a fundamental concept to uniquely identify instances of **EntityType** and allows **EntityType** instances to participate in relationships. Entities are grouped in EntitySets (for example, Customers is a set of Customer instances).

An association defines the relationship between one EntityType and another (for example, Employee Works For Department). Instances of associations are grouped in AssociationSets ([MC-CSDL] section 2.1.19).

NavigationProperties ([MC-CSDL] section 2.1.4) are special properties on an **EntityType** that are bound to a specific association and can be used to refer to associations through an entity instead of explicitly through an association instance.

Finally, all instance containers (**EntitySet** and **AssociationSet**) are grouped in an **EntityContainer** ([MC-CSDL] section 2.1.14).

EDM constructs map to the data model concepts used in the AtomPub specification as shown in the following Entity Data Model Concepts Mapped to AtomPub Resource Types table. Common Payload Syntax (section 2.2.6) describes how these conceptual AtomPub resources (such as collection and Entry Resource) are represented using multiple formats in request and response messages used by this document.

Entity Data Model	AtomPub Resource Classification
EntitySet	Collection
EntityType with **m:HasStream** equals true	Media Link Entry Resource
EntityType instance	Entry Resource
NavigationProperty	**atom:link** element
Named resource stream instance	**atom:link** element
FunctionImport with **Binding** equals true and **IsSideEffecting** equals true	**m:action** element

Entity Data Model	AtomPub Resource Classification
FunctionImport with **Binding** equals true and **IsSideEffecting** equals false	**m:function** element

Table: Entity Data Model Concepts Mapped to AtomPub Resource Types

2.2.1.1 Named Resource Stream

Applies to the OData 3.0 protocol

Named resource streams are properties of an **EntityType** that are of type **Edm.Stream**, where the Name of the named resource stream is simply the name of the property. **Edm.Stream** is an **EDMSimpleType** introduced in version 2.2 of the Entity Data Model (EDM).

Named resource streams represent and point at unstructured streams of data (such as images or documents). Each named resource stream MUST have a name, which is a simple identifier, as specified in [MC-CSDL] (section 2.2.6). For each named resource stream that is defined on an **EntityType**, any corresponding **EntityType** instance MUST have a corresponding named resource stream instance.

Named resource streams are supported only in the OData 3.0 protocol.

2.2.1.2 Named Resource Stream Instance

Applies to the OData 3.0 protocol

A named resource stream instance is a named resource stream that is associated with the **Edm.Stream** property instance with the same name on a particular **EntityType** instance. Each named resource stream instance:

- MUST have a **Name** that matches the **Name** of a named resource stream that is defined on either the **EntityType** or one of its **BaseTypes**.
- MUST have a content type, as specified in [RFC2616].
- Can have a self link, which is a URI from which the actual stream of bytes can be retrieved with a GET request.
- Can have an edit link, which is a URI that can be used to replace the existing stream with a PUT request.
- MUST have a self link or an edit link or both an edit link and a self link specified.
- Can have an **ETag**. If present, the **ETag** MUST represent the concurrency token that is associated with the named resource stream instance ETag (section 2.2.5.4).

2.2.1.3 Actions

Applies to the OData 3.0 protocol

An action is an operation that can be bound to its first parameter. After the action is invoked, the action might have a side effect.

Actions are represented by using a **FunctionImport** that:

- Is marked as **IsSideEffecting**, either because **IsSideEffecting** is set to "true" or because **IsSideEffecting** is omitted and thus defaults to "true".

- Might have **IsBindable** set to "true".
- Has a first parameter with a type that is either an **EntityType** or a collection of an **EntityType** if **IsBindable** is set to "true".
- Is marked as noncomposable, either because **IsComposable** is set to "false" or because **IsComposable** is omitted and thus defaults to "false".
- If the **ReturnType** of the action is either an **EntityType** or a collection of an **EntityType**, the action has the **EntitySet** attribute set either to an **EntitySet** name or an **EntitySetPathExpression** that is specified relative to the **EntitySet** of the **Binding** parameter.
- Does not have an **HttpMethod** data service annotations attribute.
- Might have an **IsAlwaysBindable** data service annotation attribute set to either "true" or "false", if **IsBindable** is set to "true". This attribute indicates whether all instances of the first parameter type (the **Binding** parameter) can be bound to by the action. This means that servers can omit action information when serializing an entry and its applicable actions if the server follows conventions in order to be more efficient. When omitted, the **IsAlwaysBindable** data service annotations attribute defaults to "false".
- Does not have a declaring **EntityContainer** that declares another **FunctionImport** with the same **Name**. This means that actions are not to have overloads.

2.2.1.3.1 Action Metadata URL

Applies to the OData 3.0 protocol

An action metadata URL is a URL string that points to the metadata of an action. An action metadata URL follows these rules:

- If the conceptual schema definition language (CSDL) document for data services (section 2.2.3.7.2) can be found by URI convention (section 2.2.3.1) and the action is defined in that document, the path to the metadata can be established by convention.
- If the path to the metadata can be established by convention, the action metadata URL SHOULD NOT include the metadata URL to the CSDL document.
- If the path to the metadata cannot be established by convention, the action metadata URL MUST begin with an absolute URL that locates the CSDL document that defines the action, followed by a "#".
- The **EntityContainer** name that contains the **FunctionImport** that defines the action MUST be unique across the data service.
- A **FunctionImport** that represents an action MUST NOT have any overloads.

The rules for constructing the action metadata URL are as follows.

```
actionMetadataUrl = [metadataUrl] "#" ecQualifiedActionName

ecQualifiedActionName = entityContainer "." actionName

entityContainer       = *pchar   ; section 3.3 of [RFC3986]
                      ; the name of an Entity Container in the EDM model

actionName            = *pchar ; section 3.3 of [RFC3986]
                      ; the name of a FunctionImport in the EDM model that defines an Action

metadataUrl           = ; a url that points to the Conceptual Schema Definition Language
                     ; Document that contains the definition of the Action.
```

Listing: ABNF Rule for Constructing the Action Metadata URL

2.2.1.4 Functions

Applies to the OData 3.0 protocol

A function is an operation that can optionally be bound to its first parameter and that has no side effects when it is invoked.

Functions are represented by using a **FunctionImport** element that:

- Has **IsSideEffecting** set to "false".
- Can have **IsBindable** set to "true" if the function has a least one parameter.
- Can have **IsComposable** set to "true" or "false".
- Has a **ReturnType**.
- If the **ReturnType** of the action is either an **EntityType** or a collection of **EntityType**, the function has the **EntitySet** attribute set to either an **EntitySet** name or an **EntitySetPathExpression** that is specified relative to the **EntitySet** of the **Binding** parameter.
- Does not have an **HttpMethod** data service annotations attribute.
- Can have an **IsAlwaysBindable** data service annotations attribute set to either "true" or "false", if **IsBindable** is set to "true". This attribute indicates whether all instances of the first parameter type (the **Binding** parameter) can be bound to by this function. This allows servers to omit function information when serializing an entry and its applicable functions in an effort to be more efficient. When omitted, **IsAlwaysBindable** defaults to "false".
- Represent functions that can have overloads where an overload is defined as two or more **FunctionImport** elements with the same name but a different set of parameters.
- If a **FunctionImport** has overloads, either the set of parameters for each **FunctionImport** element MUST have a different number of parameters or, if the number of parameters is the same, the ordered set of parameter types MUST be different. If the unordered sets of parameters types are the same, each unordered set of parameter names MUST differ.

2.2.1.4.1 Function Metadata URL

Applies to the OData 3.0 protocol

A function metadata URL is a URL that points to the definition of a function or group of function overloads.

The rules for constructing a function metadata URL are similar to those for constructing an action metadata URL (section 2.2.1.3.1) with one difference: functions MAY have overloads. Because functions can have overloads, the function metadata URL has to include information about the types of the parameters in order to uniquely identify a particular function overload when overloads exist. When no type information is specified, the function metadata URL identifies all overloads for the identified function.

The rules for constructing the function metadata URL are as follows.

```
functionMetadataUrl = [metadataUrl] "#" ecQualifiedFunctionName

ecQualifiedFunctionName = entityContainer "." functionName [ "(" parameterTypeNames ")" ]
                        ; the parameterTypeNames are required to uniquely identify the Function
                        ; only if the Function in question has overloads.
```

```
entityContainer       = ; section 2.2.1.3.1
                      ; the name of an Entity Container in the EDM model

functionName          = *pchar ; section 3.3 of [RFC3986]
                      ; the name of a FunctionImport in the EDM model that defines a function

parameterTypeNames    = [ parameterTypeName *( "," parameterTypeName ) ")" ]
                      ; the types of all the parameters to the corresponding functionImport
                      ; in the order they are declared in the FunctionImport

parameterTypeName     = namespaceQualifiedType ; section 2.2.3.1

metadataUrl           = ; section 2.2.1.3.1
```

Listing: ABNF Rule for Constructing the Function Metadata URL

2.2.1.5 Service Operations

Applies to the OData 3.0 protocol

A service operation is an operation that is described by using a **FunctionImport**.

Service operations can be distinguished from both actions and functions because they MUST have an **HttpMethod** data service annotation attribute on the corresponding **FunctionImport.**

As of the OData 3.0 protocol, the use of service operations, while still supported, is no longer recommended. Everything that is possible with a service operation can be achieved by using either actions or functions in OData 3.0.

Service operations are described in detail in sections 2.2.7.5. and 2.2.3.7.2.

2.2.1.6 Containment

Applies to the OData 3.0 protocol

Containment is specified by using a containment **NavigationProperty**. A containment **NavigationProperty** is a **NavigationProperty** that has a **ContainsTarget** attribute set to "true".

The **EntityType** that declares the **NavigationProperty** is the container **EntityType**.

The **AssociationType** that is specified in the containment **NavigationProperty** is the containment **AssociationType**.

The **EntityType** specified on the **End** of the containment **AssociationType**, with the **Name** specified by the containment **NavigationProperty ToRole** attribute, is the contained **EntityType**.

When the instances of both contained and container entities reside in the same **EntitySet**, this is called recursive containment.

It MUST NOT be possible for an **EntityType** to contain itself by following more than one containment **NavigationProperty**.

The contained **EntityType** MAY have a **NavigationProperty** that navigates to the container **EntityType** via the containment **AssociationType**. The **End** of the containment **AssociationType** that is specified by the **ToRole** attribute of the containment **NavigationProperty** a have any multiplicity.

For nonrecursive containment, the **End** of the containment **AssociationType** that is specified by the **FromRole** attribute of the containment **NavigationProperty** MUST have a multiplicity of '1'.

For recursive containment, the **End** of the containment **AssociationType** that is specified by the **FromRole** attribute of the containment **NavigationProperty** MUST have a multiplicity of '0..1'. Additionally, the **End** specified by the **ToRole** MUST NOT have a multiplicity of '1' because this would lead to endless recursion.

An **AssociationSet** MUST have the same **EntitySet** on both ends if it is for a containment **AssociationType** that has either the same **EntityType** on both ends or an **EntityType** on one end that derives from an **EntityType** on the other end.

An **EntitySet** MUST NOT be bound by **AssociationSet** to more than one **AssociationType** via a containment **NavigationProperty** that indicates that the **EntityType** (or derived **EntityTypes**) of that **EntitySet** is contained.

Note Because the **EntityType** of **EntitySets** on an **AssociationSet End** is the same as or derived from the **EntityTypes** on the corresponding **AssociationType Ends**, an **EntitySet** is either completely contained or completely noncontained.

Nonrecursive contained **EntitySets** are accessed only through the URL of the resource that represents the containment **NavigationProperty** on the parent **Entity** instance. Clients MUST NOT assume that nonrecursive contained **EntitySets** can be accessed directly from the **ServiceRoot**.

Recursively contained **EntitySets** MUST be accessible directly from the **ServiceRoot** for querying and inserting "root" entity instances that are not contained by another instance. Entity instances that are contained by another entity instance, the querying and inserting operations are supported via the URL of the resource that represents the container **NavigationProperty** on the parent entity instance.

When the end of the containment **AssociationType** that is specified by the **ToRole** attribute of the containment **NavigationProperty** is '1' or '0..1', there is no way to independently insert the contained entity by using a simple InsertEntity Request (section 2.2.7.1.1). A simple **InsertEntity** request would require supporting POST to a URL that represents a **ReferenceSet**, and a POST to a URL that represents a **ReferenceSet** is inconsistent with existing protocol semantics. This type of contained entity can still be created either by using a "deep insert" (section 2.2.7.1.1) that creates both the container and the contained entities in one request or by invoking an action (section 4.8) that creates the contained entity.

2.2.2 Abstract Type System

The abstract type system that is used to define the primitive types that are supported by a data service is defined in [MC-CSDL] (section 2.2.1). When the value of a primitive type needs to be represented in a URI or HTTP header, the representation that is written uses the primitive type literal form representation that is defined in the following table. A primitive type representation in a request or a response payload is defined in format-specific sections of this document.

The following listing that follows the grammar rules in this section makes reference to the following shared ABNF [RFC5234] grammar rules.

```
SQUOTE            =  %x27             ; ' (single quote)
EQ                =  %x3D             ; = (equal sign)
SEMI              =  %x3B             ; ; (semicolon)
SP                =  %x20             ;   (single-width horizontal space character)
COMMA             =  %x2C             ; , (comma)
nonZeroDigit      =  %x31-39          ; all digits except zero
doubleZeroToSixty =  "0" DIGIT
                     / "1" DIGIT
                     / "2" DIGIT
                     / "3" DIGIT
                     / "4" DIGIT
                     / "5" DIGIT
nan               =  "NaN"
negativeInfinity  =  "-INF"
positiveInfinity  =  "INF"
```

```
nanInfinity       =  nan / negativeInfinity / positiveInfinity
sign              =  "-" / ""
DIGIT             =  ; see [RFC5234] Appendix B.1 Core Rules
UTF8-char         =  ; see [RFC3629]
```

The following table defines the literal form representations of Entity Data Model (EDM) primitive types.

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
null	nullLiteral	`nullLiteral = "null"`

Edm.Binary	binaryLiteral	`binaryUriLiteral = caseSensitiveToken SQUOTE binaryLiteral SQUOTE`
		`binaryLiteral = hexDigPair`
		`caseSensitiveToken = "X" / "binary"`
		`; X is case sensitive, and binary is not case sensitive`
		`hexDigPair = 2*HEXDIG`
		`[hexDigPair]`

Edm.Boolean	booleanLiteral	`booleanLiteral = true / false`
		`true = "true" / "1"`
		`; clients/servers SHOULD also recognize "True"`
		`false = "false" / "0"`
		`; clients/servers SHOULD also recognize "False"`

Edm.Byte	byteLiteral	`byteLiteral = 1*3DIGIT;`
		`; For further information on the value range for`
		`; the Edm.Byte type, see [MC-CSDL]`

Edm.DateTime	dateTimeUriLiteral	`datetimeUriLiteral = "datetime" SQUOTE dateTimeLiteral SQUOTE` `dateTimeLiteral = year "-" month "-"`

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
		day "T" hour ":" minute [":" second ["." nanoSeconds]] [timeZone] year = 4Digit; month = <any number between 1 and 12 inclusive> day = nonZeroDigit / ("0" nonZeroDigit) /("1" DIGIT) / ("2" DIGIT) / "3" ("0" / "1") hour = nonZeroDigit / ("0" nonZeroDigit) / ("1" DIGIT) / ("2" zeroToFour) zeroToFour= <any nuumber between 0 and 4 inclusive> minute =doubleZeroToSixty second = doubleZeroToSixty nanoSeconds= 1*7Digit timeZone = (("+" / "-") hour ':' minute) / 'Z'
Edm.Decimal	decimalUriLiteral	decimalUriLiteral = decimalLiteral ("M"/"m") decimalLiteral = sign 1*29DIGIT ["." 1*29DIGIT]
Edm.Double	doubleLiteral	doubleLiteral = ((nonDecimalPoint / nonExpDecimal / expDecimal) "D" / "d") / (nanInfinity ["D" / "d"]) nonDecimalPoint= sign 1*17DIGIT nonExpDecimal = sign* DIGIT "." *DIGIT expDecimal = sign 1*DIGIT "." 16DIGIT

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
		("e" / "E") sign 1*3DIGIT ; for additional information on the value range ; of the Edm.Double type, see [MC-CSDL]
Edm.Single	singleUriLiteral	singleUriLiteral = (singleLiteral ("F" / "f"))/ (nanInfinity ["F" / "f"]) singleLiteral = nonDecimalPoint / nonExpDecimal / expDecimal nonDecimalPoint = sign 1*8DIGIT nonExpDecimal = sign *DIGIT "." *DIGIT expDecimal = sign 1*DIGIT "." 8DIGIT ("e" / "E") sign 1*2DIGIT ; for additional information on the value range ; of the Edm.Single type, see [MC-CSDL]
Edm.Float	singleLiteral	See **Edm.Single**.
Edm.Guid	guidUriLiteral	guidUriLiteral= "guid" SQUOTE guidLiteral SQUOTE guidLiteral = 8HEXDIG "-" 4HEXDIG "-" 4HEXDIG "-" 4HEXDIG "-" 12HEXDIG
Edm.Int16	int16Literal	int16Literal= sign 1*5DIGIT

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
Edm.Int32	int32Literal	`int32Literal= sign 1*10DIGIT`
Edm.Int64	int64UriLiteral	`int64UriLiteral= int64Literal ("L" / "l")` `int64Literal = sign 1*19DIGIT`
Edm.SByte	sbyteliteral	`sbyteliteral= sign 1*3DIGIT`

Edm.String	stringUriLiteral	`stringUriLiteral = SQUOTE [*characters] SQUOTE`
		
		`characters = UTF8-char`

Edm.Time	timeUriLiteral	`timeUriLiteral =`
		`"time"`
		`SQUOTE`
		`timeLiteral`
		`SQUOTE`
		
		`timeLiteral = <Defined by the lexical representation for dayTimeDuration in [XMLSCHEMA1.1/2:2012]>`

Edm.DateTimeOffset	dateTimeOffsetUriLiteral	`dateTimeOffsetUriLiteral =`
		`"datetimeoffset"`
		`SQUOTE`
		`dateTimeOffsetLiteral`
		`SQUOTE`
		
		`dateTimeOffsetLiteral = <Defined by the lexical representation for datetime (including timezone offset) in [XMLSCHEMA2/2]. The timezone offset is required.>`

Edm.Geography	N/A	N/A

Edm.GeographyPoint	geographyFullPointLiteral
	The two doubles in position literals are to be interpreted as longitude, then latitude.
		`geographyFullPointLiteral =`
		`geographyPrefix`
		`fullPointLiteral`
		`SQUOTE`
		
		`geographyPrefix =`
		`"geography"`
		`SQUOTE`

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)

		fullPointLiteral =
		  sridLiteral
		  pointLiteral
		
		pointLiteral =
		  "Point"
		  pointData
		
		pointData =
		  "("
		  positionLiteral
		  ")"
		
		positionLiteral =
		  doubleLiteral
		  SP
		  doubleLiteral
		
		sridLiteral =
		  "SRID"
		  EQ
		  1*5DIGIT
		  SEMI

Edm.GeographyLineString	geographyFullLineStringLiteral
	The two doubles in position literals are to be interpreted as longitude, then latitude.
		geographyFullLineStringLiteral =
		  geographyPrefix
		  fullLineStringLiteral
		  SQUOTE
		
		fullLineStringLiteral =
		  sridLiteral
		  lineStringLiteral
		
		lineStringLiteral =
		  "LineString"
		  lineStringData
		
		lineStringData =
		  "("
		  positionLiteral
		  [COMMA positionLiteral]+
		  ")"

Edm.GeographyPolygon	geographyFullPolygonLiteral
	The two doubles in position literals are to be interpreted as longitude, then latitude.
		geographyFullPolygonLiteral =
		  geographyPrefix
		  fullPolygonLiteral
		  SQUOTE
		
		fullPolygonLiteral =
		  sridLiteral
		  polygonLiteral
		
		polygonLiteral =
		  "Polygon"
		  polygonData
		
		polygonData =
		  "("
		  ringLiteral

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
		[COMMA ringLiteral]* ")" ringLiteral = "(" firstPosition [COMMA positionLiteral]* COMMA firstPosition ")" firstPosition = positionLiteral Within each ringLiteral, the two firstPosition elements MUST be an exact syntactic match to each other. Within the polygonData, the ringLiterals MUST specify their points in appropriate winding order. In order of traversal, points to the left side of the ring are interpreted as being in the polygon.
Edm.GeographyCollection	geographyFullGeoCollectionLiteral The two doubles in position literals are to be interpreted as longitude, then latitude.	geographyFullGeoCollectionLiteral = geographyPrefix fullGeoCollectionLiteral SQUOTE fullGeoCollectionLiteral = sridLiteral geoCollectionLiteral geoCollectionLiteral = "GeometryCollection(" geoLiteral [COMMA geoLiteral]* ")" geoLiteral = pointLiteral | lineStringLiteral | polygonLiteral | geoCollectionLiteral | multiPointLiteral | multiLineStringLiteral | multiPolygonLiteral
Edm.GeographyMultiPoint	geographyFullMultiPointLiteral The two doubles in position literals are to be interpreted as longitude, then latitude.	geographyFullMultiPointLiteral = sridLiteral multiPointLiteral multiPointLiteral = "MultiPoint(" [pointData [COMMA pointData]*

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)
		]? ")"
Edm.GeographyMultiLineString	geographyFullMultiLineStringLiteral The two doubles in position literals are to be interpreted as longitude, then latitude.	geographyFullMultiLineStringLiteral = geographyPrefix fullMultiLineStringLiteral SQUOTE fullMultiLineStringLiteral = sridLiteral multiLineStringLiteral multiLineStringLiteral = "MultiLineString(" [lineStringData [COMMA lineStringData]*]? ")"
Edm.GeographyMultiPolygon	geographyFullMultiPolygonLiteral The two doubles in position literals are to be interpreted as longitude, then latitude.	geographyFullMultiPolygonLiteral = geographyPrefix fullMultiPolygonLiteral SQUOTE fullMultiPolygonLiteral = sridLiteral multiPolygonLiteral multiPolygonLiteral = "MultiPolygon(" [polygonData [COMMA polygonData]*]? ")"
Edm.Geometry	N/A	N/A
Edm.GeometryPoint	geometryFullPointLiteral The two doubles in position literals are to be interpreted as X, then Y.	geometryFullPointLiteral = geometryPrefix fullPointLiteral SQUOTE geometryPrefix = "geometry" SQUOTE
Edm.GeometryLineString	geometryFullLineStringLiteral The two doubles in position literals are to be interpreted as X, then Y.	geometryFullLineStringLiteral = geometryPrefix fullLineStringLiteral SQUOTE

EDM primitive type	ABNF rule for primitive type representation in URIs and HTTP headers	Primitive type literal form (ABNF definition)

Edm.GeometryPolygon	geometryFullPolygonLiteral
	The two doubles in position literals are to be interpreted as X, then Y.
		`geometryFullPolygonLiteral =`
		`  geometryPrefix`
		`  fullPolygonLiteral`
		`  SQUOTE`

Edm.GeometryCollection	geometryFullGeoCollectionLiteral
	The two doubles in position literals are to be interpreted as X, then Y.
		`geometryFullGeoCollectionLiteral`
		`=`
		`  geometryPrefix`
		`  fullGeoCollectionLiteral`
		`  SQUOTE`

Edm.GeometryMultiPoint	geometryFullMultiPointLiteral
	The two doubles in position literals are to be interpreted as X, then Y.
		`geometryFullMultiPointLiteral =`
		`  geometryPrefix`
		`  fullMultiPointLiteral`
		`  SQUOTE`

Edm.GeometryMultiLineString	geometryFullMultiLineStringLiteral
	The two doubles in position literals are to be interpreted as X, then Y.
		`geometryFullMultiLineStringLiter`
		`al =`
		`  geometryPrefix`
		`  fullMultiLineStringLiteral`
		`  SQUOTE`

Edm.GeometryMultiPolygon	geometryFullMultiPolygonLiteral
	The two doubles in position literals are to be interpreted as X, then Y.
		`geometryFullMultiPolygonLiteral`
		`=`
		`  geometryPrefix`
		`  fullMultiPolygonLiteral`
		`  SQUOTE`

Edm.Stream	N/A	N/A
Collection	N/A	N/A

Table: Literal Form of Entity Data Model Primitive Types

2.2.3 URI Format: Resource Addressing Rules

The Atom Publishing (AtomPub) Protocol specifies operations for publishing and editing resources by using HTTP, but does not constrain the form of the URIs, as specified in [RFC3986], that are used to identify the resources (see [RFC5023] section 4.1). This document extends AtomPub by defining a mapping from elements in an Entity Data Model (EDM), described by using a conceptual schema definition language (CSDL) document, to the resource types defined in [RFC5023] section 4.2. See Abstract Data Model (section 2.2.1) for a mapping of EDM constructs to AtomPub resources.

As specified in [RFC5023] (section 4.1), the Atom Publishing Protocol [RFC5023] specifies the formats of the representations that are exchanged and the actions that can be performed on the Internationalized Resource Identifiers (IRIs) embedded in those representations. AtomPub does not constrain the form of the URIs that are used. Following that paradigm, this section and its subsections

defines a set of recommended, but not required, rules for constructing a URI or IRI to identify the various parts of the data and metadata in an EDM.

Servers and clients MAY use alternate URI path construction rules because HTTP [RFC2616] specifies that the URI space of each server is controlled by that server. The OData protocol imposes no further constraints on that control when constructing the authority and path segments of a URI. However, servers that conform to this specification MUST honor the rules for query string construction as defined in this section and its subsections.

Server authors are encouraged to follow the URI path construction rules, in addition to the required query string rules, that are defined in this specification when possible. Such consistency promotes an ecosystem of reusable client components and libraries.

Before an IRI can be used by an HTTP request, the IRI is first converted to a URI, according to the algorithm defined in [RFC3987] section 3.1. For the remainder of this document, the term URI is used to refer to a URI or an IRI that has been converted to a URI.

2.2.3.1 URI Syntax

The Augmented BNF for URI Construction listing in this section specifies that a data service URI (see dataSvcAbs-URI) is comprised of four sections: the scheme [RFC3986], a data service root or path prefix, a resource path, and query options, which, when composed, form an absolute URI to address any EntitySet, EntityType instance, property, or service operation result in an Entity Data Model (EDM).

Servers that conform to this specification MAY follow the grammar below when constructing the scheme, service root, and resource path URI components of a data service URI. All servers MUST follow these grammar rules when constructing and parsing the query options section of a data service URI.

```
dataSvcAbs-URI      = scheme           ; see section 3.1 of [RFC3986]
                      host             ; section 3.2.2 of [RFC3986]
                      [ ":" port ]     ; section 3.2.3 of [RFC3986]
                      (serviceRoot ["/$metadata" / "/$batch"]) ; see section 2.2.3.2
                      / (pathPrefix [dataSvcRel-URI])

dataSvcAbsNqo-URI   = scheme
                      ; see section 3.1 of [RFC3986]
                      serviceRoot           ; see section 2.2.3.2
                      [resourcePath]

dataSvcRel-URI      = resourcePath ["?" queryOptions ]   ; see section 2.2.3.3

serviceRoot         =
                      *( "/" segment-nz ) ; section 3.3 of [RFC3986]
                         ; segment-nz = the non empty sequence of characters
                         ;              outside the set of URI reserved
                         ;              characters as specified in [RFC3986]

pathPrefix          = *( "/" segment-nz )
                      ; zero or more URI path segments

resourcePath        = "/"
                      (
                        (
                          [entityContainer "."] entitySet
                          / serviceOperation-collEt

                        )
                        [ paren ] ["/" namespaceQualifiedEntityType] [ navPath ] [ count ]
                      )
                      / functionCall ["/" namespaceQualifiedEntityType] [ navPath ] [ count ]
                      / serviceOperation ["/" namespaceQualifiedEntityType]
```

```
                      / actionCall

paren               = "()"

serviceOperation    = serviceOperation-et
                      / serviceOperation-collCt
                      / serviceOperation-ct
                      / serviceOperation-collPrim
                      / serviceOperation-prim [ value ]
                      / serviceOperation-void

count               = "/$count"
     ; count is supported only in OData 2.0 and OData 3.0

navPath             = ( "("keyPredicate")"  [navPath-options] ) /
                      operation

operation           = "/"
                      ( actionCall /
                        (functionCall-partiallyBound [navPath-options])
                      )
     ; operation segments can only be composed if the type of theprevious segment matches
     ; the type of the first parameter of the action or function being called.

actionCall          = actionFQName "()"
                      ; TODO: parameters to actions are provided in the BODY
                      ; TOOD: we are considering allowing some parameters in the URL

actionFQName        = [ entityContainer "." ] actionName

actionName         = ; section 2.2.1.3.1
                   ; name of an Action defined by a FunctionImport in the EDM model
                   ; associated with this data service.


functionFQName      = [ entityContainer "." ] functionName

functionName        = ; section 2.2.1.4.1
                   ; name of a function defined by a FunctionImport in the EDM model
                   ; associated with this data service.

functionCall        = functionFQName "(" [functionParameters] ")"
                   ; if this function call is the last function call in the path,
                   ; from left to right, then the parameters MAY be provided in the query
                   ; part of the URI, without using parameterAlias(es) but instead using the
                   ; names of the FunctionImport parameters, as per serviceOperations.

functionParameters  = ( functionParameter *( "," functionParameter ) )

functionParameter   = functionParameterName "=" primitiveValue / functionParameterAlias /
null

functionParameterName = *pchar
                   ; the name of the parameter as found in the corresponding FunctionImport
                   ; definition.

functionCall-partiallyBound = functionFQName "(" [functionParameters-unbound] ")"

functionParameters-unbound  = functionParameter-unbound *( "," functionParameter-unbound )
                   ; if this function call is the last function call in the path,
                   ; from left to right, then the parameters MAY be provided in the query
                   ; part of the URI, as per serviceOperation(s), except the bound parameter
                   ; which is specified in the path prefix to the function call.

functionParameter-unbound = functionParameter
                      ; with the added restriction that the parameter must not be a
                      ; binding parameter

functionParameterAlias      = @ *pchar     ; i.e. @parameterName
```

```
navPath-options     = [
                        navPath-np /
                        propertyPath /
                        propertyPath-ct /
                        namedStreamPath /
                        value /
                        operation
                       ]

navPath-np          = ["/"namespaceQualifiedEntityType] "/"
                      ( ("$links" / entityNavProperty )
                        / (entityNavProperty-es [ paren ] [ navPath ])
                        / (entityNavProperty-et [ navPath-options ]))

entityNavProperty = (entityNavProperty-es [ paren ])
                      / entityNavProperty-et

propertyPath        = ["/"namespaceQualifiedEntityType]"/" (entityProperty [value]) /
entityCollectionProperty

propertyPath-ct     = 1*(["/" namespaceQualifiedEntityType]"/" entityComplexProperty) [
propertyPath ]

namedStreamPath     = ["/" namespaceQualifiedEntityType]"/" entityNamedStream
; the namedStreamPath is supported only in OData 3.0
keyPredicate        = keyPredicate-single
                      / keyPredicate-cmplx

keyPredicate-single = primitiveValue

primitiveValue      = 1*DIGIT                                  ; section B.1 of [RFC5234]
                      / ([1*unreserved] "'" 1*unreserved "'") ; section 2.3 of [RFC3986]
                      / 1*(HEXDIG HEXDIG))                     ; section B.1 of [RFC5234]

namespaceQualifiedType = namespaceQualifiedComplexType |
           namespaceQualifiedEntityType |
           primitiveType |
           "Collection(" namespaceQualifiedEntityType ")" |
           "Collection("
               ( namespaceQualifiedComplexType | primitiveType )
           ")"

namespaceQualifiedEntityType  = namespace "." entityType
    ; the namespaceQualifiedEntityType is supported only in OData 3.0

namespaceQualifiedComplexType = namespace "." complexType

namespace = *pchar   ; section 3.3 of [RFC3986]
     ; the Namespace of the schema of the EDM model where an EntityType(s),
     ; ComplexType(s) or PrimitiveType(s) is defined.
primitiveType = [ "Edm." ] primitiveTypeName
  null = "null" [ "'" namespaceQualifiedType "'" ]
         ; the optional namespaceQualifiedType is used to specify what type this
         ; null value should be considered for function overload resolution purposes.
primitiveTypeName = "binary" |
                    "boolean" |
                    "byte" |
                    "datetime" |
                    "decimal" |
                    "double" |
                    "single" |
                    "float" |
                    "guid" |
                    "int16" |
                    "int32" |
                    "int64" |
                    "sbyte" |
                    "string" |
                    "time" |
                    "datetimeoffset" |
```

```
                    "stream" |
                    concreteSpatialTypeName |
                    abstractSpatialTypeName


concreteSpatialTypeName  = "point" |
                       "linestring" |
                       "polygon" |
                       "geographycollection" |
                       "multipoint" |
                       "multilinedtring" |
                       "multipolygon" |
                       "geometricpoint" |
                       "geometriclinestring" |
                       "geometricpolygon" |
                       "geometrycollection" |
                       "geometricmultipoint" |
                       "geometricmultilinestring" |
                       "geometricmultipolygon" |

abstractSpatialTypeName  = "geography" |
                         "geometry" |

keyPredicate-cmplx  = entityProperty "=" keyPredicate-single
                      ["," keyPredicate-cmplx]

value               = "/$value"

queryOptions = sysQueryOption        ; see section 2.2.3.6.1
              / customQueryOption   ; section 2.2.3.6.2
              / serviceOpParam      ; see section 2.2.3.6.3
              / functionParameter       ; see section 2.2.3.6.4
             *("&"(sysQueryOption / serviceOpParam
             / customQueryOption / functionParameter))

sysQueryOption    = expandQueryOp
                  / filterQueryOp
                  / orderbyQueryOp
                  / skipQueryOp
                  / topQueryOp
                  / formatQueryOp
                  / countQueryOp
                  / selectQueryOp
                  / skiptokenQueryOpcustomQueryOption   = *pchar        ; section 3.3 of
[RFC3986]

expandQueryOp       = ; see section 2.2.3.6.1.3

filterQueryOp       = ; see section 2.2.3.6.1.4

orderbyQueryOp      = ; see section 2.2.3.6.1.6

skipQueryOp         = ; see section 2.2.3.6.1.7

serviceOpArg        = ; see section 2.2.3.6.3

topQueryOp          = ; see section 2.2.3.6.1.8

formatQueryOp       = ; see section 2.2.3.6.1.5

countQueryOp        = ; see section 2.2.3.6.1.10
     ; the countQueryOp is supported only in OData 2.0 and OData 3.0

selectQueryOp       = ; see section 2.2.3.6.1.11

skiptokenQueryOp    = ; see section 2.2.3.6.1.9

;Note: The semantic meaning, relationship to Entity Data Model
;      (EDM) constructs and additional URI construction
;      constraints for the following grammar rules are further
```

```
;      defined in (section 2.2.3.4) and (section 2.2.3.5)

; See [MC-CSDL] for further scoping rules regarding the value
;      of each of the rules below

entityContainer     = ; section 2.2.1.3.1
     ; the name of an Entity Container in the EDM model

entitySet           = *pchar   ; section 3.3 of [RFC3986]
     ; the name of an Entity Set in the EDM model

entityType          = *pchar   ; section 3.3 of [RFC3986]
     ; the name of an Entity Type in the EDM model

complexType          = *pchar   ; section 3.3 of [RFC3986]
     ; the name of an Complex Type in the EDM model

entityProperty      = *pchar   ; section 3.3 of [RFC3986]
     ; the name of a property (of type EDMSimpleType) on an
     ; Entity Type in the EDM
     ; model associated with the data service

entityComplexProperty   = *pchar   ; section 3.3 of [RFC3986]
     ; the name of a property (of type ComplexType) on an
     ; Entity Type in the EDM
     ; model associated with the data service

entityCollectionProperty   = *pchar   ; section 3.3 of [RFC3986]
     ; the entityCollectionProperty is supported only in OData 3.0
     ; the name of a property (of type Collection) on an
     ; Entity Type in the EDM
     ; model associated with the data service

entityNavProperty-es= *pchar   ; section 3.3 of [RFC3986]
     ; the name of a Navigation Property on an Entity Type in
     ; the EDM model associated with the data service.  The
     ; Navigation Property MUST identify an Entity Set.

entityNavProperty-et= *pchar   ; section 3.3 of [RFC3986]
     ; the name of a Navigation Property on an Entity Type
     ; in the EDM model associated with the data service.
     ; The Navigation Property MUST identify  an entity.

entityNamedStream   = *pchar       ; section 3.3 of [RFC3986]
     ; the entityNamedStream is supported only in OData 3.0
     ; the name of a Named Resource Stream on an Entity Type
     ; in the EDM model associated with the data service.

serviceOperation-collEt = *pchar   ; section 3.3 of [RFC3986]
     ; the name of a Function Import in the EDM model which returns a
     ; collection of entities from the same Entity Set

serviceOperation-et = *pchar       ; section 3.3 of [RFC3986]
     ; the name of a Function Import which returns a single Entity
     ; Type instance

serviceOperation-collCt = *pchar   ; section 3.3 of [RFC3986]
     ; the name of a Function Import which returns a collection of
     ; Complex Type [MC-CSDL] instances.  Each member of the
     ; collection is of the same type.

serviceOperation-ct = *pchar       ; section 3.3 of [RFC3986]
     ; the name of a Function Import which returns a single
     ; Complex Type [MC-CSDL] instance.

serviceOperation-collPrim = *pchar ; section 3.3 of [RFC3986]
     ; the name of a Function Import which returns a collection
     ; of primitive type (see section 2.2.2) values.  Each member
     ; of the collection is of the same type.
```

```
serviceOperation-prim = *pchar     ; section 3.3 of [RFC3986]
     ; the name of a Function Import which returns a single primitive
     ; type (see section 2.2.2) value.

serviceOperation-void = *pchar      ; section 3.3 of [RFC39876]
     ; the name of a Function Import that has no ReturnType.
```

Listing: Augmented BNF for URI Construction

2.2.3.2 Service Root (serviceRoot) and Path Prefix (pathPrefix)

The **serviceRoot** section of a data service URI represents the location of the root of a data service. The resource that is identified by this URI MUST be an AtomPub Service Document, as specified in [RFC5023] (or an alternate representation of Atom Service Document data if a different format is requested), that enumerates all of the collections of resources available for the data service.

Example valid URIs (as defined by the grammar in section 2.2.3.1), including only the URI scheme (`http://` in the examples below) and **serviceRoot** elements are:

```
http://host
http://::1:8080
http://api.constoso.com/v1/dataservice
```

This **pathPrefix** section of a data service URI is a data service defined sequence of URI path segments. This specification applies no further requirements to a **pathPrefix**.

Subsequent examples in this document use a URI scheme of `http://`. This is done to show a complete example. However, the URI-addressing rules defined in this document do not mandate that the `http://` scheme be used to address elements on an Entity Data Model (EDM).

2.2.3.3 Resource Path (resourcePath)

This section describes the construction rules for the resource path part of a data service URI. These rules dictate how the names of an EntitySet, EntityType, entity NavigationProperty, Member, named resource stream, and service operation can be composed to generate a URI that identifies a resource exposed by a data service.

Using the example Entity Data Model (EDM) in Appendix A: Sample Entity Data Model and CSDL Document (section 6), example URIs that include the scheme, **serviceRoot**, and **resourcePath** elements are:

```
http://host/service.svc/Customers
http://host/service.svc/Customers('ALFKI')/Orders
```

Resource Path: Semantics (section 2.2.3.5) describes the meaning of the various resource paths that can be constructed by using the rules noted in the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1). In addition, this section notes additional constraints specific to particular elements of the resource path.

2.2.3.4 Resource Path: Construction Rules

This section further defines grammar rules noted in the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1) which map directly to constructs defined in an Entity Data Model (EDM).

actionFQName: The name of an action (Actions (section 2.2.1.3)) that is defined as a **FunctionImport** in the EDM that is associated with the data service. It is possible to have multiple actions with the same name and same parameter types in different **EntityContainers**. To disambiguate between these actions, the name of the action MUST be prefixed with the name of the **entityContainer** that defines the action.

actionCall: A call to an action (section 2.2.1.3) that MUST be made with a POST request, and that MUST be the last segment of the resource path.

The first (or binding) parameter MAY be specified by the URI path to which the **actionCall** has been appended or it MAY be specified in the POST body (Action Parameters (section 2.2.3.6.5)) with all other parameters if the **actionCall** is made by directly appending the action segment to the URI that represents the data service.

In order to apply an **actionCall** to a URI path, the first (or binding) parameter of the corresponding **FunctionImport** MUST match (or be coercible to) the type that the URI path would return if retrieved directly.

functionFQName: The name of a function (section 2.2.1.4) defined as a **FunctionImport** in the EDM that is associated with the data service. It is possible to have multiple functions with the same name and parameter types in different **EntityContainers**. To disambiguate between these functions, the name of the function MAY be prefixed with the name of the **entityContainer** that defines the function.

functionCall: A call to a function (Functions (section 2.2.1.4)) by appending the function segment directly to the root of the service. Parameters can be provided in the query in the same way as parameters are provided to a **serviceOperation**, or parameters can be provided inside the **paren** of the **functionCall** in the URI path, or parameters can be provided via query parameters in the query part of the URI that are referenced by aliases that are declared inside the **paren** of the **functionCall**. If any parameters are specified inside the **paren** (either inline or via aliases), all parameters MUST be provided inline.

functionCall-partiallyBound: A call to a function (Functions (section 2.2.1.4)) where the first (or binding) parameter is specified by the URI path to which the **functionCall-partiallyBound** has been appended. Subsequent parameters can be provided in three ways: inside the **paren** of the **functionCall** in the URI path, via query parameters in the query part of the URI that are referenced by aliases declared inside the **paren** of the **functionCall** or, if this is the last **functionCall-partiallyBound** in the URI path, via query parameters with the same name as the parameters of the function as declared in the corresponding **FunctionImport**.

In order to apply a **functionCall-partiallyBound** to a URI path, the first (or binding) parameter of the corresponding **FunctionImport** MUST match (or be coercible to) the type that the URI path would return if it were retrieved directly.

entityContainer: The name of an EntityContainer in the EDM that is associated with the data service.

entitySet: The name of an EntitySet in the EDM that is associated with the data service. **EntitySet** names MAY<2> be directly followed by open and close parentheses (for example, Customers()). However, not appending the parenthesis is also valid and considered the canonical form of an **EntitySet** name.

If an **EntitySet** is not in the default EntityContainer, the URI MUST qualify the **EntitySet** name with the **EntityContainer** name as follows:

```
http://<Any iauthority [RFC3987] and optional URI path segments>/<Entity Container
name>.<Entity Set name>
```

entityType: The name of an EntityType in the EDM associated with the data service. The **EntityType** identified MAY be an **OpenEntityType** ([MC-CSDL] section 2.2.8).

namespaceQualifiedEntityType: The name of an **EntityType** in the EDM that is associated with the data service qualified with the namespace of the schema that is used in the EDM. The **EntityType** identified MAY be an **OpenEntityType**.

entityProperty: The name of a declared property or dynamic property, of type EDMSimpleType ([MC-CSDL] section 2.2.1) on an **EntityType** or a declared property of type **EDMSimpleType** defined on a ComplexType in the EDM that is associated with the data service.

If the prior URI path segment identifies an **EntityType** instance in **EntitySet** ES1, this value MUST be the name of a declared property or dynamic property, of type **EDMSimpleType**, on the base **EntityType** of set ES1.

If the prior URI path segment represents an instance of **ComplexType** CT1, this value MUST be the name of a declared property defined on **ComplexType** CT1.

entityComplexProperty: The name of a declared property, of type **ComplexType**, on an **EntityType** in the EDM associated with the data service.

If the prior URI path segment identifies an instance of an **EntityType** ET1, this value MUST be the name of a declared property or dynamic property on type ET1 which represents a **ComplexType** instance.

If the prior URI path segment identifies an instance of a **ComplexType** CT1, this value MUST be the name of a declared property on CT1 which represents a **ComplexType** instance.

entityCollectionProperty: The name of a declared property, of type **Collection**, on an **EntityType** in the EDM that is associated with the data service.

If the prior URI path segment identifies an instance of an **EntityType** ET1, the **entityCollectionProperty** MUST be the name of a declared property on type ET1 that represents a **Collection** instance.

There MUST NOT be any subsequent path segments in the URI after the **entityCollectionProperty**.

entityNavProperty: Identifies the name of a NavigationProperty on an **EntityType**.

If the prior URI path segment identifies an instance of an **EntityType** ET1, this value MUST be the name of a **NavigationProperty** on type ET1.

If the URI path segment preceding an **entityNavProperty** segment is "$links", there MUST NOT be any subsequent path segments in the URI after the **entityNavProperty**. If additional segments exist, the URI MUST be treated as invalid. For example, no path segment can follow the Orders segment in the URI:

```
http://host/service.svc/Customers('ALFKI')/$links/Orders.
```

entityNavProperty-es: This rule is the same as **entityNavProperty**, but with the added constraint that the **NavigationProperty** MUST point to an endpoint of an association with a cardinality of "many" (for example, such that traversing the association yields a set).

entityNavProperty-et: This rule is the same as **entityNavProperty**, but with the added constraint that the **NavigationProperty** MUST identify an **EntityType** instance.

entityNamedStream: Identifies the name of a property on an **EntityType** that is of type **Edm.Stream**.

The prior URI path segment MUST identify an instance of an **EntityType**.

If the **EntityType** identified by the prior segment is ET1, this value MUST be the name of a property of type **Edm.Stream** defined on either ET1 or on a base type of ET1.

keyPredicate: Specifies the property values of an **EntityKey**. When the **keyPredicate** is used in conjunction with an **entityNavProperty-es** that identifies a navigation property that, via the backing **AssociationType**, has a referential integrity constraint, then the key property values that are known to be shared between source and target entities MAY be omitted from the **keyPredicate**, but the renaming unknown key properties MUST be specified. In all other cases, all property values of the **EntityKey** MUST be specified.

keyPredicate-single: Identifies the **EntityKey** value of an **EntityType** whose **EntityKey** is comprised of only one property.

The EDM defines that each such key value is non-nullable, immutable, and an **EDMSimpleType**. The representation of an **EDMSimpleType** value in a data service URI has to follow the syntax rules defined in Abstract Type System (section 2.2.2).

An **EntityKey** consisting of a single **EntityType** property MAY<3> be represented by using the "<Entity Type property name> = <Entity Type property value>" syntax, as seen in the **keyPredicate-cmplx** grammar rule of the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1). However, the representation, which only specifies the value of the property, is the canonical representation for single property **EntityKeys**.

keyPredicate-cmplx: Identifies an EntityKey consisting of more than one property of the **EntityType**. The order in which the properties of a compound **EntityKey** appear in the URI MUST NOT be significant.

serviceOperation: Identifies a FunctionImport in an EDM, as seen in [MC-CSDL], which returns any of the following:

- Primitive type
- Collection of primitive types.
- single **ComplexType** instance.
- Collection of **ComplexType** instances.
- Single **EntityType** instance.
- Collection of **EntityType** instances.
- Nothing or void.

For additional details on the type system (primitive type, **ComplexType**, and so on) used by data services, see Message Syntax (section 2.2).

serviceOperation-collEt: Identifies a **FunctionImport** ([MC-CSDL] section 2.1.15) in an EDM, as specified in [MC-CSDL], that returns a collection of entities where each entity is in the same **EntitySet**. A service operation of this type acts as a pseudo **EntitySet** in that additional resource path (section 2.2.3.3) segments can follow that identify entities or relationships on entities within the collection that is identified by the service operation.

2.2.3.5 Resource Path: Semantics

This section describes the semantics for a base set of data service URIs. From these base cases, the semantics of longer URIs are defined by composing the rules below.

The URI segments used in this section use Augmented Backus-Naur Form (ABNF), as specified in [RFC5234] syntax, and the rules used in the segments are defined in the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1) and in [RFC3986]. Directly beneath each ABNF rule describing a URI there is a description of the semantic meaning for the URI and an example URI derived from the sample Entity Data Model (EDM) defined in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

The following rules are in addition to the grammar rules defined in the resource path semantics listing that appears later in this section:

- In each of the grammar rules below, the **serviceOperation-collEt** rule can be substituted for the first occurrence of an **entitySet** rule in the resource path. This type of a substitution redefines the replaced segment from identifying an EntitySet to identifying a group of entities.
- Any rule within the resource path portion of a data service URI, which identifies an **EntitySet** or collection of entities, MAY<4> be immediately followed by a parenthesis, as described by the "paren" rule in the ABNF grammar in URI Format: Resource Addressing Rules (section 2.2.3).
- `URI1 = scheme serviceRoot "/" entitySet`

  MUST identify all instances of the base EntityType or any of the **EntityType's** subtypes within the specified **EntitySet** specified in the last URI segment.

  If the EDM associated with the data service does not include an **EntitySet** with the name specified, this URI (and any URI created by appending additional path segments) MUST be treated as identifying a non-existent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

  Example:

  ```
  URI: http://host/service.svc/Customers
  Identifies: All customer entities in the Customers Entity Set
  ```

- `URI2 = scheme serviceRoot "/" entitySet "(" keyPredicate ")"`

  MUST identify a single **EntityType** instance, which is within the **EntitySet** specified in the URI, where key EntityKey is equal to the value of the **keyPredicate** specified.

  If no entity identified by the **keyPredicate** exists in the **EntitySet** specified, this URI (and any URI created by appending additional path segments) MUST represent a resource that does not exist in the data model.

  Example:

  ```
  URI: http://host/service.svc/Customers('ALFKI')
  Identifies: The entity in the Customers entity set with the Entity Key 'ALFKI'.
  ```

  Note The **keyPredicate** in this example represents an **EntityKey** made up of a single property (**keyPredicate-single**) and the type of that property is **Edm.String**. The literal value of the property is represented using single quotes as per the data literal syntax for primitive types, as specified in Abstract Type System (section 2.2.2).

- `URI3 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" entityComplexProperty`

  MUST identify an instance of a ComplexType on the specified **EntityType** instance. URI 2 (shown in the preceding example) describes how an **entitySet** followed by a **keyPredicate** identifies an **EntityType** instance.

Example:

```
URI: http://host/service.svc/Customers('ALFKI')/Address
Identifies: The value of the Address property of the customer entity identified by key
value 'ALFKI' in the Customers Entity Set.
```

- URI4 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" entityComplexProperty "/" entityProperty

  MUST identify a property of a **ComplexType** defined on the **EntityType** of the entity whose **EntityKey** value is specified by the **keyPredicate** and is within the specified **EntitySet**.

  As noted in the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1), a path segment containing only the rule entity property can append a '"/$value" segment. A $value MUST be interpreted as a dereference operator and indicates only the value of the property that is being addressed (for example, it does not indicate additional metadata or the surrounding envelope).

  Example:

  ```
  URI: http://host/service.svc/Customers('ALFKI')/Address/Name
  Identifies: The value of the Name property of the Address ComplexType property of the
  customer entity identified by key value 'ALFKI' in the Customers Entity Set.
  ```

  Example:

  ```
  URI: http://host/service.svc/Customers('ALFKI')/Address/Name/$value
  Identifies: Same as the example preceding, but identifies the value of the property free
  of any metadata or surrounding markup.
  ```

- URI5 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" entityProperty

  MUST identify a property whose type is an EDMSimpleType on the **EntityType** instance (identified with **EntityKey** equal to the specified key predicate) within the specified **EntitySet**.

  As noted in the Augmented BNF for URI Construction listing in URI Syntax (section 2.2.3.1), a path segment containing only the rule entity property can append a "/$value" segment. A $value MUST be interpreted as a dereference operator and indicates only the value of the property that is being addressed (for example, it indicates that no additional metadata or surrounding envelope is to be used).

  Example:

  ```
  URI: http://host/service.svc/Customers('ALFKI')/CompanyName
  Identifies: The name of the customer entity in the Customers EntitySet identified by key
  'ALFKI'.
  ```

  Example:

  ```
  URI: http://host/service.svc/Customers('ALFKI')/CompanyName/$value
  Identifies: Same as preceding, but identifies the value of the property free of any
  metadata or surrounding markup.
  ```

- URI6 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" entityNavProperty

MUST identify a set of entities or an **EntityType** instance that is reached via the specified **NavigationProperty** on the entity identified by the **EntitySet** name and key predicate specified.

For example, given an association between Customer and Order entities, an Order entity type might define a **NavigationProperty** named "OrderedBy" that represents the Customer instance associated with that particular Order instance. Similarly, the Customer entity type might define a navigation property named "Orders" that represents the Order instances associated to that particular Customer instance.

Example:

```
URI: http://host/service.svc/Customers('ALFKI')/Orders
Identifies: The set of Order Entity Type instances (or instances of a sub type of Order)
associated with the customer identified by the key 'ALFKI' through the Orders Navigation
Property.
```

- `URI7 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/$links/" entityNavProperty`

MUST identify the collection of all links from the specified **EntityType** instance (identified by the **EntitySet** name and key predicate specified) to all other entities that can be reached via the navigation property. The path segment following the $links segment specifies the specific association being addressed, which can identify a single or collection of links. Therefore, this URI identifies a link or collection of links (depending on the association multiplicity defined by the navigation property) and not the value of an entity or collection of entities.

Example:

```
URI: http://host/service.svc/Customers('ALFKI')/$links/Orders
Identifies: The collection of all Links between the entity in the Customers Entity Set
identified by key 'ALFKI' and the Orders entities associated with that customer via the
Orders navigation property.
```

Example:

```
URI: http://host/service.svc/Orders(1)/$links/Customer
Identifies: The Link between the order entity with key value 1 in the Orders Entity Set
and customer entity associated with that order via the Customer navigation property.
```

- `URI8 = scheme serviceRoot "/$metadata"`

MUST identify the Entity Data Model Extensions (EDMX) document, as specified in [MC-EDMX], which includes the EDM represented using a conceptual schema definition language (CSDL), as specified in [MC-CSDL], for the data service.

Example:

```
URI: http://host/service.svc/$metadata
Identifies: The EDMX (metadata) document for the data service
```

- `URI9 = scheme serviceRoot "/$batch"`

MUST identify the endpoint of a data service that accepts Batch Requests (section 2.2.7.6).

Example:

```
URI: http://host/service.svc/$batch
Identifies: The batch request endpoint for a data service
```

- `URI10 = scheme serviceRoot "/" serviceOperation-et`

  MUST identify a FunctionImport that returns a single **EntityType** instance.

  If no **FunctionImport** exists in the EDM associated with the data service which has the same name as specified by the **serviceOperation-et** rule, this URI MUST represent a resource that does not exist in the data model.

  As per the ABNF grammar in URI Format: Resource Addressing Rules (section 2.2.3), no further resource path segments can be composed onto a URI of this form.

- `URI11 = scheme serviceRoot "/" serviceOperation-collCt`

  MUST identify a **FunctionImport** that returns a collection of **ComplexType** instances.

  If no **FunctionImport** exists in the EDM associated with the data service that has the same name as specified by the **serviceOperation-collCt** rule, this URI MUST represent a resource that does not exist in the data model.

  As per the ABNF grammar in URI Format: Resource Addressing Rules (section 2.2.3), no further resource path segments can be composed onto a URI of this form.

- `URI12 = scheme serviceRoot "/" serviceOperation-ct`

  MUST identify a **FunctionImport** that returns a **ComplexType** instance.

  If no **FunctionImport** exists in the EDM associated with the data service that has the same name as specified by the **serviceOperation-ct** rule, this URI MUST represent a resource that does not exist in the data model.

  As per the ABNF grammar in URI Format: Resource Addressing Rules (section 2.2.3), no further resource path segments can be composed onto a URI of this form.

- `URI13 = scheme serviceRoot "/" serviceOperation-collPrim`

  MUST identify a **FunctionImport** that returns a collection of primitive type values. The set of primitive types supported is specified in URI Format: Resource Addressing Rules (section 2.2.3).

  If no **FunctionImport** exists in the EDM associated with the data service that has the same name as specified by the **serviceOperation-prim** rule, this URI MUST represent a resource that does not exist in the data model.

  As per the ABNF grammar in URI Format: Resource Addressing Rules (section 2.2.3), no further resource path segments can be composed on to a URI of this form.

- `URI14 = scheme serviceRoot "/" serviceOperation-prim`

  MUST identify a **FunctionImport** that returns a single primitive type value. The set of primitive types supported is defined in section 2.2.2.

  If no **FunctionImport** exists in the EDM associated with the data service that has the same name as specified by the **serviceOperation-prim** rule, this URI MUST represent a resource that does not exist in the data model.

  A path segment containing only the rule **serviceOperation-prim** can append a "/$value" segment. A $value MUST be interpreted as a dereference operator and indicates only the value of

the **property** that is being addressed (for example, it indicates no additional metadata or surrounding envelope is to be used).

- `URI15 = scheme serviceRoot "/" entitySet count`

  MUST identify the count of all instances of the base **EntityType** or any of the **EntityType's** subtypes within the specified **EntitySet** specified in the last URI segment.

  If the EDM associated with the data service does not include an **EntitySet** with the name specified, this URI MUST be treated as identifying a non-existent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

  The $count segment is supported only in the OData 2.0 and OData 3.0 protocols.

- `URI16 = scheme serviceRoot "/" entitySet "(" keyPredicate ") count`

  MAY identify the count of a single **EntityType** instance (the count value SHOULD always equal one), which is within the **EntitySet** specified in the URI, where key **EntityKey** is equal to the value of the **keyPredicate** specified.

  If the EDM associated with the data service does not include an **EntitySet** instance with the **keyPredicate** specified, this URI MUST be treated as identifying a nonexistent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

- `URI17 = scheme serviceRoot "/" entitySet "(" keyPredicate ")" value`

  MUST identify the Media Resource [RFC5023] associated with the identified **EntityType** instance. The **EntityType** that defines the entity identified MUST be annotated with the **HasStream** attribute, as defined in Conceptual Schema Definition Language Document for Data Services (section 2.2.3.7.2). As shown in the ABNF grammar in section 2.2.3.1, the "value" segment shown in this URI MAY be appended to any path which identifies a single entity.

  Example:

  ```
  URI: http://host/service.svc/Documents(1)/$value
  Identifies: The Media Resource associated with the Document Entity Type
  instance identified
  ```

- `URI18 = scheme serviceRoot "/" entitySet "(" keyPredicate ")"/" entityCollectionProperty`

  MUST identify a property whose type is a **Collection** on the **EntityType** instance (which is identified with **EntityKey** equal to the specified key predicate) within the specified **EntitySet**. As per the ABNF grammar in URI Format: Resource Addressing Rule (section 2.2.3), no further resource path segments can be composed onto a URI of this form.

  The **entityCollectionProperty** segment is supported only in the OData 3.0 protocol.

  Example:

  ```
  URI: http://host/service.svc/Customers(1)/AlternateAddresses
  Identifies: A Collection Property on the entity.
  ```

  The named resource stream segment is supported only in the OData 3.0 protocol.

- `URI19 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" entityNamedStream`

MUST identify a named resource stream that is associated with the identified **EntityType** instance. The **EntityType** that defines the entity that is identified MUST declare or inherit from a base type a property of type **Edm.Stream** with the same name.

Example:

```
URI: http://host/service.svc/Photos(1)/Thumbnail/
Identifies: The 'Thumbnail' Named Resource Stream associated with the specified Photo
```

The **namespaceQualifiedEntityType** segment is supported only in the OData 3.0 protocol.

- `URI20 = scheme serviceRoot "/" entitySet"/" namespaceQualifiedEntityType`

MUST identify all instances of the **EntityType**, subtype of the base **EntityType**, or any of its subtypes within the specified **EntitySet** specified in the prior URI segment.

If the EDM that is associated with the data service does not include an **EntityType** with the namespace-qualified **EntityType** specified, this URI (and any URI created by appending additional path segments) MUST be treated as identifying a nonexistent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

Example:

```
URI: http://host/service.svc/Customers/SampleModel.VipCustomer
Identifies: All customer entities in the Customers Entity Set that are of type
SampleModel.VipCustomer
```

- `URI21 = scheme serviceRoot "/" entitySet"/" namespaceQualifiedEnitityType "(" keyPredicate ")"`
- `URI22 = scheme serviceRoot "/" entitySet"(" keyPredicate ")" "/" namespaceQualifiedEnitityType`

Both of the previous URIs are equivalent and MUST identify a single namespace-qualified **EntityType** instance or one of its subtypes. In each case, the entity instance MUST be within the **EntitySet** that is specified in the URI, where key **EntityKey** is equal to the value of the specified **keyPredicate**.

If no entity identified by the **keyPredicate** exists in the specified **EntitySet**, this URI (and any URI created by appending additional path segments) MUST represent a resource that does not exist in the data model.

Examples:

```
URI: http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')
Identifies: The entity in the Customers entity set, of type SampleModel.VipCustomer, with
the Entity Key 'ALFKI2'.

URI: http://host/service.svc/Customers('ALFKI2')/SampleModel.VipCustomer
Identifies: The entity in the Customers entity set, of type SampleModel.VipCustomer, with
the Entity Key 'ALFKI2'.
```

The following path segments MAY be used following a URI in which **namespaceQualifiedEntityType** is used to specify a derived property:

- **entityComplexProperty**
- **entityProperty**

- **entityNavProperty**
- **entityCollectionProperty**
- **entityNamedStream**

Example:

```
URI:
http://host/service.svc/Customers('ALFKI2')/SampleModel.VipCustomer/CreditPurchases/Balanc
e
Identifies: The value of the Balance property of the CreditPurchases ComplexType property
of the VipCustomer entity identified by key value 'ALFKI2' in the Customers Entity Set.
```

Examples:

```
URI:
http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/CreditPurchases/Balanc
e/$value
Identifies: Same as the example preceding, but identifies the value of the property free
of any metadata or surrounding markup.

URI: http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/InHouseStaff
Identifies: The set of Employee Entity Type instances (or instances of a sub type of
Employee) associated with the VipCustomer identified by the key 'ALFKI2' through the
InHouseStaff Navigation Property.

URI:
http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/$links/InHouseStaff
Identifies: The collection of all Links between the VipCustomer entity in the Customers
Entity Set identified by key 'ALFKI2' and the Employee entities associated with that
VipCustomer via the InHouseStaff navigation property.

URI: http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/$value
Identifies: The Media Resource associated with the VipCustomer Entity Type
instance in the Customers Entity Set

URI:
http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/CountriesOfOperation
Identifies: A Collection Property on the VipCustomer entity.

URI: http://host/service.svc/Customers/SampleModel.VipCustomer('ALFKI2')/Logo
Identifies: The 'Logo' Named Resource Stream associated with the specified property of the
VipCustomer entity identified by key value 'ALFKI2' in the Customers Entity Set.
```

The **functionCall** segment is supported only in the OData 3.0 protocol.

- `URI23 = scheme serviceRoot "/" functionCall`

The **functionCall** segment MUST identify a **FunctionImport** that represents a function (Functions (section 2.2.1.4)). If the EDM that is associated with the data service does not include a **FunctionImport** with **IsSideEffecting** set to "false" and the name specified, this URI MUST be treated as identifying a nonexistent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

Further resource path segments or query options MAY be specified only if the **FunctionImport** does not have **IsComposable** set to "false".

Examples:

```
URI: http://host/service.svc/TopTenCustomersInCity(city='Seattle')
```

```
Identifies: Results of evaluating the TopTenCustomersInCity function with a parameter
value of 'Seattle' specified inline.

URI: http://host/service.svc/TopTenCustomersInCity(city=@c)?@c='Seattle'
Identifies: Results of evaluating the TopTenCustomersInCity function with a parameter
value of 'Seattle' specified via a parameter alias.

URI: http://host/service.svc/TopTenCustomersInCity()?city='Seattle'
Identifies: Results of evaluating the TopTenCustomersInCity function with a parameter
value of 'Seattle' specified by parameter name in the query.
```

The **functionCall-partiallyBound** segment is supported only in the OData 3.0 protocol.

- `URI24 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" functionCall-partiallyBound`

  The **functionCall-partiallyBound** segment MUST identify a **FunctionImport** that represents a function (Functions (section 2.2.1.4)). If the EDM that is associated with the data service does not include a **FunctionImport** (that has no side effects) with the name specified, this URI MUST be treated as identifying a nonexistent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

  The **FunctionImport** MUST have **IsBindable** set to "true", and MUST have at least one parameter. The first (binding) parameter MUST be of the same type as is represented by the resource path to which the **functionCall-partiallyBound** segment is appended.

  Further resource path segments or query options MAY be specified only if the **FunctionImport** is composable.

  Example:

```
URI: http://host/service.svc/Customers('ALFKI')/TopTenOrders
Identifies: Results of evaluating the TopTenOrders function with the first (or binding)
parameter value of 'http://host/service.svc/Customers('ALFKI')'
```

  This **functionCall-partiallyBound** segment MAY also be applied to **entitySet** segments directly.

- `URI25 = scheme serviceRoot "/" entitySet "/" functionCall-partiallyBound`

  The **functionCall-partiallyBound** segment MUST identify a **FunctionImport** that represents a function (Functions), that has a least one (the binding) parameter of type **Collection(entityType)** where **entitySet** contains **entityType** instances.

  Other rules are the same as for URI24.

  Example:

```
URI: http://host/service.svc/Customers/TopTenCustomers
Identifies: Results of evaluating the TopTenCustomers function with the first (or binding)
parameter value of 'http://host/service.svc/Customers'.
```

  Notice that **functionCall-partiallyBound**, **actionCall**, and **namespaceQualifiedEntityType** segments are the only resource path segments that can be directly appended to a resource path that represents a collection of entities without first selecting a single entity using a **keyPredicate**. A **functionCall—partiallyBound** segment is also the only segment that can be appended to a resource path that represents a collection of primitive types or **ComplexTypes**.

  This **functionCall-partiallyBound** segment MAY also be applied to arbitrary **navPaths**.

- `URI26 = scheme serviceRoot "/" entitySet navPath "/" functionCall-partiallyBound`

  Examples:

  ```
  URI: http://host/service.svc/SalesPeople(6)/Customers/TopTenCustomers()
  Identifies: Results of evaluating the TopTenCustomers function with the first (or binding)
  parameter value of 'http://host/service.svc/SalesPeople(6)/Customers', i.e. the Customers
  of SalesPerson(6).

  URI: http://host/service.svc/SalesPeople(6)/Customers/Best()/TopTenOrders()
  Identifies: Results of evaluating the TopTenOrders function with the first (or binding)
  parameter value being the result of evaluating the Best function which returns a single
  customer, and which has been evaluated with its first (or binding) parameter value set to
  'http://host/service.svc/SalesPeople(6)/Customers', i.e. the Customers of SalesPerson(6)
  ```

  Notice that multiple **functionCall-partiallyBound** segments can be used in a single resource path, as in the previous second example.

  The **actionCall** segment is supported only in the OData 3.0 protocol.

- `URI27 = scheme serviceRoot "/" entitySet "(" keyPredicate ")/" actionCall`

  The **actionCall** segment MUST identify a **FunctionImport** that represents an action (Actions (section 2.2.1.3)). If the EDM associated with the data service does not include a **FunctionImport** with the name specified, this URI MUST be treated as identifying a non-existent resource, as described in Message Processing Events and Sequencing Rules (section 3.2.5).

  The **FunctionImport** MUST have **IsBindable** set to "true", and MUST have at least one parameter. The first (binding) parameter MUST be of the same type as is represented by the resource path to which the **actionCall** segment is appended.

  Further resource path segments or query options MAY be specified only if the **FunctionImport** is composable.

  ```
  To invoke an action a POST request MUST be used. All parameters other than the first
  (binding) parameter MUST be specified in the body of the POST request used to invoke this
  action (see section 2.2.3.6.5).
  ```

- `URI28 = scheme serviceRoot "/" actionCall`

  The **actionCall** segment MAY be called directly off the service root if it has binding set to "false" or if the binding parameter is specified in the POST body (see Action Parameters (section 2.2.3.6.5)).

Listing: Resource Path Semantics

2.2.3.6 Query Options

As described in section 2.2.3, all data services MUST follow the query string parsing and construction rules as defined in this section and its subsections.

The query options section of a data service URI specifies three types of information: system query options (2.2.3.6.1), custom query options (2.2.3.6.2), and service operation parameters (2.2.3.6.3). System query options and service operation parameters MUST conform to the following rules:

- Any number of the query options MAY<5> be specified in a data service URI.
- The order of query options within a URI MUST be insignificant.

- Query option names and values MUST be treated as case sensitive.
- System query option names MUST begin with a "$", as seen in System Query Options (section 2.2.3.6.1).
- Custom Query Options (section 2.2.3.6.2) MUST NOT begin with a "$".

2.2.3.6.1 System Query Options

System query options in a data service URI, defined in URI Format: Resource Addressing Rules (section 2.2.3) are directives that are defined by this document that a client MAY specify to control the amount and order of the data that a data service returns for the resource identified by the URI. The names of all system query options are prefixed with a "$" character.

A data service MAY<6> support some or all of the system query options defined in this document. If a data service does not support a system query option, it MUST reject any requests which contain the unsupported option, as seen in Message Processing Events and Sequencing Rules (section 3.2.5) for HTTP-specific server details.

The following table summarizes the system query options defined in this document.

If a system query option is included in a data service URI identifying a resource that is incompatible with the query option, as shown in the following system query options supported per URI table, the URI MUST be considered malformed.

System query option	Description	Additional details
$expand	This option indicates entities that are associated with the EntityType instance or EntitySet, identified by the resource path section of the URI, and represented inline in the data service's response, as opposed to being represented with Deferred Content markers in Deferred Content (section 2.2.6.2.6) and Deferred Content (section 2.2.6.3.9).	See Expand System Query Option ($expand) (section 2.2.3.6.1.3).
$filter	This option specifies a predicate used to filter the elements from the **EntitySet** identified by the resource path section of the URI.	See Filter System Query Option ($filter) (section 2.2.3.6.1.4).
$orderby	This option specifies the sort properties and sort direction (ascending or descending) that the data service is to use to order the entities in the **EntitySet**, identified by the resource path section of the URI.	See OrderBy System Query Option ($orderby) (section 2.2.3.6.1.6).
$format	This option specifies the media type that is acceptable in a response. If present, this value takes precedence over the value(s) specified in an Accept (section 2.2.5.1) request header.	See Format System Query Option ($format) (section 2.2.3.6.1.5).
$skip	This option specifies a positive integer N that represents the number of entities, counted from the first entity in the **EntitySet** and ordered as specified by the **$orderby** option, that the data service is to skip when returning the entities in the **EntitySet**, which is identified by the resource path section of the URI. The data service is to return all subsequent entities, starting from the one in position N+1.	See Skip System Query Option ($skip) (section 2.2.3.6.1.7).
$top	This option specifies a positive integer N that is the maximum number of entities in the **EntitySet**, identified by the resource path section of the URI, that	See Top System Query Option ($top) (section 2.2.3.6.1.8).

System query option	Description	Additional details
	the data service is to return.	
$skiptoken	The value of a **$skiptoken** query option is an opaque token which identifies an index into the collection of entities identified by the URI containing the **$skiptoken** parameter.	See Skip Token Query Option ($skiptoken) (section 2.2.3.6.1.9)
$inlinecount	For a value of "allpages", this option indicates that the response to the request has to include the count of the number of entities in the **EntitySet**, identified by the resource path section of the URI, after all **$filter** system query options have been applied. For a value of "none", this option indicates that the response to the request is not to include the count value.	See InlineCount System Query Option ($inlinecount) (section 2.2.3.6.1.10)
$select	This option is used to specify that a subset of the properties of the entities that are identified by the path of the request URI and **$expand** query option are to be returned in the response from the data service.	See Select System Query Option (section 2.2.3.6.1.11)

Table: Summary of Supported System Query Options

In the following table, the row labels (URI1, URI2, and so on) refer to the resource path semantics table in URI types defined by the grammar rules in Resource Path: Semantics (section 2.2.3.5). A cell value of "Yes" indicates the system query option MAY be used with the URI type associated with the row. A blank cell indicates that if the system query option is present on a URI of the form indicated by the associated row, the URI is to be considered malformed.

	$expand	$filter	$format	$orderby	$skip	$top	$skiptoken	$inlinecount (Note 3)	$select (Note 3)
URI1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI2	Yes		Yes						Yes
URI3		Yes	Yes						
URI4			Yes						
URI5			Yes						
URI6 (Note 1)	Yes	Yes	Yes						Yes
URI6 (Note 2)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI7			Yes		Yes	Yes	Yes	Yes	
URI8									
URI9									
URI10			Yes						
URI11			Yes						

	$expand	$filter	$format	$orderby	$skip	$top	$skiptoken	$inlinecount (Note 3)	$select (Note 3)
URI12			Yes						
URI13			Yes						
URI14			Yes						
URI15	Yes	Yes		Yes	Yes				
URI16	Yes	Yes							
URI17			Yes						
URI18									Yes
URI19			Yes						
URI20	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI21	Yes		Yes						Yes
URI22	Yes		Yes						Yes
URI23	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI24	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI25	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI26	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
URI27			Yes						
URI28			Yes						

Table: System Query Options Supported Per URI

Note 1: Applies when the NavigationProperty in the final path segment of the URI identifies a single **EntityType** instance.

Note 2: Applies when the **NavigationProperty** in the final path segment of the URI identifies a set of entities.

Note 3: The **$inlinecount** and **$select** system query options are supported in the OData 2.0 and OData 3.0 protocols.

2.2.3.6.1.1 Common Expression Syntax

The Filter and OrderBy query options are specified in the data service URI via the common expression syntax defined in following Augmented BNF for query option expressions listing.

```
commonExpression = [WSP] (boolCommonExpression / methodCallExpression /
             parenExpression / literalExpression / addExpression /
             subExpression / mulExpression / divExpression /
             modExpression /  negateExpression / memberExpression
             / firstMemberExpression / castExpression / functionCallExpression ) [WSP]

boolCommonExpression = [WSP] (boolLiteralExpression / andExpression /
             orExpression /
             boolPrimitiveMemberExpression / eqExpression / neExpression /
```

```
                ltExpression / leExpression / gtExpression /
                geExpression / notExpression / isofExpression/
                boolCastExpression / boolMethodCallExpression  /
                firstBoolPrimitiveMemberExpression / boolParenExpression /
                boolFunctionCallExpression) [WSP]

    parenExpression = "(" [WSP] commonExpression [WSP] ")"

    boolParenExpression = "(" [WSP] boolCommonExpression [WSP] ")"

    andExpression = boolCommonExpression WSP "and" WSP boolCommonExpression

    orExpression = boolCommonExpression WSP "or" WSP boolCommonExpression

    eqExpression = commonExpression WSP "eq" WSP commonExpression

    neExpression = commonExpression WSP "ne" WSP commonExpression

    ltExpression = commonExpression WSP "lt" WSP commonExpression

    leExpression = commonExpression WSP "le" WSP commonExpression

    gtExpression = commonExpression WSP "gt" WSP commonExpression

    geExpression = commonExpression WSP "ge" WSP commonExpression

    addExpression = commonExpression WSP "add" WSP commonExpression

    subExpression = commonExpression WSP "sub" WSP commonExpression

    mulExpression = commonExpression WSP "mul" WSP commonExpression

    divExpression = commonExpression WSP "div" WSP commonExpression

    modExpression = commonExpression WSP "mod" WSP commonExpression

    negateExpression = "-" [WSP] commonExpression

    notExpression = "not" WSP commonExpression

    isofExpression = "isof" [WSP] "("[[WSP] commonExpression [WSP] ","][WSP]
                     stringUriLiteral [WSP] ")"

    castExpression = "cast" [WSP] "("[[WSP] commonExpression [WSP] ","][WSP]
                     stringUriLiteral [WSP] ")"

    boolCastExpression = "cast" [WSP]
                         "("[[WSP] commonExpression [WSP] ","][WSP]
                         "Edm.Boolean" [WSP] ")"

    firstMemberExpression = [WSP] [namespaceQualifiedEnitityType "/"]
                            [lambdaPredicatePrefixExpression]
                            ; A lambdaPredicatePrefixExpression is only defined inside a
                            ; lambdaPredicateExpression. A lambdaPredicateExpression is required
                            ; inside a lambdaPredicateExpression.
                            entityNavProperty /
                            ; section 2.2.3.1
                            entityComplexProperty /
                            ; section 2.2.3.1
                            entityProperty /
                            ; section 2.2.3.1
                            entityCollectionProperty
                            ; section 2.2.3.1

    firstBoolPrimitiveMemberExpression = [namespaceQualifiedEntityType "/"]entityProperty
                                       ; section 2.2.3.1

    memberExpression = commonExpression [WSP]  "/" [WSP] [namespaceQualifiedEntityType "/"]
                       entityNavProperty /      ; section 2.2.3.1
                       entityComplexProperty /      ; section 2.2.3.1
```

```
                    entityProperty /    ; section 2.2.3.1
                    entityCollectionProperty  ; section 2.2.3.1

boolPrimitiveMemberExpression = commonExpression [WSP]  "/" [WSP]
                                [namespaceQualifiedEntityType "/"]entityProperty
                                ; section 2.2.3.1

literalExpression = stringUriLiteral ; section 2.2.2
          / dateTimeUriLiteral ; section 2.2.2
          / dateTimeOffsetUriLiteral ; section 2.2.2
          / timeUriLiteral ; section 2.2.2
          / decimalLiteral ; section 2.2.2
          / guidUriLiteral ; section 2.2.2
          / singleLiteral ; section 2.2.2
          / doubleLiteral ; section 2.2.2
          / int16Literal ; section 2.2.2
          / int32Literal ; section 2.2.2
          / int64Literal ; section 2.2.2
          / binaryLiteral ; section 2.2.2
          / nullLiteral ; section 2.2.2
          / byteLiteral ; section 2.2.2
          / fullPointLiteral ; section 2.2.2
          / fullLineStringLiteral ; section 2.2.2
          / fullPolygonLiteral ; section 2.2.2
          / fullGeoCollectionLiteral ; section 2.2.2
          / fullMultiPointLiteral ; section 2.2.2
          / fullMultiLineStringLiteral ; section 2.2.2
          / fullMultiPolygonLiteral ; section 2.2.2

boolLiteralExpression = boolLiteral ; section 2.2.2

methodCallExpression = boolMethodCallExpression
                       / indexOfMethodCallExpression
                       / replaceMethodCallExpression
                       / toLowerMethodCallExpression
                       / toUpperMethodCallExpression
                       / trimMethodCallExpression
                       / substringMethodCallExpression
                       / concatMethodCallExpression
                       / lengthMethodCallExpression
                       / yearMethodCallExpression
                       / monthMethodCallExpression
                       / dayMethodCallExpression
                       / hourMethodCallExpression
                       / minuteMethodCallExpression
                       / secondMethodCallExpression
                       / roundMethodCallExpression
                       / floorMethodCallExpression
                       / ceilingMethodCallExpression
                       / distanceMethodCallExpression
                       / geoLengthMethodCallExpression

boolMethodCallExpression = endsWithMethodCallExpression
                       / startsWithMethodCallExpression
                       / substringOfMethodCallExpression
                       / intersectsMethodCallExpression
                       / anyMethodCallExpression
                       / allMethodCallExpression

endsWithMethodCallExpression = "endswith" [WSP]
                               "(" [WSP] commonexpression [WSP]
                               "," [WSP] commonexpression  [WSP] ")"

indexOfMethodCallExpression = "indexof" [WSP]
                              "(" [WSP] commonexpression [WSP]
                              "," [WSP] commonexpression [WSP] ")"

replaceMethodCallExpression =  "replace" [WSP]
                               "(" [WSP] commonexpression [WSP]
                               "," [WSP] commonexpression [WSP]
```

```
                                ","  [WSP] commonexpression [WSP] ")"

    startsWithMethodCallExpression = "startswith" [WSP]
                                     "(" [WSP] commonexpression [WSP]
                                     "," [WSP] commonexpression  [WSP] ")"

    toLowerMethodCallExpression = "tolower" [WSP]
                                  "(" [WSP] commonexpression [WSP] ")"

    toUpperMethodCallExpression = "toupper" [WSP]
                                  "(" [WSP] commonexpression [WSP] ")"

    trimMethodCallExpression = "trim" [WSP]
                               "(" [WSP] commonexpression [WSP] ")"

    substringMethodCallExpression = "substring" [WSP]
                                    "(" [WSP] commonexpression [WSP]
                                    "," [WSP] commonexpression [WSP]
                                    [ "," [WSP] commonexpression  [WSP] ] ")"

    substringOfMethodCallExpression = "substringof" [WSP]
                                      "(" [WSP] commonexpression [WSP]
                                      [ "," [WSP] commonexpression [WSP] ] ")"

    concatMethodCallExpression = "concat" [WSP]
                                 "(" [WSP] commonexpression [WSP]
                                 [ "," [WSP] commonexpression [WSP] ] ")"

    lengthMethodCallExpression = "length" [WSP]
                                 "(" [WSP] commonexpression [WSP] ")"

    getTotalOffsetMinutesMethodCallExpression = "gettotaloffsetminutes" [WSP]
                                 "(" [WSP] commonexpression [WSP] ")"

    yearMethodCallExpression = "year" [WSP]
                               "(" [WSP] commonexpression [WSP] ")"

    monthMethodCallExpression = "month" [WSP]
                                "(" [WSP] commonexpression [WSP] ")"

    dayMethodCallExpression = "day" [WSP]
                              "(" [WSP] commonexpression [WSP] ")"

    hourMethodCallExpression = "hour" [WSP]
                               "(" [WSP] commonexpression [WSP] ")"

    minuteMethodCallExpression = "minute" [WSP]
                                 "(" [WSP] commonexpression [WSP] ")"

    secondMethodCallExpression = "second" [WSP]
                                 "(" [WSP] commonexpression [WSP] ")"

    roundMethodCallExpression = "round" [WSP]
                                "(" [WSP] commonexpression [WSP] ")"

    floorMethodCallExpression = "floor" [WSP]
                                "(" [WSP] commonexpression [WSP] ")"

    ceilingMethodCallExpression = "ceiling" [WSP]
                                  "(" [WSP] commonexpression [WSP] ")"

    distanceMethodCallExpression = "geo.distance" [WSP]
                                   "(" [WSP] commonexpression [WSP]
                                   "," [WSP] commonexpression  [WSP] ")"

    geoLengthMethodCallExpression = "geo.length" [WSP]
                                    "(" [WSP] commonexpression [WSP] ")"

    intersectsMethodCallExpression = "geo.intersects" [WSP]
                                     "(" [WSP] commonexpression [WSP]
```

```
                                ","  [WSP] commonexpression  [WSP] ")"

implicitVariableExpression = "$it"
    ; references the unnamed outer variable of the query

lambdaVariableExpression     = *pchar
    ; section 3.3 of [RFC3986]
    ; a identifier/name that complies with EDM identifier rules

lambdaPredicatePrefixExpression = inscopeVariableExpression  "/"

lambdaPredicateExpression = boolCommonExpression
    ; this is a boolCommonExpression with the added restriction that any
    ; firstMemberExpression expressions that are inside the methodPredicateExpression
    ; MUST have a prefix of lambdaPredicatePrefixExpression.

inscopeVariableExpression = implicitVariableExpression | lambdaVariableExpression
    ; the lambdaVariableExpression must be the name of a variable introduced by either the
    ; current lambdaMethodCallExpression's lambdaVariableExpression or via a wrapping
    ; lambdaMethodCallExpression's lambdaVariableExpression.

lambdaMethodCallExpression = anyMethodCallExpression | allMethodCallExpression.

anyMethodCallExpression = pathExpression-collection "/"
                          "any"
                          "("
                          [ lambdaVariableExpression ":" lambdaPredicateExpression ]
                            ")"

allMethodCallExpression = pathExpression-collection "/"
                          "all"
                          "("
                          lambdaVariableExpression ":" lambdaPredicateExpression
                            ")"

singlePathExpression = [WSP]
                       "("singlePathExpression / inscopeVariableExpression
                       "/" entityNavProperty-et | entityComplexProperty

collectionPathExpression = [WSP] commonexpression [WSP]
                           singlePathExpression / inscopeVariableExpression
                           "/"
                           (entityNavProperty-es | entityCollectionProperty)

functionCallExpression = [ ( memberExpression / firstMemberExpression ) "/"]
                         functionFQName ; section 2.2.3.1
                         "(" [functionParametersExpression] ")"

boolFunctionCallExpression = functionCallExpression
    ; with the added restriction that the specified FunctionImport
    ; has a ReturnType of Edm.Boolean

functionParametersExpression =
                       functionParameterExpression *( "," functionParameterExpression)

functionParameterExpression = [WSP]
                              functionParameterName ; section 2.2.3.1
                              [WSP] "="
                              [WSP]
                              literalExpression / structuralValue / entityReference
                              [WSP]

structuralValue = ; a JSON or Verbose JSON encoding of a complex type, multi-value,
    ; entity, or collection of entities

entityReference =      "KEY("
                       [ entityContainer "." ]
                       entitySet
                       "("keyPredicate")"
                       ")"
```

```
                    ["/" namespaceQualifiedEntityType ]
                    ; refers a single Entity by key, and optionally allows a cast to a
                    ; derived type.
```

Listing: Augmented BNF for Query Option Expressions

A data service MAY<7> support some or all of the **boolCommonExpression** expressions for the Filter **(**$**filter)** system query option. A data service MAY<8> support some or all of the **commonExpression** expressions for the OrderBy **(**$**orderby)** query option.

If a data service does not support a given expression, it MUST reject any requests which contain the unsupported expression.

A data service MAY reject any requests that contain expressions not defined in this document.

Common expressions SHOULD be constructed and evaluated according to the rules defined in common expression syntax for each specific expression type.

2.2.3.6.1.1.1 Expression Construction and Evaluation Rules

commonExpression: A data service MAY support the **commonExpression** common expression. If supported, a **commonExpression** MUST represent any and all supported common expression types.

boolCommonExpression: A data service MAY support the **boolCommonExpression** common expression. If supported, a **boolCommonExpression** MUST be a common expression that evaluates to the Entity Data Model (EDM) primitive type **Edm.Boolean**.

parenExpression: A data service MAY support the enclosing of expressions in parentheses. This expression is represented as a **parenExpression** common expression in the common expression syntax.

If supported, a **parenExpression** MUST be evaluated by evaluating the expression with the parentheses, starting with the innermost parenthesized expressions, and proceeding outwards, following proper precedence rules where parentheses override any other operator precedence. The result of the **parenExpression** MUST be the result of the evaluation of the contained expression.

boolParenExpression: A data service MAY support the enclosing of Boolean expressions in parentheses. This expression is represented as a **boolParenExpression** common expression in the common expression syntax.

If supported, a **boolParenExpression** MUST be evaluated by evaluating the expression with the parentheses. The result of the **boolParenExpression** MUST be the result of the evaluation of the contained expression and MUST be of the EDM primitive type **Edm.Boolean**.

addExpression: A data service MAY support the binary addition operator. The operation of adding two expressions is represented as an **addExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<9> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**

- **Edm.Int64**

The **addExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **addExpression** MUST be the same type as the operands after binary numeric promotion rules have been applied to operands.

If supported, the data service SHOULD evaluate the operation represented by the **addExpression**, according to the rules of [IEEE754-2008] for the addition operation. Further, the data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

subExpression: A data service MAY support the binary subtraction operator. The operation of subtracting two expressions is represented as a subExpression common expression in the common expression syntax. If this operation is supported, the data service MAY<10> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**

The **subExpression** SHOULD NOT be supported for operands of any other EDM primitive type.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **subExpression** MUST be the same type as the operands after binary numeric promotion rules have been applied to operands.

If supported, the data service SHOULD evaluate the operation represented by the **subExpression**, according to the rules of [IEEE754-2008] for the subtraction operation. Further, the data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

mulExpression: A data service MAY support the binary multiplication operator. The operation of multiplying two expressions is represented as a **mulExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<11> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**

- **Edm.Int64**

The **mulExpression** SHOULD NOT be supported for operands of any other EDM primitive type.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **mulExpression** MUST be the same type as the operands after binary numeric promotion rules have been applied to operands.

If supported, the data service SHOULD evaluate the operation represented by the **mulExpression**, according to the rules of [IEEE754-2008] for the multiplication operation. Further, the data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

divExpression: A data service MAY support the binary division operator. The operation of dividing two expressions is represented as a **divExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<12> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**

The **divExpression** SHOULD NOT be supported for operands of any other EDM primitive type.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **divExpression** MUST be the same type as the operands after binary numeric promotion rules have been applied to operands.

If supported, the data service SHOULD evaluate the operation represented by the **divExpression**, according to the rules of [IEEE754-2008] for the division operation. Further, the data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

modExpression: A data service MAY support the binary remainder operator. The operation of computing the remainder of two expressions is represented as a **modExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<13> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**

- **Edm.Int64**

The **modExpression** SHOULD NOT be supported for operands of any other EDM primitive type.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **modExpression** MUST be the same type as the operands after binary numeric promotion rules have been applied to operands.

If supported, the data service SHOULD evaluate the operation represented by the **modExpression**, according to the rules of [IEEE754-2008] for the remainder operation. Further, the data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

negateExpression: A data service MAY support the unary negate operator. The operation of negating an expression is represented by the **negateExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<14> support some or all of the common expressions as operands of the operation. The operand expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**

The data service SHOULD NOT support operand expressions of any other EDM primitive type for the **negateExpression**.

If supported, a data service SHOULD follow the unary numeric promotion rules defined in Unary Numeric Promotions (section 2.2.3.6.1.1.3) to implicitly convert the operand to a supported EDM primitive type. The EDM primitive type of the result of evaluating the **negateExpression** MUST be the same type as the operand after binary numeric promotion rules have been applied to the operand.

If supported, the data service SHOULD evaluate the operation represented by the **negateExpression** by subtracting the operand value from zero. This result of evaluating the **negateExpression** SHOULD always be equal to the result of evaluating the **subExpression** where one operand is the value zero and the other is the value of the operand.

The data service MAY support evaluating an operand with a null value following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

andExpression: A data service MAY support the binary logical-and operator. The operation of evaluating whether two expressions both evaluate to the value of true is represented by the **andExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<15> support some or all of the **boolCommonExpression** expressions as operands of the operation. Those operand expressions MUST evaluate to the EDM primitive types of **Edm.Boolean**. The **andExpression** SHOULD NOT be supported for operands of any other EDM primitive types.

The EDM primitive type of the result of evaluating the **andExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST evaluate the expression to the value of true if the values of the operands are both true after being evaluated. If either operand is false after being evaluated, the expression MUST evaluate to the value of false.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

orExpression: A data service MAY support the binary logical-or operator. The operation of evaluating whether at least one of two expressions evaluate to the value of true is represented by the **orExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<16> support some or all of the **boolCommonExpression** common expressions as operands of the operation. Those operand expressions MUST evaluate to the EDM primitive types of **Edm.Boolean**. The **orExpression** SHOULD NOT be supported for operands of any other EDM primitive types.

The EDM primitive type of the result of evaluating the **orExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST evaluate the expression to the value of true if at least one of the operands is true after being evaluated. If both operands are false after being evaluated, the expression MUST evaluate to the value of false.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

memberExpression: A data service MAY support the referencing of a navigation, complex, or simple property of an EntityType. This is represented by the **memberExpression** common expression in the common expression syntax.

If supported, the common expression that is the target of the **memberExpression** MUST be a known **EntityType** or **ComplexType**. If supported, the **memberExpression** MAY reference an entity NavigationProperty (**entityNavProperty**, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an entity complex type property (**entityComplexProperty**, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an entity simple property, as specified in Resource Path: Construction Rules (section 2.2.3.4). For entity **NavigationProperties**, the target relationship end must have a cardinality of 1 (single entity, mandatory) or 0..1 (single entity, optional).

The type of the result of evaluating the **memberExpression** MUST be the same type as the property reference in the **memberExpression**.

The data service MAY support evaluating a **memberExpression** where instance values of a property are null following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

firstMemberExpression: A data service MAY support the referencing of a navigation, complex, or simple property of the **EntityType** or ComplexType represented by the last segment in the navigation portion of the URI. This is represented by the **firstMemberExpression** common expression in the common expression syntax.

If supported, the **firstMemberExpression** MAY reference an entity navigation property (**entityNavProperty**, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an entity complex type property (**entityComplexProperty**, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an entity simple property, as specified in Resource Path: Construction Rules (section 2.2.3.4). For entity **NavigationProperties**, the target relationship end has to have a cardinality of 1 (single entity, mandatory) or 0..1 (single entity, optional).

The type of the result of evaluating the **firstMemberExpression** MUST be the same type as the property reference in the **firstMemberExpression**.

The data service MAY support evaluating a **firstMemberExpression** where instance values of a property are null following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

When nested inside **lambdaPredicateExpression**, **firstMemberExpression** MUST be prefixed with a **lambdaPredicatePrefixExpression** that identifies to what the **firstMemberExpression** is bound.

boolPrimitiveMemberExpression: A data service MAY support the referencing of a Boolean simple property of an **EntityType** or **ComplexType**. This is represented by the **boolPrimitiveMemberExpression** common expression in the common expression syntax.

The type of the result of evaluating the **boolPrimitiveMemberExpression** MUST be EDM primitive type **Edm.Boolean**.

The data service MAY support evaluating a Boolean **memberExpression** where the property instance value is null following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

firstBoolPrimitiveMemberExpression: A data service MAY support the referencing of a Boolean simple property of an **EntityType** or **ComplexType** represented by the last segment in the navigation portion of the URI. This is represented by the **firstBoolPrimitiveMemberExpression** common expression in the common expression syntax.

The type of the result of evaluating the **boolPrimitiveMemberExpression** MUST be EDM primitive type **Edm.Boolean**.

The data service MAY support evaluating a Boolean **memberExpression** where the property instance value is null following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

eqExpression: A data service MAY support the binary equality operator. The operation of evaluating whether two expressions are equal is represented as an **eqExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<17> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of a known **EntityType** (see [MC-CSDL] for the definition of equality between two **EntityType** instances) or one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Boolean**
- **Edm.DateTimeOffset**
- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**
- **Edm.DateTime**

- **Edm.Guid**
- **Edm.Binary**

The **eqExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **eqExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the values of the operands are equal and false if they are not equal. If the type of the operands is a known **EntityType**, then a value of true MUST be returned if the operand expressions, once evaluated, represent the same entity instance. Actual comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

neExpression: A data service MAY support the binary non-equality operator. The operation of evaluating whether two expressions are not equal is represented as an **neExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<18> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of a known **EntityType** or one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Boolean**
- **Edm.DateTimeOffset**
- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**
- **Edm.DateTime**
- **Edm.Guid**
- **Edm.Binary**

The **neExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common

supported EDM primitive type. The EDM primitive type of the result of evaluating the **neExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the values of the operands are not equal and false if they are equal. If the type of the operands is a known **EntityType**, then a value of true MUST be returned if the operand expressions once evaluated do not represent the same entity instance. Actual comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

ltExpression: A data service MAY support the binary less than operator. The operation of evaluating whether one expression is less than the other expression is represented as an **ltExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<19> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.DateTimeOffset**
- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**
- **Edm.DateTime**
- **Edm.Guid**

The **ltExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **ltExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the value of the first operand is less than the value of the second operand and false if not. Actual ordering and comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when ordering and comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

leExpression: A data service MAY support the binary less than or equal to the operator. The operation of evaluating whether one expression is less than or equal to the other expression is represented as an **leExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<20> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.DateTimeOffset**
- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**
- **Edm.DateTime**
- **Edm.Guid**

The **leExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **leExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the value of the first operand is less than or equal the value of the second operand, and false if not. Actual ordering and comparison of values is data service specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when ordering and comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

gtExpression: A data service MAY support the binary greater than operator. The operation of evaluating whether one expression is greater than the other expression is represented as a **gtExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<21> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.DateTimeOffset**

- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**
- **Edm.DateTime**
- **Edm.Guid**

The **gtExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **gtExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the value of the first operand is greater than or equal to the value of the second operand, and false if not. Actual ordering and comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when ordering and comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

geExpression: A data service MAY support the binary greater than or equal operator. The operation of evaluating whether one expression is greater than or equal to the other expression is represented as a **geExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<22> support some or all of the common expressions as operands of the operation. Those operand expressions MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.DateTimeOffset**
- **Edm.Time**
- **Edm.Byte**
- **Edm.SByte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**
- **Edm.String**

- **Edm.DateTime**
- **Edm.Guid**

The **geExpression** SHOULD NOT be supported for any other EDM primitive types.

If supported, a data service SHOULD follow the binary numeric promotion rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the operands to a common supported EDM primitive type. The EDM primitive type of the result of evaluating the **geExpression** MUST be **Edm.Boolean**.

If supported, a data service MUST return a value of true if the value of the first operand is greater than or equal to the value of the second operand, and false if not. Actual ordering and comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when ordering and comparing values.

The data service MAY support evaluating operands with null values following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

notExpression: A data service MAY support the unary logical negation operator. The operation of logically negating an expression is represented by the **notExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<23> support some or all of the common expressions as operand values of the operation as long as the operand expression evaluates to a value of the EDM primitive type **Edm.Boolean**. The data service SHOULD NOT support operand expressions of any other EDM primitive type for the **notExpression**.

The EDM primitive type of the result of evaluating the **notExpression** MUST be **Edm.Boolean**.

If supported, the data service MUST evaluate the logical negation operation by returning false if the operand value is true and returning true if the operand value is false.

The data service MAY support evaluating an operand with a null value following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

isofExpression: A data service MAY support the isof operation. The operation of checking whether an instance is compatible with a given type is represented by the **isofExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<24> support some or all of the common expressions as the first operand value. In addition, the data service MAY support the first operand as being optional. In the case where it is not included, then the isof operation is interpreted to apply to the entity instance specified by the navigation portion of the request URI. The second operand MUST be a stringUriLiteral that represents the name of a known entity or EDM primitive type.

The EDM primitive type of the result of evaluating the **isofExpression** MUST be **Edm.Boolean**.

If supported, the data service MUST evaluate the **isofExpression** to return a value of true if the targeted instance MAY be converted to the specified type. If the conversion is not allowed, the expression MUST be evaluated to false.

The data service MAY support evaluating an operand with a null value following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

castExpression: A data service MAY support the cast expression. The operation of converting an expression to a given type is represented by the **castExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<25> support some or all of the common expressions as the first operand value. In addition, the data service MAY support the first operand as being optional. In the case where it is not included, then the cast operation is interpreted to apply to the entity instance specified by the navigation portion of the request URI. The second operand MUST be a stringUriLiteral that represents the name of a known entity or EDM primitive type to convert the first operand to.

The type of the result of evaluating the **castExpression** MUST be the same type as represented by the string literal value from the second operand. A data service MAY support any cast operations where there exists an explicit conversion from the targeted instance (first operand) to the type represented by second operand. In all other cases, the data service SHOULD NOT support the specified cast operation.

The data service MAY support evaluating an operand with a null value following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

boolCastExpression: A data service MAY support the Boolean cast expression. The operation of converting an expression to a Boolean value is represented by the **boolCastExpression** common expression in the common expression syntax. If this operation is supported, the data service MAY<26> support some or all of the common expressions as the first operand value. In addition, the data service MAY support the first operand as being optional. In the case where it is not included, then the cast operation is interpreted to apply to the entity instance specified by the navigation portion of the request URI. The second operand MUST be the stringUriLiteral "Edm.Boolean".

The type of the result of evaluating the **boolCastExpression** MUST be EDM primitive type **Edm.Boolean**. A data service MAY support any cast operations where there exists an explicit conversion from the targeted instance (first operand) to the EDM primitive type **Edm.Boolean**. In all other cases, the data service SHOULD NOT support the specified cast operation.

The data service MAY support evaluating an operand with a null value following the rules defined in Lifted Operators (section 2.2.3.6.1.1.5).

boolLiteralExpression: A data service MAY support expressions that are literals representing a Boolean value. These expressions are represented by the **boolLiteralExpression** common expression in the common expression syntax.

If supported, the type of the **boolLiteralExpression** MUST always be the EDM primitive type **Edm.Boolean**.

literalExpression: A data service MAY support expressions that are literals. These expressions are represented by the **literalExpression** common expression in the common expression syntax.

If supported, the type of the **literalExpression** MUST be the EDM primitive type for the lexical representation of the literal, as specified in Abstract Type System (section 2.2.2).

methodCallExpression: A data service MAY support the **methodCallExpression** common expression. If supported, a **methodCallExpression** MUST represent a method call in the common expression syntax.

boolMethodCallExpression: A data service MAY support the **boolMethodCallExpression** common expression. If supported, a **boolMethodCallExpression** MUST be a method call expression that evaluates to the EDM primitive type **Edm.Boolean**.

endsWithMethodCallExpression: A data service MAY support the **EndsWith** method. This method call is represented as an **endsWithMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<27> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **endsWithMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **endsWithMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.Boolean**.

If supported, the data service SHOULD evaluate the method call represented by the **endsWithMethodCallExpression** by returning a Boolean value indicating whether the end of the first parameter value matches the second parameter value. Actual comparison of values is data

service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when comparing values.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type.

indexOfMethodCallExpression: A data service MAY support the **IndexOf** method. This method call is represented as an **indexOfMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<28> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **indexOfMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **indexOfMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **indexOfMethodCallExpression** by returning an integer value indicating the index of the first occurrence of the second parameter value in the first parameter value. If no index is found, a value of -1 SHOULD be returned. Actual comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when comparing values.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type.

replaceMethodCallExpression: A data service MAY support the **replace** method. This method call is represented as a **replaceMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<29> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **replaceMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **replaceMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **replaceMethodCallExpression** by returning a string value with all occurrences of the second parameter value replaced by the third parameter value in the first parameter value. Actual comparison of values is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when comparing values.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters, as defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4), to implicitly convert the parameters to a supported EDM primitive type.

startsWithMethodCallExpression: A data service MAY support the **startswith** method. This method call is represented as a **startsWithMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<30> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **startsWithMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **startsWithMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.Boolean**.

If supported, the data service SHOULD evaluate the method call represented by the **startsWithMethodCallExpression** by returning a Boolean value indicating whether the beginning

of the first parameter values matches the second parameter value. Actual comparison of values is data service-specific and no semantics for doing so are mandated, however a data service MUST always use consistent semantics when comparing values.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type.

toLowerMethodCallExpression: A data service MAY support the **tolower** method. This method call is represented as a **toLowerMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<31> support some or all of the common expressions as the parameter of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **toLowerMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **toLowerMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **toLowerMethodCallExpression** by returning a string value with the contents of the parameter value converted to lower case. Actual definition of lower case is data service-specific and no semantics are mandated; however, a data service MUST always use consistent semantics when converting to lower case.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters, defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4), to implicitly convert the parameters to a supported EDM primitive type.

toUpperMethodCallExpression: A data service MAY support the **toupper** method. This method call is represented as a **toUpperMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<32> support some or all of the common expressions as the parameter of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **toUpperMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **toUpperMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **toUpperMethodCallExpression** by returning a string value with the contents of the parameter value converted to upper case. Actual definition of upper case is data service-specific and no semantics are mandated; however, a data service MUST always use consistent semantics when converting to upper case.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters, defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4), to implicitly convert the parameters to a supported EDM primitive type.

trimMethodCallExpression: A data service MAY support the **trim** method. This method call is represented as a **trimMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<33> support some or all of the common expressions as the parameter of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **trimMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **trimMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **trimMethodCallExpression** by returning a string value with the contents of the parameter value

with all leading and trailing white-space characters removed. Actual definition of white space is data service-specific and no semantics are mandated; however, a data service MUST always use consistent semantics when identifying white space.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters, defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4), to implicitly convert the parameters to a supported EDM primitive type.

substringMethodCallExpression: A data service MAY support the **substring** method. This method call is represented as a **substringMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<34> support some or all of the common expressions as the parameters of this method. The first parameter expression MUST evaluate to a value of the EDM primitive type **Edm.String**. The second and third parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.Int32**.

The **substringMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **substringMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **substringMethodCallExpression** by returning the string value starting at the character index specified by the second parameter value in the first parameter string value. If the optional third parameter is specified, the resulting string is the length (in characters) of the third parameter value. Otherwise, the entire string from the specified starting index is returned.

subStringOfMethodCallExpression: A data service MAY support the **substringof** method. This method call is represented as a **substringOfMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<35> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **substringOfMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **substringOfMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.Boolean**.

If supported, the data service SHOULD evaluate the method call represented by the **substringOfMethodCallExpression** by returning a Boolean value indicating whether the first parameter string value occurs in the second parameter string value. Actual comparison of values is data service-specific and no semantics for doing so is mandated; however, a data service MUST always use consistent semantics when comparing values.

concatMethodCallExpression: A data service MAY support the **concat** method. This method call is represented as a **concatMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<36> support some or all of the common expressions as the parameters of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **concatMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **concatMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.String**.

If supported, the data service SHOULD evaluate the method call represented by the **concatMethodCallExpression** by returning a string value that is the first and second parameter values merged together with the first parameter value coming first in the result.

lengthMethodCallExpression: A data service MAY support the **length** method. This method call is represented as a **lengthMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<37> support some or all of the common expressions as the parameter of this method. The parameter expressions MUST evaluate to a value of the EDM primitive type **Edm.String**.

The **lengthMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types. If supported, the EDM primitive type of the result of evaluating the **lengthMethodCallExpression** SHOULD be a value of the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **lengthMethodCallExpression** by returning the number of characters in the specified parameter value. Actual definition of how to calculate string length is data service-specific and no semantics for doing so are mandated; however, a data service MUST always use consistent semantics when calculating the length.

getTotalOffsetMinutesMethodCallExpression: A data service MAY support the **getTotalOffsetMinutes** method. This method call is represented as a **getTotalOffsetMinutesMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<38> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of the EDM primitive type **Edm.DateTimeOffset**.

The **getTotalOffsetMinutesMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **getTotalOffsetMinutesMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **getTotalOffsetMinutesMethodCallExpression** by returning the signed number of minutes in the time zone offset part of the *DateTimeOffset* parameter value, evaluated in the time zone of the *DateTimeOffset* parameter value.

yearMethodCallExpression: A data service MAY support the **year** method. This method call is represented as a **yearMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<39> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of either the EDM primitive type **Edm.DateTime** or the EDM primitive type **Edm.DateTimeOffset**.

The **yearMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **yearMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **yearMethodCallExpression** by returning the year component value of the parameter value.

monthMethodCallExpression: A data service MAY support the **month** method. This method call is represented as a **monthMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<40> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of either the EDM primitive type **Edm.DateTime** or the EDM primitive type **Edm.DateTimeOffset**.

The **monthMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **monthMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **monthMethodCallExpression** by returning the month component value of the parameter value.

dayMethodCallExpression: A data service MAY support the **day** method. This method call is represented as a **dayMethodCallExpression** common expression in the common expression

syntax. If this method is supported, the data service MAY<41> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of either the EDM primitive type **Edm.DateTime** or the EDM primitive type **Edm.DateTimeOffset**.

The **dayMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **dayMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **dayMethodCallExpression** by returning the day component value of the parameter value.

hourMethodCallExpression: A data service MAY support the **hour** method. This method call is represented as an **hourMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<42> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of EDM primitive type **Edm.DateTime**, **Edm.DateTimeOffset**, or **Edm.Time**.

The **hourMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **hourMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **hourMethodCallExpression** by returning the hour component value of the parameter value using a 24-hour range to cover an entire day without an AM/PM indicator and by starting at 0.

minuteMethodCallExpression: A data service MAY support the **minute** method. This method call is represented as a **minuteMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<43> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of EDM primitive type **Edm.DateTime**, **Edm.DateTimeOffset**, or **Edm.Time**.

The **minuteMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **minuteMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **minuteMethodCallExpression** by returning the minute component value of the parameter value.

secondMethodCallExpression: A data service MAY support the **second** method. This method call is represented as a **secondMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<44> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of EDM primitive type **Edm.DateTime**, **Edm.DateTimeOffset**, or **Edm.Time**.

The **secondMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, the EDM primitive type of the result of evaluating the **secondMethodCallExpression** SHOULD be the EDM primitive type **Edm.Int32**.

If supported, the data service SHOULD evaluate the method call represented by the **secondMethodCallExpression** by returning the second component value of the parameter value.

roundMethodCallExpression: A data service MAY support the **round** method. This method call is represented as a **roundMethodCallExpression** common expression in the common expression

syntax. If this method is supported, the data service MAY<45> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**

The **roundMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type. The EDM primitive type of the result of evaluating the **roundMethodCallExpression** MUST be the same type as the parameter.

If supported, the data service SHOULD evaluate the method call represented by the **roundMethodCallExpression** by returning the nearest integral value to the parameter value, following the rules defined in [IEEE754-2008] for the rounding operation.

floorMethodCallExpression: A data service MAY support the **floor** method. This method call is represented as a **floorMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<46> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**

The **floorMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type. The EDM primitive type of the result of evaluating the **floorMethodCallExpression** MUST be the same type as the parameter after the promotion rules have been applied.

If supported, the data service SHOULD evaluate the method call represented by the **floorMethodCallExpression** by returning the largest integral value less than or equal to the parameter value, following the rules defined in [IEEE754-2008] for the floor operation.

ceilingMethodCallExpression: A data service MAY support the **ceiling** method. This method call is represented as a **ceilingMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY<47> support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.Decimal**
- **Edm.Double**

The **ceilingMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types.

If supported, a data service SHOULD follow the numeric promotion rules for method call parameters defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4) to implicitly convert the parameters to a supported EDM primitive type. The EDM primitive type of the result of evaluating the **floorMethodCallExpression** MUST be the same type as the parameter after the promotion rules have been applied.

If supported, the data service SHOULD evaluate the method call represented by the **ceilingMethodCallExpression** by returning the smallest integral value greater than or equal to the parameter value, following the rules defined in [IEEE754-2008] for the ceiling operation.

distanceMethodCallExpression: A data service MAY support the **geo.distance** method. This method call is represented as a **distanceMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.GeographyPoint**
- **Edm.GeometryPoint**

The **distanceMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types, except in compliance with Geospatial Extension Methods (section 2.2.3.6.1.1.8).

If supported, a data service MAY perform coordinate transformations, as specified in Geospatial Coordinate Transformations (section 2.2.3.6.1.1.7) to implicitly convert the parameters to the same EDM primitive type, with the same system reference identifier (SRID). For further details about SRID, see [MC-CSDL] **Edm.GeographyPoint** SRID (section 2.2.1.18.1.1) and **Edm.GeometryPoint** SRID (section 2.2.1.26.1.1). If the system does not support coordinate transformations, it MUST require that the two arguments be of the same type and in the same SRID. The EDM primitive type of the result of evaluating the **distanceMethodCallExpression** MUST be **Edm.Double**.

If supported, the data service SHOULD evaluate the method call that is represented by the **distanceMethodCallExpression** by computing the geospatial distance between the two points in the coordinate reference system signified by the two points' SRIDs, according to an algorithm that is correct for the datum of that coordinate reference system. For **Edm.GeometryPoint**, the algorithm is minimum-distance, as defined in [OGC-SFOLECOM]. Multiple reasonable algorithms exist for **Edm.GeographyPoint**, such as minimum-distance or length of great elliptical arc. The implementation SHOULD choose some reasonable algorithm.

If the implementation supports both **geoLengthMethodCallExpression** and **distanceMethodCallExpression**, it MUST use compatible algorithms for these two operations. In particular, for any two positions A and B, geo.distance (Point(A), Point(B)) MUST equal geo.Length (LineString(A, B)).

geoLengthMethodCallExpression: A data service MAY support the **geo.length** method. This method call is represented as a **geoLengthMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value of one of the following EDM primitive types:

- **Edm.GeographyLineString**
- **Edm.GeometryLineString**

The **distanceMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types, except in compliance with Geospatial Extension Methods (section 2.2.3.6.1.1.8).

The EDM primitive type of the result of evaluating the **distanceMethodCallExpression** MUST be **Edm.Double**.

If supported, the data service SHOULD evaluate the method call represented by the **geoLengthMethodCallExpression** by computing the path length of the LineString in the coordinate reference system signified by its SRID, according to an algorithm that is correct for the datum of that coordinate reference system. For **Edm.GeometryLineString**, the algorithm is sum of minimum-distance, as defined in [OGC-SFOLECOM]. Multiple reasonable algorithms exist for

Edm.GeographyLineString, such as sum of minimum-distance or sum of length of great elliptical arc. The implementation SHOULD choose some reasonable algorithm.

If the implementation supports both **geoLengthMethodCallExpression** and **distanceMethodCallExpression**, it MUST use compatible algorithms for these two operations. In particular, for any two positions A & B, geo.distance(Point(A), Point(B)) MUST equal geo.Length(LineString(A, B)).

intersectsMethodCallExpression: A data service MAY support the **geo.intersects** method. This method call is represented as an **intersectsMethodCallExpression** common expression in the common expression syntax. If this method is supported, the data service MAY support some or all of the common expressions as the parameter of this method. The parameter expression MUST evaluate to a value such that the arguments are one of the following sets of EDM primitive types:

- **Edm.GeographyPoint** and **Edm.GeographyPolygon**, in either order
- **Edm.GeometryPoint** and **Edm.GeometryPolygon**, in either order

The **intersectsMethodCallExpression** SHOULD NOT be supported for parameters of any other EDM primitive types, except in compliance with Geospatial Extension Methods (section 2.2.3.6.1.1.8).

If supported, a data service MAY perform coordinate transformations, as specified in Geospatial Coordinate Transformations (section 2.2.3.6.1.1.7) to implicitly convert the parameters to a supported set of EDM primitive types, with the same SRID. If the system does not support coordinate transformations, it MUST require that the two arguments be of a supported set of types and in the same SRID. The EDM primitive type of the result of the evaluation of the **distanceMethodCallExpression** MUST be **Boolean**.

If supported, the data service SHOULD evaluate the method call that is represented by the **intersectsMethodCallExpression** by computing whether the point lies within the interior or the boundary of the polygon, as defined in [OGC-SFOLECOM].

implicitVariableExpression: Within a **lambdaPredicateExpression** a data service SHOULD support an **implicitVariableExpression** ($it) to unambiguously refer to a variable that represents the members of the collection that are being filtered by the **filterQueryOption**.

lambdaVariableExpression: To allow a **lambdaPredicateExpression** to refer to the members of the collection that a **lambdaMethodCallExpression** call is bound to, a **lambdaVariableExpression** MUST be specified.

A variable name is specified by using a **lambdaVariableExpression** common expression in the common expression syntax. This variable name can then be used in a **lambdaPredicateExpression** to specify a filter over the corresponding collection.

inscopeVariableExpression: When **firstMemberExpression** expressions are inside a **lambdaPredicateExpression**, variable names MUST be used to avoid ambiguity. Referring to variables that are currently available (in scope) is represented by using an **inscopeVariableExpression** common expression in the common expression syntax.

Inside a **lambdaPredicateExpression** there are at least two collection variables that can be referred to, namely **$it** (**implicitVariableExpression**) and whatever explicit names have been introduced by **lambdaVariableExpression**.

lambdaMethodCallExpression expressions can be nested inside an **inscopeVariableExpression**, which would mean that three or more variables can be in scope.

lambdaPredicateExpression: A data service MAY support use of nested filters with **lambdaMethodCallExpression** expressions. The nested filter is applied to the members of the

collection to which the **lambdaMethodCallExpression** is bound. This nested filter is represented as a **lambdaPredicateExpression** common expression in the common expression syntax.

A **lambdaPredicateExpression** follows the same rules as a **boolCommonExpression**, with the additional restriction that any nested **firstMemberExpression** expressions MUST be disambiguated using a **lambdaPredicatePrefixExpression**.

lambdaPredicatePrefixExpression: This is a prefix based on an **inscopeVariableExpression** that MUST be prepended to all **firstMemberExpression** expressions that are found inside **lambdaPredicateExpression** expressions.

collectionPathExpression: lambdaMethodCallExpression expressions MUST be bound to either an **entityCollectionProperty** or an **entityNavProperty-es**. This is specified by using **collectionPathExpression** common expression in common expression syntax.

If the **collectionPathExpression** is nested inside a **lambdaPredicateExpression**, an **inscopeVariableExpression** MUST be used to identify the variable that the expression is bound to.

singlePathExpression: Sometimes the **entityCollectionProperty** or **entityNavProperty-es** that ought to be bound to **lambdaMethodCallExpression** is not directly available; in this case, clients might need to first go through an **entityNavProperty-et** or an **entityComplexProperty**. A data service MAY choose to support accessing nested collections by using a **singlePathExpression** common expression in common expression syntax.

If the **singlePathExpression** is nested inside a **lambdaPredicateExpression**, an **inscopeVariableExpression** MUST be used to identify what variable to bind the expression to.

lambdaMethodCallExpression: A **lambdaMethodCallExpression** is either an **anyMethodCallExpression** or an **allMethodCallExpression**.

anyMethodCallExpression: A data service MAY support the **any** method. This method MUST be bound to a collection, either an **entityNavProperty-es** or an **entityCollectionProperty**, via a **collectionPathExpression**.

If there is no **lambdaVariableExpression** or **lambdaPredicateExpression**, the data service MUST return false if the bound collection is empty or otherwise true.

If a **lambdaVariableExpression** is specified, a **lambdaPredicateExpression** MUST also be specified.

The data service method MUST return true if any members of the collection satisfy the filter that is specified in the **lambdaPredicateExpression** or otherwise false.

lambdaMethodCallExpression expressions can be nested inside an **anyMethodCallExpression**.

allMethodCallExpression: A data service MAY support the **all** method. This method call is represented as an **allMethodCallExpression** common expression in the common expression syntax.

The data service method MUST return true if there are no members in the bound collection.

The data service MUST return true if all members of the bound collection satisfy the filter that is specified by the **lambdaVariableExpression** or otherwise false.

lambdaMethodCallExpression expressions can be nested inside an **allMethodCallExpression**.

functionCallExpression: A data service MAY support **functionCallExpression**. Non-binding parameters to **functionCallExpression** MUST be provided inline.

Binding parameters to **functionCallExpression** expressions MUST be provided either inline, or via the optional **memberExpression** or **firstMemberExpression** that starts the **functionCallExpression**.

boolFunctionCallExpression: A data service MAY support **boolFunctionCallExpression**, which is a **functionCallExpression** with the added restriction that the **ReturnType** of the corresponding function MUST be a Boolean value.

functionParametersExpression: A data service MAY support passing one or more parameters in a **functionCallExpression** by using the **functionParametersExpression**.

functionParameterExpression: A parameter within a **functionParametersExpression** is represented by using a **functionParameterExpression** that contains the name and value of the parameter. The data service MAY<48> support some or all of the allowable expressions for the value of the parameter.

structuralValue: A data service MAY support passing a complex type, multi-value, entity, or collection of entities as the value in a **functionParameterExpression** by using a **structuralValue**.

entityReference: A data service MAY support passing an entity or collection of entities by reference as the value in a **functionParameterExpression** by using an **entityReference**.

2.2.3.6.1.1.2 Operator Precedence

The following table summarizes the precedence of operators in the common expression syntax. Operators are listed by operator category in order of precedence from highest to lowest. Operators in the same category have equal precedence.

Category	Expression	Common Expression
Grouping	(x)	parenExpression, boolParenExpression
Primary	x/m	memberExpression
Primary	x(…)	methodCallExpression, boolMethodCallExpression
Unary	-x	negateExpression
Unary	not x	notExpression
Unary	cast(T), cast(x, T)	castExpression
Multiplicative	x mul y	mulExpression
Multiplicative	x div y	divExpression
Multiplicative	x mod y	modExpression
Additive	x add y	addExpression
Additive	x sub y	subExpression
Relational and type testing	x lt y	ltExpression
Relational and type testing	x gt y	gtExpression
Relational and type testing	x le y	leExpression
Relational and type testing	x ge y	geExpression
Relational and type testing	isof(T), isof(x, T)	isofExpression

Category	Expression	Common Expression
Equality	x eq y	eqExpression
Equality	x ne y	neExpression
Conditional AND	x and y	andExpression
Conditional OR	x or y	orExpression

Table: Operator Precedence for Query Option Expressions

A data service MAY<49> support some or all of the common expressions that represent the operators above. For supported operators, the data service SHOULD evaluate the operators in a common expression in order of precedence of operator category.

2.2.3.6.1.1.3 Unary Numeric Promotions

A data service MAY support unary numeric promotions for the negation operator (**negateExpression** common expressions). Unary promotions consist of converting operands of type **Edm.Byte** or **Edm.Int16** to **Edm.Int32** and of type **Edm.Single** to **Edm.Double**.

2.2.3.6.1.1.4 Binary Numeric Promotions

A data service MAY support binary numeric promotion for operands of the following operations.

Operation	Common Expression
Addition	addExpression
Subtraction	subExpression
Multiplication	mulExpression
Division	divExpression
Modulo	modExpression
Equality	eqExpression
Non-Equality	neExpression
Greater Than	gtExpression
Less Than	ltExpression
Greater Than or Equal	geExpression
Less Than or Equal	leExpression

Table: Operations that Support Binary Numeric Promotion

If supported, binary numeric promotion SHOULD implicitly convert both operands to a common type and, in the case of the nonrelational operators, also become the return type.

If supported, a data service SHOULD support binary numeric promotion for the following Entity Data Model (EDM) primitive types:

- **Edm.Decimal**
- **Edm.Double**

- **Edm.Single**
- **Edm.Byte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**

If supported, binary numeric promotion SHOULD consist of the application of the following rules in the order specified:

- If either operand is of type **Edm.Decimal**, the other operand is converted to **Edm.Decimal** unless it is of type **Edm.Single** or **Edm.Double**.
- Otherwise, if either operand is **Edm.Double**, the other operand is converted to type **Edm.Double**.
- Otherwise, if either operand is **Edm.Single**, the other operand is converted to type **Edm.Single**.
- Otherwise, if either operand is **Edm.Int64**, the other operand is converted to type **Edm.Int64**.
- Otherwise, if either operand is **Edm.Int32**, the other operand is converted to type **Edm.Int32**
- Otherwise, if either operand is **Edm.Int16**, the other operand is converted to type **Edm.Int16**.
- If binary numeric promotion is supported, a data service SHOULD use a **castExpression** to promote an operand to the target type.

2.2.3.6.1.1.5 Lifted Operators

A data service MAY support the allowance of operators that operate on Entity Data Model (EDM) primitive types to also be used with nullable forms of those types for the following operations.

Type	Operation	Common Expression
unary	negate not	negateExpression notExpression
binary	add sub mul div mod	addExpression subExpression mulExpression divExpression modExpression
relational	gt ge lt le	gtExpression geExpression ltExpression leExpression
equality	eq ne	eqExpression neExpression
logical	and or	andExpression orExpression
member	/	memberExpression

Type	Operation	Common Expression
		boolPrimitiveMemberExpression

Table: Lifted operators

- If supported, for unary operators, a data service MUST return the value null if the operand value is null.
- If supported, for binary operators, a data service MUST return the value null if either operand value is null.
- If supported, for relational operators, a data service MUST return the value false if one or both of the operands is null.
- If supported, for equality operators, a data service MUST consider two null values equal and a null value unequal to any non-null value.
- If supported, for logical operators, a data service MUST return the true or false, even if one or more operand values are null. For the purposes of comparison semantics, any null operand to a logical operation SHOULD be treated as false.<50>
- If supported, for member operators, a data service MUST return null if any of the **NavigationProperties** are null.
- If supported, for Boolean expressions evaluated to the value of null, a data service MUST return the value of false.

2.2.3.6.1.1.6 Numeric Promotions for Method Call Parameters

A data service MAY support numeric promotions for method call parameters.

If supported, a data service SHOULD support binary numeric promotions for the following Entity Data Model (EDM) primitive types:

- **Edm.Decimal**
- **Edm.Double**
- **Edm.Single**
- **Edm.Byte**
- **Edm.Int16**
- **Edm.Int32**
- **Edm.Int64**

If supported, numeric promotions for method parameters SHOULD consist of the application of the following rules in the order specified:

- If either operand is of type **Edm.Double**, the other operand is converted to type **Edm.Double**.
- Otherwise, if either operand is of type **Edm.Single**, the other operand is converted to type **Edm.Single**.
- Otherwise, if either operand is of type **Edm.Decimal**, the other operand is converted to type **Edm.Decimal**.

- Otherwise, if either operand is of type **Edm.Int64**, the other operand is converted to type **Edm.Int64**.
- Otherwise, if either operand is of type **Edm.Int32**, the other operand is converted to type **Edm.Int32**
- Otherwise, if either operand is of type **Edm.Int16**, the other operand is converted to type **Edm.Int16**.
- If binary numeric promotion is supported, a data service SHOULD use a **castExpression** to promote an operand to the target type.

2.2.3.6.1.1.7 Geospatial Coordinate Transformations

A data service MAY choose to support transformation between geospatial coordinate systems. If so, the implementation MUST support one of the following two levels of transformation: coordinate transformations within a topology, or arbitrary coordinate transformations, that is, transformation between any two coordinate systems.

2.2.3.6.1.1.7.1 Coordinate Transformations Within a Topology

Coordinate transformations within a topology MUST NOT change the type of the **Edm.Primitive** that is being transformed. Coordinate transformations change the SRID, as well as the values for the positions in the value.

A data service MAY choose to allow coordinate transformations between any pairs of SRIDs that the data service is capable of transforming. If a service chooses to support a particular transformation, the algorithm that it chooses is undefined by this standard. However, it MUST meet the following two criteria:

- The geospatial value before and after the transformation MUST represent the same, or as a similar as possible, set of positions on the earth. One potential algorithm is to directly transform each control position within the geospatial value. Whichever algorithm is chosen MUST NOT result in a greater margin of error than this potential algorithm.
- The algorithm SHOULD comply with common practice in the geospatial field. If possible, it SHOULD follow a standard provided by a relevant standards body.

2.2.3.6.1.1.7.2 Arbitrary Coordinate Transformations

The types that inherit from **Edm.Geography** constitute the geographic topology, while the types that inherit from **Edm.Geometry** constitute the geometric topology. The former uses a round model of Earth, while the latter uses a flat model.

A service that supports arbitrary coordinate transformation MAY allow coordinate transformations that change the type of the value, but only to transform it between corresponding types in these two topologies. In particular, the type transformations MUST be between types in one of the following pairs, in either direction:

- **Edm.GeographyPoint** and **Edm.GeometryPoint**
- **Edm.GeographyLineString** and **Edm.GeometryLineString**
- **Edm.GeographyPolygon** and **Edm.GeometryPolygon**
- **Edm.GeographyCollection** and **Edm.GeometryCollection**
- **Edm.GeographyMultiPoint** and **Edm.GeometryMultiPoint**
- **Edm.GeographyMultiLineString** and **Edm.GeometryMultiLineString**

- **Edm.GeographyMultiPolygon** and **Edm.GeometryMultiPolygon**

The algorithm that is used for the coordinate transformation MUST meet all of the same requirements as specified in Coordinate Transformations Within a Topology (section 2.2.3.6.1.1.7.1).

2.2.3.6.1.1.8 Geospatial Extension Methods

A data service MAY support arbitrary method calls on geospatial types by using the function's extensibility point. If the data service supports methods beyond those that are defined in this standard, each method MUST meet exactly one of the following sets of criteria:

- Extends types that are supported for defined functions.
- Implements one of the functions defined in [OGC-SFOLECOM].
- Arbitrary extension in a private namespace.

2.2.3.6.1.1.8.1 Extending Type Support for Defined Functions

A data service MAY extend the **Edm.Primitive** types that are allowed as parameters to the standard geospatial functions. If the data service extends the **Edm.Primitive** types that are allowed as parameters to the standard geospatial functions, the implementation MUST meet the following criteria:

- The parameters MUST be from the same topology after any coordinate transformations are applied.
- For types in the geometric topology, the behavior of the function MUST match the [OGC-SFOLECOM] specification.
- For types in the geographic topology, the behavior SHOULD be as similar to the [OGC-SFOLECOM] specification as possible, taking into account the differences in the topologies.
- The method MUST be implemented as an overload of the defined method. In particular, it MUST have the same representation in URLs, varying only in the types of the parameters.

2.2.3.6.1.1.8.2 Implementing One of the Functions Defined in [OGC-SFOLECOM]

A data service MAY support additional functions as defined in [OGC-SFOLECOM]. If the data service supports additional functions as defined in [OGC-SFOLECOM], the implementation MUST meet the following criteria:

- The parameters MUST be from the same topology after any coordinate transformations are applied.
- The parameters MUST be of types on which the function is supported in [OGC-SFOLECOM].
- For types in the geometric topology, the behavior of the function MUST match the [OGC-SFOLECOM] specification.
- The function MAY be defined for types in the geographic topology. If it is, the behavior SHOULD be as similar to the [OGC-SFOLECOM] specification as possible, taking into account the differences in the topologies.
- The method MAY be named in the geo namespace. If so, the name MUST be the name as defined in [OGC-SFOLECOM], but translated to all lowercase. For example, a compliant version of the [OGC-SFOLECOM] function “Union” MAY be named "geo.union".

2.2.3.6.1.1.8.3 Arbitrary Extensions

A data service MAY provide any additional geospatial operations, by defining them as functions. Each such extension MUST meet the criteria of that section. Such extensions MUST NOT be placed in the geo namespace. They SHOULD be placed in a namespace under the control of that data service.

2.2.3.6.1.2 Evaluating System Query Options

Any combination of the system query options defined in this document MAY be present on a valid data service URI. A data service URI with more than one query option present MUST be evaluated as if the query options were applied to the resources identified by the resource path section of the URI, in the following order: **$filter**, **$inlinecount**, **$orderby**, **$skiptoken**, **$skip**, **$top**, **$expand**, **$select**, **$format**.

For example, using data from Appendix A: Sample Entity Data Model and CSDL Document (section 6), the resource identified by the data service URI `http://host/service/Customers?$expand=Orders&$filter=substringof(CompanyName, 'bikes')&$orderby=CompanyName asc&$top=2&$skip=3&$skiptoken='Contoso','AKFNU'&$inlinecount=allpages&$select=CustomerID,CustomerName,Orders` is determined as follows:

1. Start with the set of all EntityType instances in the Customers EntitySet in the data service.
2. Remove all customer instances that do not have "bikes" in their company name (the entities that do not satisfy the condition in the **$filter** query option).
3. Determine the count N of all customers identified in step 2.
4. Sort the set of customers identified in step 2 in ascending order using the values from the **CompanyName** property defined on the Customer **EntityType**.
5. Seek in to the collection up to the index identified by the **$skiptoken** query option. Select all entities at the index through the end of the collection.
6. Starting from the 4th entity (from the collection returned by step 4), as directed by **$skip**=3, select the next two entities (for example, the 5th and 6th entities in the set), as directed by **$top**=2.
7. Of the two entities returned from step 6, select only the **CustomerName** and **CustomerID** properties of the Customer entities and all properties of the Order entities. The preceding URI identifies the two entities (and their related entities) returned from this step.

2.2.3.6.1.3 Expand System Query Option ($expand)

The presence of the **$expand** system query option indicates that entities associated with the EntityType instance or EntitySet, identified by the resource path section of the URI, MUST be represented inline instead of as Deferred Content (section 2.2.6.2.6) and Deferred Content (section 2.2.6.3.9).

The server MUST include any actions or functions that are bound to the associated entities that are introduced via an **expandClause**, unless a select system query option is also included in the request and that **$select** requests that the actions/functions be omitted (section 2.2.3.6.1.11).

The following rules supplement the grammar below, which represents the syntax of this system query option.

```
expandQueryOp = "$expand=" expandClause *("," expandClause)
expandClause  = [ namespaceQualifiedEnitityType"/"] entityNavProperty
*(["/"namespaceQualifiedEnitityType]"/" entityNavProperty)
              ; section 2.2.3.1
```

The left most **entityNavProperty** in an **expandClause** MUST represent a NavigationProperty defined in the EntityType, or a subtype thereof, associated with the resource path section of the URI. A subsequent **NavigationProperty** in the same **expandClause** MUST represent a **NavigationProperty** defined on the **EntityType**, or a subtype thereof, represented by the prior **NavigationProperty** in the **expandClause**.

Redundant **expandClause** rules on the same data service URI MAY be considered valid, but MUST NOT alter the meaning of the URI.

Examples

```
http://host/service.svc/Customers?$expand=Orders
```

For each customer entity within the Customers **EntitySet**, the value of all associated Orders are represented inline.

```
http://host/service.svc/Orders?$expand=OrderLines/Product,Customer
```

For each Order within the Orders **EntitySet**, the following is represented inline:

- The Order lines associated to the Orders identified by the resource path section of the URI and the products associated to each Order line.
- The customer associated with each Order returned.

The OData 3.0 protocol supports specifying the namespace-qualified **EntityType** on which the **NavigationProperty** is defined as part of the expand statement.

```
http://host/service.svc/Customers?$expand=SampleModel.VipCustomer/InHouseStaff
```

For each Customer entity in the Customers **EntitySet**, the value of all associated **InHouseStaff** MUST be represented inline if the entity is of type **VipCustomer** or a subtype of that. For entity instances that are not of type **VipCustomer**, or any of its subtypes, that entity instance MUST be returned with no inline representation for the expanded **NavigationProperty**.

2.2.3.6.1.4 Filter System Query Option ($filter)

A data service URI with a **$filter** system query option identifies a subset of the entities in the EntitySet, identified by the resource path section of the URI, by only selecting the entities that satisfy the predicate expression specified by the query option.

The syntax of the filter system query option is defined as:

```
filterQueryOption = "$filter" [WSP] "=" [WSP] boolCommonExpression
```

Examples:

```
http://host/service.svc/Orders?$filter=ShipCountry eq 'France'
```

The set of Order entity instances where the **ShipCountry** is equal to the value "France".

```
http://host/service.svc/Orders?$filter = Customer/ContactName ne 'Fred'
```

The set of Order entity instances where the associated Customer entity instance has a **ContactName** not equal to the value "Fred".

The syntax, construction, and evaluation rules for **boolCommonExpression** are defined in Common Expression Syntax (section 2.2.3.6.1.1).

The OData 3.0 protocol supports specifying the namespace-qualified **EntityType** of the property that is used in the filter.

```
http://host/service.svc/Customers?$filter =
SampleModel.VipCustomer/CreditPurchases/CreditLimit gt '5000'
```

The previous example shows the set of Customer entity instances that are of subtype **VipCustomer**, or any of its subtypes, with a **CreditLimit** greater than $5000. If an entity instance in the Customers **EntitySet** is not of type **VipCustomer**, or any of its subtypes, this falsifies the terms within the predicate and MUST result in the exclusion of the entity instance from the results.

The OData 3.0 protocol supports **anyMethodCallExpression** and **allMethodCallExpression** used in the filter.

Examples:

```
http://host/service.svc/Orders?$filter=OrderLines/any(ol: ol/Quantity gt 10)
```

In the previous example, the Orders that have any Orderlines with a Quantity greater than 10.

```
http://host/service.svc/Orders?$filter=OrderLines/all(ol: ol/Quantity gt 10)
```

In the previous example, the Orders for which all Orderlines have a Quantity greater than 10.

2.2.3.6.1.5 Format System Query Option ($format)

A data service URI with a **$format** system query option specifies that a response to the request SHOULD use the media type specified by the query option.

If the **$format** query option is present in a request URI, it SHOULD take precedence over the value(s) specified in the Accept (section 2.2.5.1) request header.

The syntax of the format system query option is defined as follows.

```
formatQueryOp = "$format="
                ("verbosejson"
                 / "json"
                 / "atom"
                 / "xml"
                 / <a data service specific value indicating a format specific
                    to the specific data service>
                 / <An IANA-defined [IANA-MMT] content type>
                )
```

- If the value of the query option is "atom", the media type used in the response MUST be "application/atom+xml".
- If the value of the query option is "verbosejson", the media type used in the response MUST be "application/json;odata=verbose".

- If the value of the query option is "json", the media type used in the response MUST be "application/json".
- If the value of the query option is "xml", the media type used in the response MUST be "application/xml".

Examples:

```
http://host/service.svc/Orders?$format=verbosejson
```

The set of Order entities represented using the Verbose JSON media type, as specified in [RFC4627].

The **$format** query option MAY be used in conjunction with RAW format (section 2.2.6.4) to specify which RAW format is returned.

Example:

```
http://host/service.svc/Orders(1)/ShipCountry/$value/?$format=verbosejson
```

2.2.3.6.1.6 OrderBy System Query Option ($orderby)

A data service URI with a **$orderby** system query option specifies an expression for determining what values are used to order the entities in the EntitySet, identified by the resource path section of the URI.

The syntax of the OrderBy system query option is defined as:

```
orderByQueryOption = "$orderby" [WSP] "=" [WSP] commonExpression [WSP] [asc / desc]
                     *( "," [WSP]  commonExpression [WSP] [asc / desc])
```

If supported, the data service MUST return the entities, in order, based on the expression specified. If multiple expressions are specified and a data service supports sorting based on multiple values, then a data service MUST return the entities ordered by a secondary sort for each additional expression specified.

If the expression includes the optional asc clause or if no option is specified, the entities MUST be returned in ascending order. If the expression includes the optional desc clause, the entities MUST be returned in descending order. Actual ordering of results is data service specific and no semantics for doing so are mandated. However, a data service MUST always use the same semantics when ordering the results of a URI request.

Examples:

```
http://host/service.svc/Orders?$orderby=ShipCountry
```

The set of Order entity instances returned in ascending order of the **ShipCountry** property.

```
http://host/service.svc/Orders?$orderby = ShipCountry ne 'France' desc
```

The set of Order entity instances returned in descending order of the **ShipCountry** property not equal to the value "France".

The syntax, construction, and evaluation rules for **commonExpression** are defined in Common Expression Syntax (section 2.2.3.6.1.1).

The OData 3.0 protocol supports specifying the namespace-qualified **EntityType** of the property that is used in the order by clause.

```
http://host/service.svc/Customers?$orderby
=SampleModel.VipCustomer/CreditPurchases/CreditLimit desc
```

The set of Customer entity instances returned in descending order **CreditLimit** if the instance is of type **VipCustomer**. If the entity instance is not of type **VipCustomer**, or any of its subtypes, the order for that entity instance is undefined. Actual ordering of such entity instances is specific to the data service and no semantics for doing so are mandated; however, a data service MUST always use the same semantics when ordering the results for such a URI request.

2.2.3.6.1.7 Skip System Query Option ($skip)

A data service URI with a **$skip** system query option identifies a subset of the entities in the collection of entities identified by the resource path section of the URI. That subset is defined by seeking N entities into the collection and selecting only the remaining entities (starting with entity N+1). N is a positive integer specified by this query option.

The value of this query option, referred to as N in the preceding paragraph, MUST be an integer greater than or equal to zero. If a value less than 0 is specified, the URI is considered to be malformed.

If the data service URI contains a **$skip** query option, but does not contain an **$orderby** option, then the entities in the set MUST first be fully ordered by the data service. Such a full order SHOULD be obtained by sorting the entities based on their EntityKey values. While no ordering semantics are mandated, a data service MUST always use the same semantics to obtain a full ordering for all requests.

The syntax of the skip system query option is defined as follows.

```
skipQueryOp = "$skip=" 1*DIGIT
```

Examples:

```
http://host/service.svc/Orders?$order=OrderDate desc&$skip=10
```

The set of Order entities sorted by **ShippedDate** (descending), starting with the 11th order.

```
http://host/service.svc/Customers('ALFKI')/Orders?$skip=10
```

The set of Order entity type instances (associated with the Customer entity type instance identified by **EntityKey** value 'ALFKI') starting with the 11th order.

2.2.3.6.1.8 Top System Query Option ($top)

A data service URI with a **$top** system query option identifies a subset of the entities in the collection of entities, identified by the resource path section of the URI. This subset is formed by selecting only the first N items of the set, where N is a positive integer specified by this query option.

The value of this query option, referred to as N in the preceding paragraph, MUST be an integer greater than or equal to zero. If a value less than 0 is specified, the URI is considered to be malformed.

If the data service URI contains a **$top** query option, but does not contain an **$orderby** option, then the entities in the set MUST first be fully ordered by the data service. Such a full order SHOULD be obtained by sorting the entities based on their EntityKey values. While no ordering semantics are mandated, a data service MUST always use the same semantics to obtain a full ordering across requests.

The syntax of the top system query option is defined as follows.

```
topQueryOp = "$top=" 1*DIGIT
```

Examples:

```
http://host/service.svc/Orders?$orderby=ShippedDate desc&$top=20
```

The first 20 Order entity instances returned in descending order when sorted by the **ShippedDate** property.

```
http://host/service.svc/Orders?$top=20
```

The first 20 Order entity instances returned in order of a sorting scheme determined by the data service.

2.2.3.6.1.9 Skip Token System Query Option ($skiptoken)

The value of a **$skiptoken** system query option is an opaque token that MUST identify a starting point in the collection of entities identified by the URI containing the **$skiptoken** parameter. For example, the value of a **$skiptoken** query option could identify the first entity in a collection, the 3rd entity in a collection containing 10 entities, or any other position within the collection represented by the URI containing the **$skiptoken** parameter.

Since the value of a **$skiptoken** query option identifies an index into a collection of entities, a data service URI containing a **$skiptoken** query option identifies a subset of the entities identified by the resource path section of the URI. The subset identified consists of the entities in the entity set identified by the resource path section of the URI, starting from the first entity at the index identified by the value of the **$skiptoken** query option through the last entity in the entity set.

If the data service URI contains a **$skiptoken** query option, but does not contain an **$orderby** option that identifies a full ordering of the collection of entities identified by the URI, then the entities in the set MUST first be fully ordered by the data service. Such a full order SHOULD be obtained by sorting the entities based on their **EntityKey** values. While no ordering semantics are mandated, a data service MUST always use the same semantics to obtain a full ordering across different requests on the same entity set. The syntax of the skip token system query option is defined as follows.

```
skiptokenQueryOp = "$skiptoken=" 1*pchar
```

Examples:

```
http://host/service.svc/Orders?$orderby=OrderID&$skiptoken=13S35K
```

A subset of the Order entity instances (sorted by the **OrderID** property) starting from a position in the collection of all Order entities identified by the skip token parameter.

2.2.3.6.1.10 InlineCount System Query Option ($inlinecount)

Applies to the OData 2.0 and OData 3.0 protocols

A data service URI with an **$inlinecount** system query option specifies that the response to the request MUST include the count of the number of entities in the collection of entities, which are identified by the resource path section of the URI after all **$filter** system query options have been applied. A data service MAY support the **$inlinecount** system query option on a **RetrieveEntity** request. Actual counting of items in the **EntitySet** is data-service specific and no semantics for doing so are mandated. The InlineCount system query option is supported only in the OData 2.0 and OData 3.0 protocols.

The syntax of the InlineCount system query option is defined as follows.

```
inlineCountQueryOp = "$inlinecount="
                     ("allpages" / "none")
```

- If a value other than "allpages" or "none" is specified, the data service MUST return a 4*xx* error response code.
- If a value of "none" is specified, the data service MUST NOT include the count in the response.

For information about including the count when using the AtomPub, JSON, and Verbose JSON serialization formats, refer to section 2.2.6.2.8 of this document, to section 4.5.4 of [ODataJSON4.0], and to section 2.2.6.3.9.1 of this document, respectively.

Examples:

```
http://host/service.svc/Orders?$inlinecount=allpages
```

All order instances and the count of all order instances returned.

```
http://host/service.svc/Orders?$inlinecount=allpages&$top=10
```

Returns the first 10 order instances and the count of all matching order instances.

```
http://host/service.svc/Orders?$inlinecount=none&$top=10
```

Returns the first 10 order instances with no count of matching order instances.

```
http://host/service.svc/Orders?$inlinecount=allpages&$filter=ShipCountry eq 'France'
```

Returns all order instances with **ShipCountry** equal to "France" and the count of all order instances with **ShipCountry** equal to "France".

2.2.3.6.1.11 Select System Query Option ($select)

Applies to the OData 2.0 and OData 3.0 protocols

A data service URI with a **$select** system query option identifies the same set of entities as a URI without a **$select** query option; however, the presence of a **$select** query option specifies that a response from the data service SHOULD return a subset, as identified by the value of the **$select** query option, of the properties that would have been returned had the URI not included a **$select**

query option. A data service MAY return properties of the resources identified by the request URI beyond those identified by the **$select** query option.

The following rules supplement the following grammar that represents the syntax of this system query option. The select system query option is supported only in the OData 2.0 and OData 3.0 protocols.

```
selectQueryOp  = "$select=" selectClause
selectClause   = [WSP] selectItem [[WSP] "," selectClause] [WSP]
selectItem     = star / ["/"namespaceQualifiedEntityType](selectedProperty /
selectedNamedStream /  selectedAction / selectedFunction / (selectedNavProperty ["/"
selectItem]))
selectedProperty = entityProperty / entityComplexProperty
selectedNamedStream = entityNamedStream

selectedAction = allActions / individualAction
individualAction = ecQualifiedActionName ; section 2.2.1.3.1.
allActions = entityContainer ".*" ; section 2.2.1.3.1

selectedFunctions = allFunctions / individualFunction
individualFunction = ecQualifiedFunctionName; section 2.2.1.4.1.
allFunctions = functions ".*"

selectedNavProperty = entityNavProperty-es / entityNavProperty-et
star           = "*"
```

- The left most **selectedProperty**, **selectedNamedStream**, or **selectedNavProperty** in a **selectClause** MUST be a star or represent a member that is defined in the **EntityType**, or a subtype thereof, that is identified by the resource path section of the URI.
- A subsequent **selectedProperty** or **selectedNavProperty** in the same **selectClause** MUST represent a property defined on the **EntityType**, or a subtype thereof, that is represented by the prior navigation property in the **selectClause**.
- For AtomPub formatted responses: The value of a **selectQueryOp** applies only to the properties returned within the **m:properties** element as specified in section 2.2.6.2.2. For example, if a property of an entity type is mapped with the attribute **KeepInContent**=false, according to Customizable Feeds (section 2.2.6.2.2.1), to an element or attribute in the response, then that property MUST always be included in the response according to its customizable feed mapping.
- For JSON formatted responses: See [ODataJSON4.0] section 6.
- For Verbose JSON formatted responses: The value of a **selectQueryOp** applies only to the name/value pairs with a name that does not begin with two consecutive underscore characters.
- If a property is not requested as a **selectItem** (explicitly or via a star), it SHOULD NOT be included in the response.
- If a **selectedProperty** appears alone as a **selectItem** in a request URI, then the response MUST contain the value of the property as per the serialization rules defined in sections 2.2.6.2.2 and 2.2.6.3.3.
- If a star appears alone in a **selectClause**, all properties on the **EntityType** within the collection of entities identified by the last path segment in the request URI MUST be included in the response.
- If a star appears in a **selectItem** following a **selectedNavProperty**, all non-navigation properties of the entity or entities represented by the prior **selectedNavProperty** MUST be included in the response.
- If a navigation property appears as the last segment of a **selectItem** and does not appear in an **$expand** query option, the entity or collection of entities identified by the navigation property MUST be represented as deferred content as described in sections 2.2.6.2.6 and 2.2.6.3.9.

- If a navigation property appears as the last segment of a **selectItem** and the same property is specified as a segment of a path in an **$expand** query option, then all the properties of the entity identified by the **selectItem** MUST be in the response. In addition, all the properties of the entities identified by segments in the **$expand** path after the segment that matched the **selectItem** MUST also be included in the response.
- If multiple **selectClause** instances exist in a **$select** query option, then the total set of property values to be returned is equal to the union of the set of properties identified by each **selectClause**.
- Redundant **selectClause** rules on the same URI MAY be considered valid, but MUST NOT alter the meaning of the URI.
- The presence of a **selectClause** means that both actions and functions SHOULD be omitted from the response, unless they are reintroduced explicitly by using a **selectedAction** or **selectedFunction** clause. A star SHOULD NOT reintroduce actions or functions.

  The **individualAction** clause requests a single action by name.

  The **allActions** clause requests all actions that are known to the server and that are bindable to the entities in the response.

  If an action is requested as a **selectItem**, either explicitly by using an **individualAction** clause or implicitly by using an **allActions** clause, the server MUST include in the response information about how to invoke that action (sections 2.2.6.2.2.2 and 2.2.6.3.3.1) for each of the entities identified by the last path segment in the request URI, if and only if the action can be bound to those entities.

  If an action that is requested by an **individualAction** clause cannot be bound to the entities requested, the server MUST ignore the **individualAction** clause.

  The **individualFunction** clause requests a single function by name.

  The **allFunctions** clause requests all functions that are known to the server and that are bindable to the entities in the response.

  If a function is requested as a **selectItem**, either explicitly by using an **individualFunction** clause or implicitly by using an **allFunctions** clause, the server MUST include in the response information about how to invoke that function (sections 2.2.6.2.2.3 and 2.2.6.3.3.2) for each of the entities that are identified by the last path segment in the request URI, if and only if the function can be bound to those entities.

  If a function requested by an **individualFunction** clause cannot be bound to the entities that are requested, the server MUST ignore the **individualFunction** clause.

  The **selectClause** does not provide a way to request actions or functions that are bound to the feed definition.

The following set of examples use the data model described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) and describe the semantics for a base set of data service URIs using the **$select** system query option. From these base cases, the semantics of longer URIs are defined by composing the following rules.

Examples:

```
http://host/service.svc/Customers?$select=CustomerID,CompanyName,Address
```

In a response from a data service, only the **CustomerID**, **CompanyName**, and **Address** property values are returned for each customer entity within the Customers **EntitySet**. When a complex type is selected, all properties defined on that complex type property MUST be returned.

```
http://host/service.svc/Customers?$select=CustomerID,Orders
```

In a response from a data service, only the **CustomerID** property value and a link to the collection of related entities identified by the **Orders** navigation property SHOULD be returned for each customer entity within the Customers **EntitySet**. Note: For the **Orders** navigation property referenced in the above URI, the **$select** option only states a link to the corresponding collection of orders will be returned.

```
http://host/service.svc/Customers?$select=CustomerID,Orders&$expand=Orders/OrderDetails
```

In a response from a data service, only the **CustomerID** property of the customer's entities SHOULD be returned, but all the properties of the entities identified by the **Orders** and **OrderDetails** navigation properties SHOULD be returned.

```
http://host/service.svc/Customers?$select=*
```

In a response from a data service, all properties are returned for each customer entity within the Customers **EntitySet**. Note: The star syntax is used to refer to all properties of the entity that are identified by the path of the URI or all properties of a navigation property, but does not refer to actions or functions. In other words, the * syntax causes all properties on an entity to be included without traversing associations or including information about actions or functions.

```
http://host/service.svc/$select=CustomerID,Orders/*&$expand=Orders/OrderDetails
```

In a response from a data service, the **CustomerID** is included, along with the Order entities and all properties of the Order entities. But rather than including the fully expanded Order Detail entities referenced in the expand clause, each Order will contain a link that references the corresponding collection of Order Detail entities.

```
http://host/service.svc/Photos/?$select=Name,Thumbnail
```

In a response from a data service, only the **Name** property and the thumbnail named resource stream are included.

The OData 3.0 protocol supports specifying the namespace-qualified **EntityType** in conjunction with **$select**.

```
http://host/service.svc/Customers/SampleModel.VipCustomer/?$select=Logo
```

In a response from a data service, only the logo named resource stream is included for all **VipCustomer EntityTypes** or its subtypes.

```
http://host/service.svc/Customers/?$select= SampleModel.VipCustomer/Logo
```

In a response from a data service, only the logo named resource stream is included for all **VipCustomer EntityTypes** or its subtypes, with empty content for non-**VipCustomers**.

```
http://host/service.svc/Customers('ALFKI2')/?$select=SampleModel.VipCustomer/Logo
```

In a response from a data service, only the logo named resource stream is included for the single **VipCustomer** type identified.

```
http://host/service.svc/Customers/?$select=CustomerID,CompanyName,$actions.CreateOrder
```

In a response from a data service, only the **CustomerID** and **CompanyName** properties are included for all Customer **EntityType** and/or subtypes. Additionally, the **CreateOrders** action is included for each customer if and only if the action is bindable to that customer. All other actions and functions are omitted.

2.2.3.6.1.12 System Query Option: Additional Construction Rules

The following rules are in addition to the grammar rules that are defined in each of the individual system query option sections:

- If the last segment of the **ResourcePath** is an **entityNamedStream**, the system query options MAY include the format system query option. But all other query options MUST NOT be used.

2.2.3.6.2 Custom Query Options

Custom query options provide an extensible mechanism for data service-specific information to be placed in a data service URI query string. A custom query option is any query option of the form shown by the rule "customQueryOption" in URI Syntax (section 2.2.3.1). Custom query options MUST NOT begin with a "$" character because the character is reserved for system query options, as described in System Query Options (section 2.2.3.6.1).

2.2.3.6.3 Service Operation Parameters

Service operations represent the **FunctionImports**, as specified in [MC-CSDL], which accept only primitive type input parameters defined in the Entity Data Model (EDM) associated with a data service. If a **FunctionImport** requires input parameters, those parameters are passed via query string name/value pairs appended to the data service URI identifying the **FunctionImport**, as described in Resource Path: Semantics (section 2.2.3.5).

If a service operation requires input parameters, a null value can be specified for nullable type parameters by not including the parameter in the query string of the request URI.

To pass parameters to a service operation, the following syntax is used.

```
serviceOpParam = <The name of the parameter required by the Service Operation
                  addressed in the Resource Path>
                 "="
                 <The value of the Primitive type parameter formatted as per section
                  2.2.2>

; see the ABNF grammar in section 2.2.3.1 which describes how
; each parameter is to be composed to a data service URI query string.
```

Listing: ABNF Grammar for Service Operation Parameters

2.2.3.6.4 Function Parameters

Applies to OData 3.0 protocol

Functions are represented by using **FunctionImport** elements, as specified in Functions (section 2.2.1.4). Functions, unlike service operations and actions, have **IsSideEffecting** set to "false". If a **Function** requires input parameters, those parameters can be passed to the function in four possible ways:

- If the **FunctionImport** has **IsBindable** set to "true", the first (or binding) parameter MAY be provided by appending a **functionCall-partiallyBound** (section 2.2.3.1) to a resource path that represents that parameter value.
- If a **functionCall** or **functionCall-partiallyBound** (URI Syntax (section 2.2.3.1)) is the last call to a function in the resource path, the function's unbound parameters MAY be specified via query string name/value pairs that are appended to the data service URI that identifies the **FunctionImport**, as described in Resource Path: Semantics (section 2.2.3.5). The name of each of the function's unbound parameters in the corresponding **FunctionImport** is used as the name in the name/value pair. All unbound parameters MUST be provided either in the query string or inline in the URI Path. Primitive type parameters MAY be specified in the resource path between the parentheses of the **functionCall** or **functionCall-partiallyBound** (URI Syntax (section 2.2.3.1)) segments.
- Parameters, primitive or otherwise, MAY be passed by using parameter aliases (URI Syntax (section 2.2.3.1)). The aliases are introduced between the parentheses of the **functionCall** or **functionCall-partiallyBound** segments, in place of parameter values. Then the actual parameter values are specified via query string name/value pairs where the name is the alias that is introduced and the value is the parameter value provided.
- If parameters to a **functionCall** or **functionCall-partiallyBound** segment are specified inline either directly or by using parameter aliases, the parameter names MUST be included via the **functionParameterName** in the **functionParameter**.

```
; OData 3.0 only
functionParameter = functionParameterName
                "="
                ( functionParameterValue / functionParameterAlias )

functionParameterName = ; section 2.2.3.1
                      ; the name of a parameter of the final Function segment
                      ; in the Uri path, defined in the EDM model
                      ; associated with the Data Service, specified in the URI path

functionParameterAlias = ; section 2.2.3.1
                       ; the name of an alias, introduced in the URI path for a
                       ; particular parameter value, prefixed with '@'.

functionParameterValue = null
                         primitiveLiteral /
                         entityTypeBody /
                         entityCTBody /
                         collectionInVJson /
                         entityCollectionValueInVJson /
                         collectionInJson /
                         entityCollectionValueInJson

null = ; section 2.2.3.1
       ; not to be confused with nullLiteral (in JSON)

entityTypeBody   =  ; section 2.2.6.3.3

entityCTBody     =  ; section 2.2.6.3.4

collectionInVJson = ; a Collection of ComplexTypes or PrimitiveTypes formatted
                    ; in OData Verbose JSON format (section 2.2.6.3.3) and UrlEncoded.

entityCollectionValueInVJson = ; a collection of Entities formatted in
                               ; OData Verbose JSON format (section 2.2.6.3.2)
```

```
                                ; and UrlEncoded.

collectionInJson = ; a collection of primitive values or of complex values formatted
                   ; in JSON format as per [ODataJSON4.0] sections 7.3 and 7.4,
                   ; respectively, and UrlEncoded.

entityCollectionValueInJson = ; a collection of entities formatted in JSON format
                              ; as per [ODataJSON4.0] section 12 and UrlEncoded.
```

Listing: ABNF Grammar for Function Parameters

2.2.3.6.5 Action Parameters

Applies to the OData 3.0 protocol

Actions are represented by using **FunctionImport** elements, as specified in [MC-CSDL] (section 2.1.15).

Actions are invoked by using a POST request to a URI that identifies an action. Where specified, parameters to actions MUST be provided in the POST body. If not specified, parameter values MUST be assumed to be null.

The POST body MUST be encoded in JSON format (according to Action Parameters in [ODataJSON4.0] section 17) or in Verbose JSON format 2.2.6.3. Therefore, the Content-Type of the POST request SHOULD be set to application/json or application/json;odata=verbose, respectively.

When passing parameters by using the Verbose JSON format, the body MUST consist of a single JSON object. Each parameter that is specified in the body MUST be a top level property of this single JSON object, where the property name is the same as the parameter name and the property value is the standard OData Verbose JSON encoding for that parameter type (Verbose JSON Format (section 2.2.6.3)).

Any parameters to the action that is identified by the request URI that are omitted MUST be interpreted as having a null value.

A client MAY choose to provide no Body and Content-Type header if the Action has either no parameters or all parameter values are null.

To pass parameters to an action in a POST request body by using the preferred OData 3.0 JSON format, see Action Parameters in [ODataJSON4.0] section 17. To pass parameters by using the Verbose JSON format, the following syntax is used.

```
; OData 3.0 only
actionParametersInVJson = begin-object
                    [actionParameter]
                    *(value-seperator actionParameter)
                    end-object

actionParameter = quotation-mark
                  actionParameterName
                  quotation-mark
                  name-seperator
                  actionParameterValue

actionParameterName = *pchar   ; section 3.3 of [RFC3986]
     ; the name of a parameter to an Action defined as a
     ; FunctionImport in the EDM
     ; model associated with the data service

actionParameterValue = nullLiteral |
                       primitiveValueInVJson |
                       entityTypeBody |
                       entityCTBody |
```

```
                        collectionInVJson |
                        entityCollectionValueInVJson


primitiveValueInVJson = <EDMSimple type serialized as per section 2.2.6.3.1>


entityTypeBody   =  ; section 2.2.6.3.3
entityCTBody     =  ; section 2.2.6.3.4
collectionInVJson = ; section 2.2.6.3.3
entityCollectionValueInVJson = ; section 2.2.6.3.2
nullLiteral      =  ; section 2.2.3.6.1.1
begin-object     =  ; [RFC4627] section 2
name-seperator   =  ; [RFC4627] section 2
value-seperator  =  ; [RFC4627] section 2
```

Listing: ABNF Grammar for Action Parameters

2.2.3.7 Data Service Metadata

2.2.3.7.1 Service Document

For a client to interact with a data service, it needs to discover the locations of the available collections of resources AtomPub [RFC5023] defines Service Documents to support this discovery process.

The **ServiceRoot** of a data service MUST identify the Service Document for the data service.

Service Document (section 2.2.6.2.7) describes how Entity Data Model (EDM) constructs are represented in a Service Document. As per [RFC5023], AtomPub Service Documents MUST be identified with the "application/atomsvc+xml" media type (see [RFC5023] section 8).

See Service Document in [MS-ODATAJSON] section 2.1.16 for details about the representation of the data provided by a Service Document in the preferred OData 3.0 JSON format.

Service Document (section 2.2.6.3.12) describes a Verbose JSON representation of the data provided by a Service Document. The section also describes how EDM constructs are represented in the Verbose JSON-based Service Document. Verbose JSON Service Documents MUST be identified using the "application/jsonodata=verbose" media type.

EntitySets that are contained by a different **EntitySet** (see Containment (section 2.2.1.6)) MUST NOT have a corresponding collection for AtomPub (section 2.2.6.2.7) or a corresponding **EntitySet** for JSON ([MS-ODATAJSON] section 2.1.16) or for Verbose JSON (section 2.2.6.3.12) in the Service Document, as described in this section.

2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services

All data services SHOULD expose a conceptual schema definition language (CSDL) based metadata endpoint that describes the structure and organization of all the resources exposed as HTTP endpoints by a data service.

This document defines data-service-specific extensions and mappings to the constructs of a CSDL document. These extensions MUST be used by a data service in conjunction with the "dataservices" node defined in [MC-EDMX] such that an Entity Data Model Extensions (EDMX) document is returned from the URI identifying the **serviceRoot** with a "/$metadata" path segment appended to the path.

The syntax of the metadata document of a data service returned as the content of the **dataservices** element is described in [MC-EDMX]. As noted in [MC-EDMX], the contents of the **edmx:Edmx** element, in the context of a data service, is an **edmx:DataServices** element that contains one or more CSDL documents, as specified in [MC-CSDL], with data service annotations.

The data service CSDL annotations are described and highlighted in the XML schema that follows, as specified in [XMLSCHEMA1/2]. Elements that do not include data-service-specific annotations have been omitted from the XSD document. See [MC-CSDL] for a complete structural description of a CSDL document.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:cg="http://schemas.microsoft.com/ado/2006/04/codegeneration"
xmlns:edm="http://schemas.microsoft.com/ado/2006/04/edm"
xmlns:m="http:// http://schemas.microsoft.com/ado/2007/08/dataservices"
targetNamespace="http://schemas.microsoft.com/ado/2006/04/edm">


<!-- Elements not annotated for data services have been omitted.  See [MC-CSDL] for a
complete structural description of a CSDL document -->

 <!-- Elements extended to specify mime type content for data services -->

 <xs:element name="Property" type="edm:TComplexTypeProperty" minOccurs="0"
             maxOccurs="unbounded" />

 <xs:complexType name="TComplexTypeProperty">
   <xs:sequence>
      <xs:group ref="edm:GEmptyElementExtensibility" minOccurs="0"
                maxOccurs="1" />
   </xs:sequence>
   <xs:attributeGroup ref="edm:TCommonPropertyAttributes" />
   <!-- The m:TWebCustomizableFeedsAttributes attributeGroup is
 supported only in Oata 2.0 and OData 3.0 (see section 2.2.3.7.2.1)  -->
   <xs:attributeGroup ref="m:TWebCustomizableFeedsAttributes" />
   <xs:anyAttribute namespace="##other" processContents="lax" />
 </xs:complexType>

 <xs:attributeGroup name="TCommonPropertyAttributes">
    <xs:attribute name="Name" type="edm:TSimpleIdentifier" use="required" />
    <xs:attribute name="Type" type="edm:TPropertyType" use="required" />
    <xs:attribute name="Nullable" type="xs:boolean" default="true"
                  use="optional" />
    <xs:attribute name="DefaultValue" type="xs:string" use="optional" />
    <!-- Start Facets -->
    <xs:attribute name="MaxLength" type="edm:TMaxLengthFacet"
                  use="optional" />
    <xs:attribute name="FixedLength" type="edm:TIsFixedLengthFacet"
                  use="optional" />
    <xs:attribute name="Precision" type="edm:TPrecisionFacet"
                  use="optional" />
    <xs:attribute name="Scale" type="edm:TScaleFacet" use="optional" />
    <xs:attribute name="Unicode" type="edm:TIsUnicodeFacet"
                  use="optional" />
    <xs:attribute name="Collation" type="edm:TCollationFacet"
                   use="optional" />
    <!--End Facets -->
    <xs:attribute name="ConcurrencyMode" type="edm:TConcurrencyMode"
                  use="optional" />
    <xs:attribute ref="cg:SetterAccess" use="optional" />
    <xs:attribute ref="cg:GetterAccess" use="optional" />
    <!-- Data Service Attributes -->
    <xs:attribute name="m:MimeType" type="xs:string" use="optional" />
 </xs:attributeGroup>

 <!-- Elements extended to specify HTTP method information
     for data services -->
 <xs:element name="FunctionImport">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Documentation" type="edm:TDocumentation"
                     minOccurs="0" maxOccurs="1" />
```

```
        <xs:element name="Parameter" type="edm:TFunctionImportParameter"
                    minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attributeGroup ref="edm:TFunctionImportAttributes" />
    </xs:complexType>
  </xs:element>

  <xs:attributeGroup name="TFunctionImportAttributes">
    <xs:attribute name="Name" type="edm:TSimpleIdentifier" use="required" />
    <xs:attribute name="ReturnType" type="edm:TFunctionType" use="optional" />
    <xs:attribute name="EntitySet" type="edm:TSimpleIdentifier" use="optional" />
    <xs:attribute ref="cg:MethodAccess" use="optional" />
    <xs:attribute ref="m:HttpMethod" type="m:HttpMethod" use="optional" />
    <xs:attribute ref="m:IsAlwaysBindable" type="xs:boolean" default="false" />
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:attributeGroup>

  <xs:simpleType name="m:HttpMethod">
    <xs:restriction base="xs:string">
      <xs:enumeration value="POST" />
      <xs:enumeration value="PUT" />
      <xs:enumeration value="GET" />
      <xs:enumeration value="MERGE" />
      <xs:enumeration value="DELETE" />
      <xs:enumeration value="PATCH" />
    </xs:restriction>
  </xs:simpleType>

 <xs:element name="EntityType" type="edm:TEntityType" minOccurs="0" maxOccurs="unbounded" />
   <xs:complexType name="TEntityType">
     <xs:sequence>
       <xs:element name="Documentation" type="edm:TDocumentation"
  minOccurs="0" maxOccurs="1"
         />
       <xs:element name="Key" type="edm:TEntityKeyElement" minOccurs="0"
  maxOccurs="1" />
       <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="Property" type="edm:TEntityProperty" minOccurs="0"
            maxOccurs="unbounded" />
         <xs:element name="NavigationProperty" type="edm:TNavigationProperty"
  minOccurs="0"
            maxOccurs="unbounded" />
       </xs:choice>
       <xs:any namespace="##other" processContents="lax" minOccurs="0"
  maxOccurs="unbounded"
            />
     </xs:sequence>
     <xs:attributeGroup ref="edm:TDerivableTypeAttributes" />
     <--Data Service Attribute group. The m:TWebCustomizableFeedsAttributes attributeGroup
        is supported only in OData 2.0 and OData 3.0 (see section 2.2.3.7.2.1)  -->
     <xs:attributeGroup ref="m:TWebCustomizableFeedsAttributes" />
     <xs:attribute name="m:HasStream" type="xs:boolean" use="optional" />
     <xs:attribute ref="cg:TypeAccess" use="optional" />
     <xs:anyAttribute namespace="##other" processContents="lax" />
   </xs:complexType>
  </xs:element>

  <xs:complexType name="TEntityProperty">
     <xs:sequence>
       <xs:group ref="edm:GEmptyElementExtensibility" minOccurs="0"
  maxOccurs="1" />
     </xs:sequence>
     <xs:attributeGroup ref="edm:TCommonPropertyAttributes" />
     <xs:anyAttribute namespace="##other" processContents="lax" />
     <--Data Service Attribute group. The m:TWebCustomizableFeedsAttributes
         attributeGroup is supported only in OData 2.0 and OData 3.0 (see section 2.2.3.7.2.1)
 -->
      <xs:attributeGroup ref="m:TWebCustomizableFeedsAttributes" />
   </xs:complexType>
```

```
</xs:schema>
```

The following listing describes the extensions to CSDL [MC-CSDL] that are defined by this document (and shown in the previous XML schema).

IsDefaultEntityContainer: This attribute MUST be used on an EntityContainer element [MC-CSDL] to indicate which **EntityContainer** is the default container for the data service. Each CSDL document that is used to describe a data service MUST mark exactly one **EntityContainer** with this attribute to denote that it is the default.

The valid values for this attribute are "true" or "false". True signifies that the container is the default container and, therefore, does not need to be specified in a resource path (section 2.2.3.3). False signifies that the container is not the default container and needs to be specified in the URI according to the URI construction rules that are noted in section 2.2.3.4.

MimeType: This attribute MAY<51> be used on a **Property** element [MC-CSDL] to indicate the media type of the content to be stored in the property that is being defined by the XML element. Each **Property** element that defines an EDMSimpleType property MAY include exactly one occurrence of this attribute.

Any media type (see [IANA-MMT]) is a valid value for this attribute. If this attribute is present on a property definition, any RetrieveValue Request (RetrieveValue Request (section 2.2.7.2.5)) for the property MUST return a response that uses the specified MIME type as the content type of the response body.

HttpMethod: This attribute MUST be used on a **FunctionImport** element [MC-CSDL] (section 2.1.15)] to indicate the HTTP method that is to be used to invoke the **ServiceOperation** that exposes the **FunctionImport**. If this attribute is present, the **FunctionImport** MUST be callable by using the HTTP method that is specified.

IsAlwaysBindable: This attribute MAY be used on a **FunctionImport** element [MC-CSDL] (section 2.1.15) to indicate whether the corresponding action or function can be bound to all instances of the binding parameter type. This attribute MUST not be specified if the **IsBindable** attribute is set to false or omitted (see sections 2.2.1.3 and 2.2.1.4).

HasStream: This attribute MUST only be used on an **EntityType** element [MC-CSDL]. The presence of this attribute with a value of "true" on an **EntityType** element states that the entity type is associated with a Media Resource (for example, the entity type represents a Media Link Entry [RFC5023]).

In addition to the extensions defined in the preceding example, the following mapping of data service constructs to CSDL constructs MUST be used by a data service to generate its metadata document.

Service Operations: A service operation MUST be represented as a **FunctionImport** element in the data services' CSDL document. The **Name** attribute on the element MUST be equal to the name of the service operation that is exposed by the data service.

For an example of a data services EDMX document, see Appendix A: Sample Entity Data Model and CSDL Document (section 6).

The following listing describes the extensions to the EDM for data services packaging format [MC-EDMX] that is defined by this document.

DataServiceVersion: This attribute MUST be in the data service metadata namespace (`http://schemas.microsoft.com/ado/2007/08/dataservices`) and SHOULD be present on an **edmx:DataServices** element [MC-EDMX] to indicate the version of the data service CSDL annotations (attributes in the data service metadata namespace) that are used by the document. Consumers of a data-service metadata endpoint ought to first read this attribute value to determine if they can safely interpret all constructs within the document. The value of this attribute MUST be 1.0

unless a "FC_KeepInContent" customizable feed annotation (section 2.2.3.7.2.1) with a value equal to false is present in the CSDL document within the **edmx:DataServices** node. In this case, the attribute value MUST be 2.0 or greater.

In the absence of **DataServiceVersion**, consumers of the CSDL document can assume the highest **DataServiceVersion** they can handle.

Named resource streams: A named resource stream MUST be represented as a **Property** of type **Edm.Stream** under the **EntityType** element in the data services' conceptual schema definition language (CSDL) document. The **Name** attribute of the **Property** MUST be equal to the name of the named resource stream exposed by the data service.

Properties of type **Edm.Stream** are used to define the named resource streams of an **EntityType**. For example, this **EntityType** has two named resource streams (Thumbnail and PrintReady).

```
<EntityType Name="Photo" m:HasStream="true">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Name" Type="Edm.String" Nullable="true" />
  <Property Name="Thumbnail" Type="Edm.Stream" />
  <Property Name="PrintReady" Type="Edm.Stream" />

</EntityType>
```

2.2.3.7.2.1 Conceptual Schema Definition Language Document Extensions for Customizable Feeds

Applies to the OData 2.0 and OData 3.0 protocols

This section defines OData protocol specific extensions (shown in the XML schema that follows) to the data-service-specific metadata document that is defined in the preceding section. These attributes define customizable feed property mappings for the AtomPub format. Customizable feed property mappings are defined in the OData 2.0 and OData 3.0 protocols.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="qualified"
           elementFormDefault="qualified"
           targetNamespace="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:attributeGroup name="TWebCustomizableFeedsAttributes">
    <xs:attribute name="FC_KeepInContent"/>
    <xs:attribute name="FC_ContentKind"/>
    <xs:attribute name="FC_NsPrefix"/>
    <xs:attribute name="FC_NsUri" />
    <xs:attribute name="FC_SourcePath" />
    <xs:attribute name="FC_TargetPath" />
    <xs:attribute name="FC_Criteria" />
    <xs:attribute name="FC_CriteriaValue" />
  </xs:attributeGroup>
</xs:schema>
```

Listing: Conceptual Schema Definition Language Document XSD Schema Extensions for Customizable Feeds

The customizable feed property mappings are used to define a mapping from the properties of an **EntityType** to elements or attributes in any namespace (including the Atom namespace) in an AtomPub document. When a property is mapped to an element or to an attribute of an element, the value for the property is equal to the value of the specified element or attribute in the AtomPub

document. An example of a mapped property value is described in Retrieve a Single Entity with a Mapped Property by Using the AtomPub Format (section 4.2.2.1).

The following example shows a Conceptual Schema Definition Language (CSDL) definition of an entity type that includes customizable feed property mappings. The customizable feed annotations that are defined on the entity type map the EmployeeName property of the Employee entity type to the **atom:title** element in the Atom namespace. The following example also includes a mapping that specifies the City property of the Address **ComplexType** property to be represented by using the service-defined XML element **emp:Location xmlns:"http://www.microsoft.com"**.

```
<EntityType Name="Employee" m:FC_KeepInContent="true"
   m:FC_TargetPath="Location" m:FC_SourcePath="Address/City"
   m:FC_NsUri="http://www.microsoft.com" m:FC_NsPrefix="emp">
   <Key>
      <PropertyRef Name="EmployeeID" />
   </Key>
   <Property Name="EmployeeID" Type="Edm.String" Nullable="false"
      MaxLength="5" Unicode="true" FixedLength="true" />
   <Property Name="EmployeeName" Type="Edm.String" Nullable="false"
      MaxLength="40" Unicode="true" FixedLength="false"
      m:FC_KeepInContent="false"
      m:FC_TargetPath="SyndicationTitle"/>
   <Property Name="Address" Type="Sample.EAddress" Nullable="true" />
   <Property Name="Version" Type="Edm.Binary" Nullable="true" MaxLength="8"
      FixedLength="true" ConcurrencyMode="Fixed" />
</EntityType>

<ComplexType Name="EAddress">
  <Property Name="Street" Type="Edm.String" Unicode="true" />
  <Property Name="City" Type="Edm.String" Unicode="true"/>
</ComplexType>
```

A property mapping MUST be defined as an attribute on the **Property** element of an entity type (as in the previous example) or on the **EntityType** element that contains the property to be mapped. A single **EntityType** property MUST NOT have more than one property mapping defined.

The following listing describes the extensions to CSDL [MC-CSDL] that are defined by this section (and are shown in the preceding XML schema). These extensions are supported only in the OData 2.0 and OData 3.0 protocols.

FC_ContentKind: The **FC_ContentKind** attribute specifies the content type of the value of the property being mapped via a customizable feed mapping.

The syntax of the **FC_ContentKind** attribute is defined as follows.

```
FC_ContentKind = "text " / "html" / "xhtml"
```

If the **FC_ContentKind** property is not defined for an **EntityType** property, the value of the property is assumed to be "text ".

FC_KeepInContent: The **FC_KeepInContent** attribute specifies whether the value of the property on the **EntityType** of this attribute is to be included in both the element specified via the customizable feed mapping as well as the original **m:properties** elements inside the **atom:content** element, as specified in section 2.2.6.2.2. The value for the **FC_KeepInContent** attribute MUST have either the value "true" or "false". If the **FC_KeepInContent** attribute is not supplied in the mapping, the data service MUST function as if it were specified with a value of true.

FC_NsPrefix: The **FC_NsPrefix** attribute specifies the XML namespace prefix to use when an entity type's property is mapped to an XML element in a data-service-specific namespace. If the **FC_TargetPath** attribute of the property mapping does not specify an Atom element, then the

FC_NsPrefix attribute is optional. If the entity type property is being mapped to an AtomPub element, then the **FC_NsPrefix** attribute MUST NOT be specified. If the **FC_TargetPath** attribute of the property mapping does not specify an Atom element and the **FC_NsPrefix** attribute is not supplied, then this document does not mandate a particular prefix be used.

FC_NsUri: The **FC_NsUri** attribute specifies the namespace to use when the **FC_TargetPath** of the entity type's property mapping does not specify an AtomPub element. If the **FC_TargetPath** of the entity type's property mapping does not specify an AtomPub element, then the **FC_NsUri** attribute MUST be specified on the entity type's definition. If the property is being mapped to an AtomPub element, then the **FC_NsUri** attribute MUST NOT be specified on the entity type's definition.

FC_TargetPath: The **FC_TargetPath** attribute identifies the element within an **atom:entry** element to which to map the entity type's property. All mapped properties MUST be mapped to distinct elements within the Atom feed.

If the mapping is not to an Atom element, the value of the **FC_TargetPath** MUST identify either an element or an attribute on an element in the AtomPub document. All paths' expressions MUST be rooted at the **entry** element in the Atom feed.

The syntax of the target path is defined as follows.

```
targetPath = targetPathExpression ["/" attribute] / atomSpecificElement

targetPathExpresion = targetPathExpression "/" targetPathExpression
                / element

element = <the name of an element in the AtomPub document>

atomSpecificElementName =
                     ; the following values are defined in OData 2.0 and OData 3.0
                        "SyndicationAuthorName"
                         / "SyndicationAuthorEmail"
                         / "SyndicationAuthorUri"
                         / "SyndicationPublished"
                         / "SyndicationRights"
                         / "SyndicationTitle"
                         / "SyndicationUpdated"
                         / "SyndicationContributorName"
                         / "SyndicationContributorEmail"
                         / "SyndicationContributorUri"
                         / "SyndicationSource"

attribute = "@" <the name of an attribute in the AtomPub document>
```

A data service SHOULD allow customizable feed mappings for distinct entity type properties that have partially overlapping **FC_TargetPaths** (a set of **FC_TargetPaths** are overlapping when they have some target elements in common). A data service SHOULD support target paths that define elements with mixed content. For example, the following three target paths on distinct properties will create an element structure with overlapping target paths and XML elements with mixed content.

```
TargetPath1: "a/b/c"
TargetPath2: "a/b/d"
TargetPath3: "a/b"
```

The following is the resulting element structure.

```
<a>
   <b>
        <c>propertyValue1</c>
        <d>propertyValue2</d>
        propertyValue3
```

```
    </b>
</a>
```

FC_SourcePath: The **FC_SourcePath** attribute specifies the entity type property that is the source property for the mapping. If the property mapping is specified directly on a property definition (properties are defined by using the **Property** element) in the CSDL and the property is a primitive property, the mapping cannot specify the **FC_SourcePath** attribute and the property identified by the **Property** element MUST be used as the source property of the mapping. If the property mapping is defined on an **EntityType** element, the **FC_SourcePath** attribute MUST be specified and the value MUST be the name of either a primitive property defined on the **EntityType** or a primitive property of a **ComplexType**, which is the type of a property on the **EntityType**.

The syntax of the **FC_SourcePath** is defined as follows.

```
FC_SourcePath = FC_SourcePathExpression

FC_SourcePathExpression = FC_SourcePathExpression "/" FC_SourcePathExpression
                / element
```

The **FC_SourcePath** MUST NOT address an **EntityType** property that represents a **ComplexType** but MAY address a declared **EDMSimpleType** property that is defined on a **ComplexType**.

For example, using the **EntityType** Employee, as specified in Appendix A: Sample Entity Data Model and CSDL Document (section 6), specifying the City property of an Address **ComplexType** on an Employee **EntityType**, an **FC_TargetPath** value of "Location", an **FC_NsPrefix** value of "emp", and an **FC_NsUri** value of "http://www.microsoft.com" would create the following element under the **atom:entry** element.

```
<emp:Location xmlns:emp="http://www.microsoft.com">Seattle</emp:Location>
```

2.2.3.8 URI Equivalence

When determining if two URIs are equivalent, each URI SHOULD be normalized by using the rules specified in [RFC3987] and [RFC3986] and then compared for equality by using the equivalence rules specified in [RFC2616] section 3.2.3.

2.2.3.9 Canonical URIs

For data services conformant with the URI path construction rules defined in this specification, the canonical form of an absolute URI identifying a single EntityType instance MUST be formed by adding a single path segment to the Path Prefix. The path segment MUST be made up of the EntitySet name associated with the entity followed by the key predicate identifying the entity within the set.

For example, the URIs `http://host/service.svc/Customers('ALFKI')/Orders(1)` and `http://host/service.svc/Orders(1)` identify the same entity in the example data model shown in Appendix A: Sample Entity Data Model and CSDL Document (section 6). Following the rules outlining the canonical form for URIs identifying single entities, the canonical URI in the example is `http://host/service.svc/Orders(1)`.

If an **EntityType** instance is contained by another **EntityType** instance (see Containment (section 2.2.1.6)), the canonical URI for the contained **EntityType** instance SHOULD be relative to the canonical URI of the containing **EntityType** instance.

If the server follows the URI conventions that are specified by the OData protocol, the canonical URL for contained entities MUST start with the canonical URL of the container entity, followed by a '/', and

then the name of the containment **NavigationProperty**. If the container can contain at most one contained entity, the canonical URL is complete.

For example, if a BookAbstract **EntityType** contains at most one Book **EntityType** via a 'Book' containment **NavigationProperty**, the canonical URL of the Book would be as follows: `http://host/service.svc/BookAbstracts(1)/Book`.

More commonly, the container can contain many contained entities. For example, an 'Order' entity can contain many 'OrderLines' via the 'Lines' containment **NavigationProperty**. In situations like this, the URI conventions in the OData protocol state that the canonical URI of a contained entity has to include a **keyPredicate** (see Resource Path: Construction Rules (section 2.2.3.4)) that identifies the particular contained entity among the other contained entities. An example would be the following: `http://host/service.svc/Orders(1)/Lines(6)`.

Further, if a containment **NavigationProperty** is backed by an **AssociationType** that includes a Referential Integrity Constraint that indicates that the contained **EntityType** shares key values with its container **EntityType**, the **keyPredicate** that is used in the canonical URI for the contained **EntityType** instance ought to omit the key values that are shared between the containing and the contained **EntityType** instances.

2.2.4 HTTP Methods

This section describes only the **HTTP** methods that are defined by this document and are not specified in [RFC2616]. All additional **HTTP** methods that are referred to in this document are specified in [RFC2616].

2.2.4.1 PATCH/MERGE

Data services support two types of update operations: merge and replace. In accordance with [RFC5023] and as described in Update Request Types (section 2.2.7.3), the HTTP PUT method specifies that an update operation MUST be carried out by using replace semantics.

PATCH is an HTTP verb defined by [RFC5789] and is supported only in the OData 3.0 protocol. The semantics of the method for use in this protocol are as defined by [RFC5789], except where explicitly noted in this document.

Note that this protocol does not define the semantics for the OPTIONS request method that is defined in [RFC5789].

The remainder of this section defines a custom HTTP MERGE method that is used in the OData 1.0 and OData 2.0 protocols to specify that an update is to be completed by using merge semantics. All the directives defined for the PUT method in HTTP, as specified in [RFC2616], and AtomPub, as specified in [RFC5023], apply equally to the HTTP MERGE method. The only difference between an HTTP request that uses MERGE or PATCH and PUT is client intent.

Since MERGE is not one of the verbs that is defined in the HTTP specification [RFC2616], using the MERGE verb might not flow through network intermediaries as seamlessly as methods that are defined in the HTTP specification. The HTTP PATCH verb is preferred over HTTP MERGE when working with data services that support the OData 3.0 protocol. Data services that support the OData 2.0 and OData 3.0 protocols can support verb tunneling to mitigate this limitation, as defined in Tunneled Requests (section 2.2.7.7).

The semantics of a MERGE request on a data service entity is to merge the content in the request payload with the entity's current state. The merging is done by comparing each component (that is, each individual primitive-valued or collection-valued property of the entity or complex type) within the request body to the entity as it exists on the server.

If a component in the request body is not defined on the entity that is to be updated, the request MAY be considered malformed.

If a component in the request body does match a component on the entity that is to be updated, the value of the component in the request body MUST replace the matching component of the entity to be updated and the matching process continues with the children of the component from the request body.

The ABNF syntax of the HTTP MERGE method is defined as follows.

```
Method = "MERGE"    ; see [RFC2616] section 5.1.1
```

Listing: ABNF Grammar for HTTP MERGE Method

2.2.5 HTTP Header Fields

The OData protocol uses existing headers as specified in [RFC2616] as well as custom HTTP headers that are defined in this document. Some of the headers that are specified in [RFC2616] are further restrained in how they can be used, as specified in this document. These additionally restrained headers and the custom headers are defined in this section.

Unless otherwise specified, the headers that are defined in this document and any tokens (also called tags or directives) that are used on those headers are defined for use on both requests and responses.

If a client or server receives an HTTP header that is not defined in this section or if the header is not defined in the current context (for example, a request-only header is received in a response), the header MUST be ignored, as specified in [RFC2616].

If a client or server receives an HTTP header that is defined in this section and the header contains an unknown or malformed token, as specified in this section, the request or response that contains the header MUST be considered malformed.

If a client or server receives an HTTP header that is defined in this section, but the header contains a token that is not defined in the current context (for example, a request-only token is received in a response), the request or response that contains the header and token MUST be considered malformed.

This section defines the syntax of the HTTP headers that are defined in this section by using the Augmented Backus-Naur Form (ABNF) syntax, as specified in [RFC5234]. Any ABNF syntax rules that are not specified in [RFC5234] use the ABNF extensions that are specified in [RFC2616].

The grammar in this section is word-based. Except where noted otherwise, linear white space (LWS), as specified in [RFC2616], can be included between any two adjacent words (token or quoted-string) and between adjacent words and separators without changing the interpretation of a field. At least one delimiter (LWS and/or separators) MUST exist between any two tokens, as specified in [RFC2616], because they would otherwise be interpreted as a single token.

2.2.5.1 Accept

A primary goal of data services is to allow a client of the service to focus on the data being transmitted and not be required to understand a single data format. As such, the OData protocol that is defined in this document enables exchanging resources using AtomPub semantics in multiple serialization formats (AtomPub, JSON, Verbose JSON, and so on).

The nature of the client application using a data service and its runtime environment determines which format is best. For example, Asynchronous JavaScript (AJAX)-based applications that run inside web browsers might find JSON easier to use because this format can be directly turned into JavaScript

objects. On the other hand, a client application might be written with a language/runtime library that has a rich, built-in XML parser, making an XML-based format an appropriate choice.

The OData protocol uses the Accept request-header field, as specified in [RFC2616]. Once a requested format is determined using the rules specified in [RFC2616], the following Accept Request Header to Content-Type Response Header Mapping table is used to determine the value of the Content-Type response header and the format of the response payload.

Value of Accept request header	Value of Content-Type response header
/ Specified in [RFC2616] and [RFC2045]	application/atom+xml
text/* Specified in [RFC2046]	Behavior is not defined by this document
application/* Specified in [RFC2046]	Behavior is not defined by this document
text/plain Specified in [RFC3676]	text/plain
text/xml Specified in [RFC3023]	text/xml
application/xml Specified in [RFC3023]	application/xml
application/atom+xml Specified in [RFC5023]	application/atom+xml
application/atom+xml;type=entry Specified in [RFC5023]	application/atom+xml;type=entry
application/atom+xml;type=feed Specified in [RFC5023]	application/atom+xml;type=feed
application/json Specified in [RFC4627]	For OData 1.0 and OData 2.0 responses: application/json;odata=verbose For OData 3.0 responses: application/json
application/json;odata=verbose	application/json;odata=verbose

Table: Accept Request Header to Content-Type Response Header Mapping

If the server cannot send a response that is acceptable, as indicated in the preceding Accept Request Header to Content-Type Response Header Mapping table and according to the Accept header value, then, as specified in [RFC2616], the server SHOULD return a 4*xx* response.

The OData protocol can be extended to support arbitrary message formats. However, the scope of this section is to define the use of the application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), and application/json;odata=verbose (section 2.2.5.1.3) formats. A data service MAY accept requests with Accept header values other than those shown in the preceding table. The returned Content-Type response header value for such requests is not defined by this specification.

2.2.5.1.1 application/atom+xml

This content type is used to request the data service format for the response payload by using the application/atom+xml format according to the formatting rules that are outlined in AtomPub Format (section 2.2.6.2). A data service MUST support this content type.

2.2.5.1.2 application/json

For OData 1.0 and OData 2.0 services, this content type is used to request the data service format for the response payload according to the formatting rules that are outlined in Verbose JSON Format (section 2.2.6.3).

For OData 3.0 services, the application/json content type returns the preferred OData 3.0 JSON format as defined in [MS-ODATAJSON] section 2.1.5.

2.2.5.1.3 application/json;odata=verbose

This content type is used to request the data service format for the response payload by using the application/json;odata=verbose format according to the formatting rules that are outlined in Verbose JSON Format (section 2.2.6.3). A data service MAY support this content type.

2.2.5.2 Content-Type

The Content-Type header is used as specified in [RFC2616]. However, because this document describes messages for the application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), application/json;odata=verbose (section 2.2.5.1.3), application/xml, text/plain, and text/xml formats, a data service client or server SHOULD only use HTTP messages with a Content-Type header value as shown in the ABNF grammar that follows and is specified in [RFC5234]. The exception to the above rule is when messages are used that represent a Media Resource [RFC5023] or the raw value of an entity's property (see section 2.2.3.5).

```
Content-Type     = "Content-Type:"
                   ("application/atom+xml"
                   / "application/atom+xml;type=entry"
                   / "application/atom+xml;type=feed"
                   / "application/json;odata=verbose"
                   / "application/json"
                   / "application/xml"
                   / "text/plain"
                   / "text/xml"
                   / "octet/stream")
                   CRLF
                   ";" <Remainder of rule is per the Content-Type rule in [RFC2616]>
```

Listing: Content-Type Header ABNF Grammar

```
Example: Content-Type: application/atom+xml;charset=UTF-8
```

2.2.5.3 DataServiceVersion

This header is a custom HTTP header defined by this document for protocol versioning purposes. This header MAY be present on any request or response message.

The syntax of the DataServiceVersion header is defined as follows:

```
DataServiceVersion       =  "DataServiceVersion:"
                            VersionNum
                            [";"
                            VersionClientUserAgent]
```

```
                              CRLF

VersionNum                =   DIGIT *DIGIT "." DIGIT *DIGIT

VersionClientUserAgent    =   *token     ; see [RFC2616] section 2.2
                              ; SHOULD contain information about the user agent
                              ; originating the request
```

Listing: DataServiceVersion Header ABNF Grammar

Example: DataServiceVersion: 1.0;AspNetAjax

If present in a request, the VersionNum section of the header value states the version of the OData protocol that the client used to generate the request, as specified in the preceding DataServiceVersion Header ABNF Grammar listing.

If present in a response, the VersionNum section of the header value states the version of the OData protocol that the server used to generate the response, as specified in the preceding DataServiceVersion Header ABNF Grammar listing.

For additional processing rules for this header, see Versioning and Capability Negotiation (section 1.7).

The VersionClientUserAgent section, as specified in the previous listing in this section, DataServiceVersion Header ABNF GrammarService Operation Parameters, of the header value is not significant, SHOULD NOT be interpreted by a data service, and SHOULD NOT affect the versioning semantics of a data service.

This document defines OData 1.0, OData 2.0, and OData 3.0.

2.2.5.4 ETag

An ETag (entity tag) is an HTTP response header returned by an HTTP/1.1 compliant web server used to determine change in content of a resource at a given URL. The value of the header is an opaque string representing the state of the resource at the time the response was generated.

The ETag header is used as specified in [RFC2616]. However, this document adds constraints to the header value to enable its use for optimistic concurrency control when performing operations that update entities on the server. Optimistic concurrency support is described in If-Match (section 2.2.5.5), If-None-Match (section 2.2.5.6), and later in this section.

In the Entity Data Model (EDM) associated with a service, some **EntityTypes** might have properties defined with a **concurrencyMode**, as specified in [MC-CSDL] section 2.2.4, whose value is "Fixed". Such **EntityTypes** are considered enabled for optimistic concurrency control. For example, the **Version** property on the Customer **EntityType** defined in the model shown in Appendix A: Sample Entity Data Model and CSDL Document (section 6) defines the entire concurrency token of the type because no other properties on the type have the **concurrencyMode** facet. If a base **EntityType** does not define a concurrency token, then that **EntityType**, and any of its subtypes, are not considered enabled for optimistic concurrency control.

When a server responds, indicating success, to a request performed against a URI that identifies a single entity, properties of an entity or a Media Resource (as specified in URI Format: Resource Addressing Rules (section 2.2.3)), and whose **EntityType** is enabled for optimistic concurrency control, it MUST include an ETag header in the HTTP response. The value of the header MUST represent the concurrency token for the entity that is identified by the request URI. The server MUST NOT include an ETag header in a response to any request performed against a URI that does not identify, as specified in URI Format: Resource Addressing Rules (section 2.2.3), a single entity, properties of an entity, or a Media Resource. In addition, the server MUST NOT include an ETag header if the request URI identifies a single entity whose type is not enabled for optimistic concurrency control. Server response requests performed against URIs that do not return single

entities MUST provide concurrency tokens in the response payload for any entities that are enabled for concurrency control.

For example, using the model in Appendix A: Sample Entity Data Model and CSDL Document, a valid URI identifying an EntityType instance is `http://host/service.svc/Customers('ALFKI')`. As a counter example, the URIs `http://host/service.svc/Customers` and `http://host/service.svc/Customers('ALFKI')?$expand=Orders` identify a collection of entities and thus MUST NOT include an ETag in a response associated with these request URIs.

A data service MAY<52> define concurrency tokens on the base **EntityType** associated with each EntitySet in the EDM used by the service. Concurrency tokens are defined using the **concurrencyMode** facet, with a value of "Fixed", on a property of an **EntityType**, as specified in [MC-CSDL].

An entity's concurrency token MUST be comprised of only primitive type values (NavigationProperties and ComplexTypes cannot be annotated with the **concurrencyMode** facet and therefore cannot participate in the concurrency token for an **EntityType**), as specified in [MC-CSDL]. Because this document relies on the definition of concurrency tokens per [MC-CSDL], optimistic concurrency control is not defined for links or associations. Therefore, a server cannot perform optimistic concurrency control for operations that create or remove associations. If a data service implementation is able to provide concurrency support on such operations without altering the rules in this section, it SHOULD do so.

Implementers of this document are recommended to order the property values that make up the concurrency token for an **EntityType**, and all of its subtypes, using the same order of the properties listed in the metadata document of the data service. RetrieveServiceMetadata Request (section 2.2.7.2.7) describes the Data Service Metadata document.

The syntax of the ETag header is defined as follows:

```
ETag       = "ETag" ":"
             entity-tag
             CRLF                    ; exactly as specified in [RFC2616] section 14.19

entity-tag = [weak] opaque-tag       ; exactly as specified in [RFC2616] section 3.11

weak       = "W/"                    ; exactly as specified in [RFC2616] section 3.11

; The rule below redefines the opaque-tag rule defined in [RFC2616]
opaque-tag = <ASCII encoded value of entityProperty rules from (section 2.2.3.1)>
["," opaque-tag]
```

Example following the model defined in Appendix A: Sample Entity Data Model and CSDL Document: ETag: W/"X'000000000000D2F3'"

2.2.5.5 If-Match

The If-Match request-header field is used with a method to make it conditional. As specified in [RFC2616], "the purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used, on updating requests, to prevent inadvertent modification of the wrong version of a resource".

The If-Match header is used in this document as specified in [RFC2616]. However, this document adds additional constraints to the types of requests for which the header can be provided. Additional constraints are also added to the syntax of the header value.

This header MAY<53> only be present on HTTP GET, MERGE, PATCH, or PUT requests to request URIs which identify the same Entity Data Model (EDM) constructs as URI 2, URI 3, URI 4, URI 5, and URI

17 that are defined in Resource Path: Semantics (section 2.2.3.5). Additionally, this header MAY<54> be present on DELETE requests to request URIs that identify the same EDM constructs as URI 2, as specified in Resource Path: Semantics (section 2.2.3.5), and any data service URI whose last path segment is "/$value".

Additionally, this header MAY be present on POST requests to invoke an action (section 2.2.1.3) bound to an entity. This allows clients to prevent an action from having inadvertent side effects based on the wrong version of a resource.

This header MUST NOT be on any POST request other than a request to invoke an action (section 2.2.1.3).

Client processing rules for this header are defined in Request Types (section 2.2.7) and server processing rules are in Message Processing Events and Sequencing Rules (section 3.2.5).

The syntax of the If-Match header is defined as follows:

```
; entity-tag is as per the definition in (section 2.2.5.4)
If-Match = "If-Match" ":" ( "*" / 1*entity-tag ) CRLF
```

Example: If-Match: W/"X'000000000000D2F3'"

2.2.5.6 If-None-Match

The If-None-Match request header is used with a method to make it conditional. As specified in [RFC2616], "The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used to prevent a method (for example, PUT) from inadvertently modifying an existing resource when the client believes that the resource does not exist."

The If-None-Match header is used in this document as specified in [RFC2616]. However, this document further limits it to the types of requests with which the header can be used. Additional constraints are also added to the syntax of the header value.

This header MAY<55> be present only on HTTP GET, MERGE, PATCH, or PUT requests to request URIs that identify the same Entity Data Model (EDM) constructs as URI 2, URI 3, URI 4, URI 5, and URI 17, as defined in the Resource Path Semantics table in Resource Path: Semantics (section 2.2.3.5). Additionally, this header MAY<56> be used on DELETE requests to URIs which identify the same EDM constructs as URI 2, as specified in the table in Resource Path: Semantics (section 2.2.3.5), and any data service URI whose last path segment is "/$value". This header MUST NOT be used on any POST requests to a data service.

Client processing rules for this header are defined in Request Types (section 2.2.7) and server processing rules are in Message Processing Events and Sequencing Rules (section 3.2.5).

The syntax of the If-None-Match header is defined as follows:

```
; entity-tag is as per the definition in (section 2.2.4.4)
If-None-Match = "If-None-Match" ":" ( "*" / 1*entity-tag ) CRLF
```

2.2.5.7 MaxDataServiceVersion

This header is a custom HTTP request only header defined by this document for protocol versioning purposes. This header MAY be present on any request message from client to server.

If present in a request, the VersionNum section, as specified in the following ABNF grammar list, of the header value states the maximum version of the protocol the client can accept in a response.

For additional processing rules for this header, see Versioning and Capability Negotiation (section 1.7).

The VersionServerUserAgent section, as specified in the following ABNF grammar list, of the header value is not significant and SHOULD NOT affect the versioning semantics of a data service.

The syntax of the MaxDataServiceVersion header is defined as follows:

```
MaxDataServiceVersion  =  "MaxDataServiceVersion: "
                          VersionNum               ; (section 2.2.5.3)
                          [";"
                          VersionServerUserAgent]
                          CRLF

VersionServerUserAgent =  <0 or more of any valid character in an HTTP header that
                          identifies the server sending the request>
```

Listing: Syntax of the MaxDataServiceVersion Header

```
Example: MaxDataServiceVersion: 1.0;AspNetAjax
```

2.2.5.8 X-HTTP-Method

This header is a custom HTTP request header defined by this document.

It is possible to instruct network intermediaries (proxies, firewalls, and so on) inspecting traffic at the application protocol layer (for example, HTTP) to block requests that contain certain HTTP verbs. In practice, GET and POST verbs are rarely blocked (traditional web pages rely heavily on these HTTP methods), while, for a variety of reasons (such as security vulnerabilities in prior protocols), other HTTP methods (PUT, DELETE, and so on) are at times blocked by intermediaries. Additionally, some existing HTTP libraries do not allow creation of requests using verbs other than GET or POST. Therefore, an alternative way of specifying request types which use verbs other than GET and POST is needed to ensure that this document works well in a wide range of environments.

To address this need, the X-HTTP-Method header can be added to a POST request that signals that the server MUST process the request not as a POST, but as if the HTTP verb specified as the value of the header was used as the method on the HTTP request's request line, as specified in [RFC2616] section 5.1. This technique is often referred to as "verb tunneling".

This header is valid only when on POST requests. A server MAY<57> support verb tunneling as defined in the preceding paragraph. If a server implementing this document does not support verb tunneling, it MUST ignore an X-HTTP-Method header, if present in a POST request, and treat the request as a standard POST request. This implies that a client of such a data service has to determine in advance (using server documentation and so on) if a given data service endpoint supports verb tunneling. A tunneled request sent to a service that does not support verb tunneling interprets the request as an insert request since POST requests map to an insert request, as specified in [RFC5023].

The syntax of the X-HTTP-Method is defined as follows:

```
XHTTPMethod  =   "X-HTTP-Method: "
                 ("PUT"
                 / "MERGE"
                 / "PATCH"
                 / "DELETE")
                 CRLF
```

For example, the HTTP request in the following Delete Request Tunneled in a POST Request listing instructs the server to delete the **EntityType** instance identified by **EntityKey** value 5 in the Categories **EntitySet** instead of performing an insert operation.

```
POST /Categories(5)
HTTP/1.1
Host: server
X-HTTP-Method: DELETE
```

Listing: Delete Request Tunneled in a POST Request

2.2.5.9 Prefer

A Prefer header is included in a request to state the client's preferred, but not required, server behavior (that is, a hint to the server). The Prefer header MAY be included on any request type (within a standalone or batch request), and a server MAY honor the header for HTTP POST, PUT, PATCH, and MERGE requests. A Prefer header with a value of "return-content" MUST NOT be specified on a DELETE request, a batch request as a whole, or a PUT request to update a named stream.

The syntax of the Prefer header is defined as follows:

```
Prefer       =  "Prefer" ":" preference

preference   =  "return-no-content" |
                "return-content" |
                preference-extension

preference-extension =  prefer-params *( ";" prefer-params ])
prefer-params =  token [ "=" ( token | quoted-string )
```

A header value of "return-no-content" indicates that the client prefers that the server not include an entity representing the current state of the resource in the response to a successful request. If the value is "return-no-content", the server MAY choose to return an empty response payload with status code 204.

A header value of "return-content" indicates that the client prefers that the server include an entity that represents the current state of the resource in the response to a successful request. If the value is "return-content", the representation (as requested by the Accept header) of the current state of the inserted or updated resource MAY be returned in the body of the response. If it is included, the resource MUST be formatted as if it was being retrieved with a GET operation. However, in the case of a PUT request to a media stream, the response is formatted as if it is the response to a successful POST operation with the specified content.

When the Prefer header is included with a request that uses POST tunneling, as specified in section 2.2.5.8, the preference is to be applied to the tunneled request.

Preference values of the prefer header other than return-content or return-no-content are not defined by this specification. If any of the preference values of the Prefer header are not recognized by the service, those values of the Prefer header MUST be ignored by the server and the server MUST continue processing the request as if the Prefer header was not specified.

2.2.5.10 Preference-Applied

When a Prefer (section 2.2.5.9) header value is successfully honored by the server, it MAY include a Preference-Applied response header that states which preference values were honored by the server. The grammar of the Preference-Applied response header is shown below.

The syntax of the Preference-Applied response header is defined as follows:

```
Prefer       =  "Preference-Applied" ":" preference
```

2.2.5.11 DataServiceId

The DataServiceId response header is returned by the server when the response payload for an InsertEntity request (section 2.2.7.1.1) or an InsertMediaResource request (section 2.2.7.1.3) is empty. The value of the header is the identifier of the entity that was acted on by the request. The identifier, in this case, is the same identifier that would have been returned in the response payload (for example, as the value of the **atom:id** element for Atom responses).

The DataServiceId response header is defined by the following grammar.

```
DataServiceId       =  "DataServiceId" ":" ( token )
```

The content of the DataServiceId header MUST be an IRI as defined by [RFC3987].

The DataServiceId header is only meaningful in a response to an InsertEntity request or an InsertMediaResource request with a 204-No Content response. If it is included on any other response or request, the DataServiceId header MUST be ignored.

2.2.6 Common Payload Syntax

The OData protocol that is defined in this document enables clients and servers to perform actions (for example, CRUD operations) on entities in an Entity Data Model (EDM), as specified in [MC-CSDL], that is represented by using one of multiple possible formats (AtomPub, JSON, Verbose JSON, and so on). Each serialization format or representation of an entity can be used in the payload of request and response messages, as specified in Request Types (section 2.2.7).

AtomPub Format (section 2.2.6.2) specifies how to represent EDM constructs (single EntityType instance, multiple **EntityType** instances in an EntitySet, NavigationProperties, and so on) by using the AtomPub [RFC5023] format.

[MS-ODATAJSON] and Verbose JSON Format (section 2.2.6.3) each specify how to represent EDM constructs (single **EntityType** instance, multiple **EntityType** instances in an **EntitySet**, NavigationProperty properties, and so on) by using the JavaScript Object Notation (JSON) [RFC4627] format: see [MS-ODATAJSON] for the preferred OData 3.0 representation and section 2.2.6.3 for the Verbose JSON legacy representation.

Note that Request Types (section 2.2.7) defines additional payload syntax directives, dependent on the message context, that MUST be adhered to in addition to those defined in this section.

2.2.6.1 Common Serialization Rules for XML-Based Formats

AtomPub and custom XML formatted payloads that represent Entity Data Model (EDM) constructs, as specified in [MC-CSDL] and defined in AtomPub Format (section 2.2.6.2) and XML Format (section 2.2.6.5), consist of XML elements and attributes in the XML Namespaces, as specified in [XMLNS], that are shown in the following table. All XML elements and attributes associated with the protocol that is defined in this document and custom XML formats that hold data are defined in the "data service" namespace.

All metadata-related elements that are defined in this document are defined in the "data service metadata" namespace.

Namespace	Namespace URI
Atom 1.0 Namespace (Atom only)	`http://www.w3.org/2005/Atom` The common namespace prefix for this namespace is "atom". Subsequent sections of this document refer to elements and attributes in this namespace by using the notation "atom:*elementName*".
Data Service Namespace	`http://schemas.microsoft.com/ado/2007/08/dataservices` This namespace URI can be changed to something more applicable to the particular service. The namespace URI preceding SHOULD be used if a data service does not wish to use an alternate. Servers are not required to use the namespace prefix "d" for this namespace. Subsequent sections of this document refer to elements in this namespace by using the notation "d:*elementName*".
Data Services Metadata Namespace	`http://schemas.microsoft.com/ado/2007/08/dataservices/metadata` Servers are not required to use the namespace prefix 'm' for this namespace. Subsequent sections of this document refer to elements in this namespace by using the notation "m:*elementName*".

Table: Protocol Namespace Definitions

XML payloads defined in this document can use the **xml:base** [XML-BASE] attribute, as specified in [RFC5023]. A data service and its client MUST understand and appropriately process this directive.

All EDM primitive types represented as XML element values MUST be formatted as defined by the rules in the following EDM Primitive Type Formats for Element Values table. In addition to the rules stated in the table, if the value of a primitive or **ComplexType** type is null, then the value of the associated XML element MUST be empty. In addition, an **m:null** attribute with value set to "true" MUST be present on the containing element.

EDM primitive type	ABNF rule for primitive type representation in XML-based payloads	Serialization format (ABNF grammar)
Edm.Binary	binary	binary = <Base64 encoded byte stream. See [RFC4648] for further details>
Edm.Boolean	booleanLiteral	See booleanLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Byte	byteLiteral	See byteLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.DateTime	dateTimeLiteral	See dateTimeLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Decimal	decimalLiteral	See decimalLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Double	doubleLiteral	See doubleLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Single	singleLiteral	See singleLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).

EDM primitive type	ABNF rule for primitive type representation in XML-based payloads	Serialization format (ABNF grammar)
Edm.Guid	guidLiteral	See guidLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Int16	int16Literal	See int16Literal in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Int32	int32Literal	See int32Literal in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Int64	int64Literal	See int64Literal in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.SByte	sbyteliteral	See sbyteliteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.String	stringUriLiteral	See stringUriLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Time	timeLiteral	See timeLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.DateTimeOffset	dateTimeOffsetLiteral	See dateTimeOffsetLiteral in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.Geography	N/A	N/A
Edm.GeographyPoint	pointGmlLiteral	pointGmlLiteral = <See [OGC-SFOLECOM] representation for a Point>
Edm.GeographyLineString	lineStringGmlLiteral	lineStringGmlLiteral = <See [OGC-SFOLECOM] representation for a LineString>
Edm.GeographyPolygon	polygonGmlLiteral	polygonGmlLiteral = <See [OGC-SFOLECOM] representation for a Polygon, except as modified below> In Ellipsoidal coordinates, all rings are equally interpretable as "outer". Therefore, the rings has to have their control points in left-foot winding order. This means that the points to the left side of the ring, when traversing in serialized order, are in the polygon, while those to the right side are not. In planar coordinates, where "outer" is well defined, the first ring has to be the outer ring, in accordance with the GML standard.
Edm.GeographyCollection	geoCollectionGmlLiteral	geoCollectionGmlLiteral = <See [OGC-SFOLECOM] representation for a GeometryCollection>
Edm.GeographyMultiPoint	multiPointGmlLiteral	multiPointGmlLiteral = <See [OGC-

EDM primitive type	ABNF rule for primitive type representation in XML-based payloads	Serialization format (ABNF grammar)
		SFOLECOM] representation for a MultiPoint>
Edm.GeographyMultiLineString	multiLineStringGmlLiteral	multiLineStringGmlLiteral = <See [OGC-SFOLECOM] representation for a MultiLineString>
Edm.GeographyMultiPolygon	multiPolygonGmlLiteral	multiPolygonGmlLiteral = <See [OGC-SFOLECOM] representation for a MultiPolygon, as modified below> In Ellipsoidal coordinates, all rings are equally interpretable as "outer". Therefore, the rings have to have their control points in left-foot winding order. This means that the points to the left side of the ring, when traversing in serialized order, are in the polygon, while those to the right side are not. In planar coordinates, where "outer" is well defined, the first ring has to be the outer ring, in accordance with the GML standard.
Edm.Geometry	N/A	N/A
Edm.GeometryPoint	pointGmlLiteral	N/A
Edm.GeometryLineString	lineStringGmlLiteral	N/A
Edm.GeometryPolygon	polygonGmlLiteral	N/A
Edm.GeometryCollection	geoCollectionGmlLiteral	N/A
Edm.GeometryMultiPoint	multiPointGmlLiteral	N/A
Edm.GeometryMultiLineString	multiLineStringGmlLiteral	N/A
Edm.GeometryMultiPolygon	multiPolygonGmlLiteral	N/A

Table: EDM Primitive Type Formats for XML Element Values

2.2.6.2 AtomPub Format

Atom is an XML-based document format described in [RFC4287] and extended with AtomPub specific extensions in [RFC5023]. This section uses the term AtomPub as shorthand to refer to the union of the document format rules that are defined in [RFC4287] and [RFC5023].

AtomPub describes lists of related information (a collection in the abstract AtomPub Protocol Model [RFC5023] section 4.2) known as "feeds". Feeds are composed of a number of items, known as "entries" (Entry Resources in the abstract AtomPub Protocol Model [RFC5023] section 4.2), each with an extensible set of attached metadata. For example, each entry has a title.

The following subsections define the mapping of constructs in the Entity Data Model (EDM) to Atom format elements for use in request/response messages as specified in Request Types (section 2.2.7). In all the subsections that follow, if a data model construct is not explicitly described, an associated Atom-based representation is not defined by this document. For such constructs, servers and clients MAY<58> either:

- Define their own representations and include them in a request or response if valid according to [RFC5023].

- Exclude them from requests and responses.

The examples in this section use the sample data model that is defined in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

2.2.6.2.1 Entity Set (as an Atom Feed Element)

An EntitySet or collection of entities is represented as an **atom:feed** element, as specified in [RFC4287] section 4.1.1. This section adds constraints to the formatting rules defined in AtomPub for **atom:feed** elements.

An Atom-formatted **EntitySet** or collection of entities has to adhere to the rules that are defined in this section.

atom:feedElement

The **atom:feed** element is specified in [RFC4287] section 4.1.1

atom:feedSubelements

The **atom:id** element, as specified in [RFC4287] section 4.2.6, MUST contain the URI that identifies the **EntitySet** that is represented by the parent **atom:feed** element. For example, assuming the parent element represented the Customers **EntitySet** (as described in Appendix A: Sample Entity Data Model and CSDL Document (section 6)), the value of this element would be `http://host/service.svc/Customers`.

The **atom:title** element, as specified in [RFC4287] section 4.2.14, MAY contain the name of the **EntitySet** that is represented by the parent **atom:feed** element. The set name MAY be qualified with the name of the Entity Data Model (EDM) namespace in which it is defined, as specified in [MC-CSDL]. If the URI in the sibling **atom:id** element is of the same form as URI6, as defined in Resource Path: Semantics (section 2.2.3.5) (last path segment is a **NavigationProperty**) and the **NavigationProperty** identifies an **EntitySet**, then the **atom:title** element MAY contain the name of the **NavigationProperty** instead of the name of the **EntitySet** that is identified by the property.

An **atom:link** element, as specified in [RFC4287] section 4.2.7, with a **rel="self"** attribute MUST contain an **href** attribute with a value equal to the URI used to identify the set that the parent **atom:feed** element represents. When used in HTTP responses, this URI MUST be equal to the associated HTTP request URI. When used in HTTP deep insert requests, this URI MUST identify a related collection of entities (identified by a **NavigationProperty** on the base EntityType of the **EntitySet** that is identified by the request URI) into which the deep/related new entities will be inserted, as specified in InsertEntity Request (section 2.2.7.1.1).

An **atom:entry** element, as specified in [RFC4287] section 4.1.2, within the **atom:feed** element, is formatted as specified in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

In response payloads only, if the server does not include an **atom:entry** element as a child element of the **atom:feed** element for every entity in the collection of entities that is identified by the associated URI, the **atom:feed** element represents a partial collection as defined in AtomPub [RFC5023] section 10.1. The **href** attribute of the **atom:linkrel="next"** element that is mandated by AtomPub [RFC5023] section 10.1 for such partial representations MUST have a value equal to the URI that identifies the next partial set of entities from the originally identified complete set. Such a URI SHOULD include a Skip Token system query option (section 2.2.3.6.1.9) to indicate that the URI addresses the next (after the partial set represented by the parent **atom:feed** element) partial set of entities.

Note The inclusion of an **atom:linkrel="next"** element in a response payload has protocol versioning implications as described in Executing a Received RetrieveValue Request (section 3.2.5.4.2).

2.2.6.2.1.1 InlineCount Representation (for Collections of Entities)

Applies to the OData 2.0 and OData 3.0 protocols

This section defines an extended representation of a collection of entities from that described in section 2.2.6.2.1. This representation is supported only in the OData 2.0 and OData 3.0 protocols.

A request URI MAY contain an **$inlinecount** system query option to indicate that the count of the number of entities represented by the query after filters have been applied and before applying any other query option processing MUST be included in the result sent by the data service.

The count value included in the result MUST be enclosed in an **m:count** element. The **m:count** element MUST be a direct child element of the feed element and occur before any **atom:entry** elements in the feed.

The **m:count** element MUST NOT be present in any feed elements nested within any **atom:entry** elements.

For example, the count of all Customer entities using the Customer **EntityType** instance described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in Atom as described in the following feed. In the example, the request included the InlineCount system query option and the Top system query option with a value of 1.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://sburges-devpc/FFEdmx/ffedmx.svc/"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Customers</title>
  <id>http://host/service.svc/Customers</id>
  <updated>2009-03-27T23:41:29Z</updated>
  <link rel="self" title="Customers" href="Customers" />
  <m:count>91</m:count>
  <entry>
    <id> http://host/service.svc/Customers('ALFKI')</id>
    <title type="text"></title>
    <updated>2009-03-27T23:41:29Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Customers" href="Customers('ALFKI')" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
      type="application/atom+xml;type=feed" title="Orders"
      href="Customers('ALFKI')/Orders" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Orders"
      type="application/xml" title="Orders"
      href="Customers('ALFKI')/$links/Orders" />    <link
       rel="http://schemas.microsoft.com/ado/2007/08/
       dataservices/related/CustomerDemographics" type="application/atom+xml;type=feed"
       title="CustomerDemographics" href="Customers('ALFKI')/CustomerDemographics" />
    <link
       rel="http://schemas.microsoft.com/ado/2007/08/
       dataservices/relatedlinks/CustomerDemographics" type="application/xml "
       title="CustomerDemographics" href="Customers('ALFKI')/$links/CustomerDemographics" />
    <category term="NorthwindModel.Customers"
       scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:CustomerID>ALFKI</d:CustomerID>
        <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
        <d:ContactName>Maria Anders</d:ContactName>
        <d:ContactTitle>Sales Representative</d:ContactTitle>
        <d:Address>Obere Str. 578</d:Address>
        <d:City>Toronto</d:City>
        <d:Region m:null="true" />
        <d:PostalCode>12209</d:PostalCode>
```

```
          <d:Country>Germany</d:Country>
          <d:Phone>030-0074321</d:Phone>
          <d:Fax>030-0076545</d:Fax>
        </m:properties>
      </content>
    </entry>
  </feed>
```

Listing: OData 3.0 Atom-Formatted InLineCount Representation

2.2.6.2.1.2 Entity Set (as an Atom Feed Element) with Actions

Applies to the OData 3.0 protocol

In OData 3.0, it is possible to advertise the availability of actions (section 2.2.1.3) that are bound to the definition of the feed, in the feed.

Actions that are advertised by the server MUST be encoded in an **m:action** element under the **atom:feed** element that corresponds to the feed that the action is bound to. There can be any number of actions bound to the feed, and therefore, there can be an arbitrary number of **m:action** elements.

The **m:action** element MUST have a metadata attribute that contains the Action Metadata URL (section 2.2.1.3.1) of the action.

The **m:action** element MUST have a target attribute containing a URL. This is the URL to which clients issue an Invoke Action request (section 2.2.7.5.1) to invoke the action. The binding parameter is assumed to be bound to the encompassing feed definition. Therefore, if the client invokes the action via the target URL, the target URL cannot include a value for the binding parameter in the request body.

The **m:action** element MUST have a title attribute that contains a simple string that is used as a simple but not necessarily unique name for the action. Generally, servers SHOULD specify a title that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

Actions advertised in the Atom feed element MUST be interpreted as being bound to the definition of the feed and not to the items that are represented in the feed.

Actions that operate on a feed MUST only be advertised in an Atom feed element if the server can fully encode the action, the resource path, and the appropriate system query options that define the feed.

System query options (section 2.2.3.6.1) that change the membership of the feed MUST be considered to be part of the feed definition. In practice, this means that the target URL that is used to invoke the action MUST encode the following system query options if they are used to define the feed:

$filter (section 2.2.3.6.1.4)

$expand (section 2.2.3.6.1.3)

$orderby (section 2.2.3.6.1.6)

$skip (section 2.2.3.6.1.7)

$top (section 2.2.3.6.1.8)

The remaining system query options, generally, do not define the feed and, therefore, do not need to be encoded in the target of the action:

$format (section 2.2.3.6.1.5)

$skiptoken (section 2.2.3.6.1.9)

$inlinecount (section 2.2.3.6.1.10)

$select (section 2.2.3.6.1.11)

2.2.6.2.1.3 Entity Set (as an Atom Feed Element) with Functions

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, it is possible to advertise the availability of functions (section 2.2.1.4) that are bindable to the definition of the feed, in the feed.

Functions that are advertised by the server MUST be encoded in an **m:function** element under the **atom:feed** element that corresponds to the feed that the function is bound to. There can be any number of functions bound to the feed, and therefore, there can be an arbitrary number of **m:function** elements.

The **m:function** element MUST have a metadata attribute that contains the function metadata URL (section 2.2.1.4.1). The function metadata URL MUST identify only functions that are bindable to the current feed definition. If overloads exist that cannot be bound to the current feed definition, individual **m:Function** elements SHOULD be returned that each have a function metadata URL that identifies a specific bindable overload.

The **m:function** element MUST have a target attribute that contains a URL. This is the URL to which clients issue an Invoke Function request (section 2.2.7.5.2) to invoke the function. The binding parameter is assumed to be bound to the encompassing feed definition. Therefore, if the client invokes the function via the target URL, it MUST not include a value for the binding parameter in the request via a parameter appended to the target URL.

The **m:function** element MUST have a title attribute that contains a simple string that is used as a simple but not necessarily unique name for the function. Generally, servers SHOULD specify a title that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

If a function to be advertised has overloads, the server SHOULD expose a single **m:function** element with a metadata attribute that identifies all the overloads.

Functions advertised in the Atom feed element MUST be interpreted as being bound to the definition of the feed and not to the items that are represented in the feed.

Functions that operate on a feed MUST only be advertised in an Atom feed element if the server can fully encode the function, the resource path, and the appropriate system query options that define the feed.

System query options (section 2.2.3.6.1) that change the membership of the feed MUST be considered part of the feed definition. In practice, this means that the target URL that is used to invoke the function MUST encode the following system query options if they are used to define the feed:

$filter (section 2.2.3.6.1.4)

$expand (section 2.2.3.6.1.3)

$orderby (section 2.2.3.6.1.6)

$skip (section 2.2.3.6.1.7)

$top (section 2.2.3.6.1.8)

The remaining system query options, generally, do not define the feed and, therefore, do not need to be encoded in the target of the function:

$format (section 2.2.3.6.1.5)

$skiptoken (section 2.2.3.6.1.9)

$inlinecount (section 2.2.3.6.1.10)

$select (section 2.2.3.6.1.11)

2.2.6.2.2 Entity Type (as an Atom Entry Element)

An EntityType instance MUST be represented as an **atom:entry** element, as specified in [RFC4287], section 4.1.2. This section adds additional constraints to the formatting rules defined in Atom for **atom:entry** elements.

An Atom-formatted **EntityType** instance MUST adhere to the rules defined in this section.

atom:entry element: This element is specified in [RFC4287] section 4.1.2

atom:entry child elements: If the entity represents an AtomPub Entry Resource [RFC5023] (section 4.2), the **atom:content** element MUST contain a "type" attribute with the value "application/xml". The **atom:content** element SHOULD<59> also contain one **m:properties** child element. The **m:properties** element MUST contain one child element for each **EDMSimpleType**, **ComplexType**, and **Collection** type property of the **EntityType** instance that is represented by the **atom:entry** element that is not otherwise mapped through a customizable feed property mapping in its Data Service Metadata Document (as defined in section 2.2.3.7.2.1). Child elements of the **m:properties** element that represent entity properties MUST have the same name as the property they represent, MUST belong to the data services namespace (section 2.2.6.1), and MAY have an **m:type** attribute to specify the EDM type of the property. If the **m:type** attribute is missing, the EDM type of the property MUST be assumed to be **Edm.String**. If the **EntityType** instance being represented was identified with a URI that includes a Select system query option (section 2.2.3.6.1.11), the prior rule is relaxed such that only the properties identified by the **$select** query option SHOULD be represented as child elements of the **m:properties** element. Each child element that represents a property MUST be defined in the data service namespace, as described in Common Serialization Rules for XML-Based Formats (section 2.2.6.1), and the element name has to be the same as the property it represents. The rules for how to represent an entity type as an **atom:entry** element that contains customizable feed property mappings are defined in Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping (section 2.2.6.2.2.1).

If the entity represents an AtomPub Media Link Entry, as specified in [RFC5023] (section 4.2), the **m:properties** element MUST also contain the **EDMSimpleType**, **ComplexType**, and, in the OData 3.0 protocol, the collection properties of the **EntityType** instance, as described in the preceding paragraph. However, the **m:properties** element MUST be a direct child of the **atom:entry** element (as opposed to the **atom:content** element). Additionally, as specified in [RFC5023], an **atom:link** element SHOULD be included, which contains a **rel="edit-media"** attribute. If such an **atom:link** element identifies a Media Resource with an associated concurrency token, the element SHOULD include an **m:etag** attribute with a value equal to the ETag of the media resource identified by the **atom:link** element.

An **atom:category** element containing a **term** attribute and a **scheme** attribute MUST be included if the **EntityType** of the **EntityType** instance represented by the **atom:entry** object is part of an inheritance hierarchy, as described in [MC-CSDL] (section 1). If the **EntityType** is not part of an inheritance hierarchy, the **atom:category** element SHOULD be included. The value of the **term** attribute MUST be the namespace qualified name of the **EntityType** of the instance represented by the **atom:entry** element. The value of the **scheme** attribute MUST be a data service specific IRI which, as specified in [RFC4287], identifies the categorization scheme used. If a data service does

not have a scheme IRI, it SHOULD use the URI shown in grammar rule dataServiceSchemeURI in the Entity Type Atom Representation URIs (ABNF Grammar) listing that follows in this section.

An **m:etag** attribute MAY be included on the **entry** element representing the **EntityType** instance. In this context, the "m" prefix refers to the data service metadata namespace defined in Common Serialization Rules for XML-Based Formats (section 2.2.6.1). When included, it MUST represent the concurrency token associated with the **EntityType** instance, as defined in ETag (section 2.2.5.4), and MUST be used instead of the ETag HTTP header defined in ETag (section 2.2.5.4), which, according to [RFC2616], is used to represent a single entity when multiple entities are present in a single payload.

An **atom:link** element SHOULD be included, which contains a **rel="edit"** or **rel="self"** attribute. The **rel** attribute MAY<60> be used to indicate that a resource is read-only (when the value of the attribute is "self") or read-write (when the attribute's value is "edit"). If such an **atom:link** element is included, it MUST have an **href** attribute whose value is a URI that identifies the entity represented by the **atom:entry** element.

In responses to retrieve requests, as specified in RetrieveEntity Request (section 2.2.7.2.2), servers MUST represent the URI that represents a related entity or collection of entities (NavigationProperty) of the **EntityType** as an **atom:link** element that is a child element of the **atom:entry** element. Each **atom:link** element MUST contain a **rel** attribute with the value defined by the relNavigationlLinkURI rule shown in the following grammar, as defined in the listing that follows. The element SHOULD also contain a **title** attribute with the value equal to the **NavigationProperty** name and MUST contain an **href** attribute with value equal to the URI which identifies the **NavigationProperty** on the **EntityType**.

Note Atom also requires a **type** attribute, which MUST have a value of "application/atom+xml;type=entry" when the **NavigationProperty** identifies a single entity instance and "application/atom+xml;type=feed" when the property identifies an EntitySet.

```
dataServiceNs       = "http://schemas.microsoft.com/ado/2007/08/dataservices"
                      / <Server specified "Data Service namespace" URI>
                         ; see section 2.2.6.1

dataServiceSchemeURI= "http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
relNavigationLinkURI=  dataServiceNs
                        "/related/"
                        ; line below represents the name of the Navigation Property
                        ; (not type qualified) being represented by the current
                        ; atom:link element
                        entityNavProperty         ; section 2.2.3.1
```

In OData 3.0, responses to retrieve requests, as specified in RetrieveEntity Request (section 2.2.7.2.2), MAY represent the URI that identifies the association between the entity represented by the response payload and a related entity (or collection of entities) as an **atom:link** element that is a child element of the **atom:entry** element. If present, each **atom:link** element MUST contain a **rel** attribute with the value defined by the relAssociationlLinkURI rule shown in the grammar defined in the listing that follows. The element MUST also contain an **href** attribute with a value equal to the URI that identifies the association represented by the **atom:link** element (that is, the **NavigationProperty** on the **EntityType**).

Note Atom also requires an **atom:type** attribute, which MUST have a value of "application/xml".

```
relAssociationLinkURI=  dataServiceNs
                        "/relatedlinks/"
                        ; line below represents the name of the Navigation Property
                        ; (not type qualified) being represented by the current
                        ; atom:link element
                        entityNavProperty         ; section 2.2.3.1
```

Listing: URI of the atom:link rel attribute for Navigation Properties of an Entity Type (ABNF Grammar)

For example, the Customer **EntityType** instance described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in Atom as described in the following listing.

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:gml="http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/Orders" />

   <link
       rel="http://schemas.microsoft.com/ado/2007/08/
       dataservices/relatedlinks/Orders" type="application/xml"
       title="Orders"
       href="Customers('ALFKI')/$links/Orders" />
  <content type="application/xml">
    <m:properties>
      <d:CustomerID>ALFKI</d:CustomerID>
      <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
      <d:Address>
        <d:Street>57 Contoso St</d:Street>
        <d:City>Seattle</d:City>
        <d:Location m:type="Edm.GeographyPoint"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
      </d:Address>
      <d:EmailAddresses m:type="Collection(Edm.String)">
        <d:element>altaddress1@company.com</d:element>
        <d:element>altaddress2@company.com</d:element>
      </d:EmailAddresses>
      <d:AlternateAddresses m:type="Collection(SampleModel.Address)">
        <d:element m:type="SampleModel.EAddress">
          <d:Street>123 contoso street</d:Street>
        </d:element>
        <d:element>
          <d:Street>834 1st street</d:Street>
          <d:Apartment>102</d:Apartment>
        </d:element>
      </d:AlternateAddresses>
      <d:Version>AAAAAAAA+gE=</d:Version>
    </m:properties>
  </content>
</entry>
```

Listing: OData 3.0 Atom-Formatted Customer Entity

2.2.6.2.2.1 Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping

If the **EntityType** instance that is represented includes customizable feed annotations in the data services metadata document, the properties with custom mappings are represented as directed by the mappings information specified in Conceptual Schema Definition Language Document Extensions for Customizable Feeds (section 2.2.3.7.2.1). Properties that do not have customizable feed mappings defined are represented according to the previous section, Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

In the OData 2.0 protocol, if the property of an **EntityType** instance in a data service response includes customizable feed annotations in the data services metadata document and has a value of null, the element or attribute being mapped to MAY be present and MUST be empty.

For example, the Employee **EntityType** instance that is described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in Atom as described in the following listing.

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:m="http://schemas.microsoft.com/ado/2008/11/dataservices/metadata"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns="http://www.w3.org/2005/Atom" m:etag="W/"X'000000000000FA01'""">
  <category term="SampleModel.Employee"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Employees(1)</id>
  <title type="text">Eric Gruber</title>
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Employees" href="Employees(1)" />
  <content type="application/xml">
   <m:properties>
    <d:EmployeeID>ALFKI</d:EmployeeID>
    <d:Address>
      <d:Street>4567 Main Street<d:Street>
      <d:City>Seattle</d:City>
      <d:Location m:type="Edm.GeographyPoint"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
    </d:Address>
    <d:Version>BBBBBBBB+gE=</d:Version>
   </m:properties>
  </content>
  <emp:Location xmlns:emp="http://www.microsoft.com">Seattle</emp:Location>
</entry>
```

Listing: Atom-formatted Customer Entity with a Property Mapping

2.2.6.2.2.2 Entity Type (as an Atom Entry Element) with Actions

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, it is possible to advertise the availability of actions (section 2.2.1.3) that are bindable to an entity.

For the server to advertise actions that can be bound to the current entity, the current entity MUST be encoded in a **m:action** element under the **atom:entry** element that corresponds to the entity that the action is bound to. Any number of actions can be bound to the entity, and therefore, there can be an arbitrary number of **m:action** elements.

The **m:action** element MUST have a metadata attribute that contains the action metadata URL (section 2.2.1.3.1) of the action.

The **m:action** element MUST have a target attribute containing a URL. This is the URL to which clients issue an Invoke Action request (section 2.2.7.5.1) to invoke the action. The binding parameter is assumed to be bound to the encompassing **EntityType**, and therefore, if the client invokes the action via the target URL, it MUST not include a value for the binding parameter in the request body.

The **m:action** element MUST have a title attribute containing a simple string used as a simple but not necessarily unique name for the action. Generally, servers SHOULD specify a title that is easily understood by the user because the title is likely to be used by clients to display options to an end user.

Actions can also be selectively requested or omitted by using different Select system query option (section 2.2.3.6.1.11) combinations.

If the Atom Entry Element is retrieved as part of a feed (section 2.2.6.2.1) and it is expensive to identify whether an action requested, either implicitly or explicitly, by using Select system query option (section 2.2.3.6.1.11) can be bound to a particular entity, a server SHOULD advertise the action and fail later if the action is invoked and found to be unavailable.

2.2.6.2.2.3 Entity Type (as an Atom Entry Element) with Functions

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, it is possible to advertise the availability of functions (section 2.2.1.4) that are bindable to an entity.

Functions that the server advertises MUST be encoded in an **m:function** element under the **atom:entry** element that corresponds to the entity that the function is bound to. Any number of functions can be bound to the entity, and therefore, there can be an arbitrary number of **m:function** elements.

The **m:function** element MUST have a metadata attribute that contains the Function Metadata URL (section 2.2.1.4.1). The function metadata URL MUST identify only functions that are bindable to the current **EntityType**. If overloads exist that cannot be bound to the current **EntityType**, individual **m:Function** elements SHOULD be returned that each have a function metadata URL that identifies a specific bindable overload.

The **m:function** element MUST have a target attribute that contains a URL. This is the URL to which clients issue an Invoke Function request (section 2.2.7.5.2) if they want to invoke the function. The binding parameter is assumed to be bound to the encompassing **EntityType**. Therefore, if the client invokes the function via the target URL, it MUST not include a value for the binding parameter in the request via a parameter that is appended to the target URL.

The **m:function** element MUST have a title attribute that contains a simple string that is used as a simple but not necessarily unique name for the action. Generally, servers SHOULD specify a title that is easily understood by the user because the title is likely to be used by clients to display options to an end user.

If a function to be advertised has overloads, the server SHOULD if possible expose a single **m:function** element with a metadata attribute that identifies all the overloads bindable to the current entity.

Functions and function groups can also be selectively requested or omitted by using different Select system query option (section 2.2.3.6.1.11) combinations. If the Atom Entry Element is retrieved as part of a feed (section 2.2.6.2.1) and it is expensive to identify whether a function requested explicitly by using a Select system query option (section 2.2.3.6.1.11) is available, a server SHOULD advertise the requested function and fail later if the function is invoked and found to be unavailable.

2.2.6.2.3 Complex Type

A **ComplexType** property on an **EntityType** MUST be serialized within the **m:properties** element of an **atom:content** element or **atom:entry** element, as specified in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

Each declared property defined on a **ComplexType** MUST be represented as a child element (in the data service namespace defined in Common Serialization Rules for XML-based Formats (section 2.2.6.1)) of the element representing the **ComplexType** as a whole, and MAY have an **m:type** attribute to specify the EDM type of the property. If the **m:type** attribute is missing, the EDM type of the property MUST be assumed to be **Edm.String**.

An Atom representation of a **ComplexType** outside the context of an **atom:entry** element as described in the preceding paragraph is not defined by this document. See Complex Type (section 2.2.6.5.1) for details regarding formatting a **ComplexType** by using XML independent from the content of the defining **EntityType**.

2.2.6.2.4 Navigation Property

See Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) and Deferred Content (section 2.2.6.2.6) for the description of using the **atom:link** element to represent:

- A URI that identifies a related entity or collection of entities.
- A URI that identifies the association to a related entity or collection of entities.

2.2.6.2.5 EDMSimpleType Property

For a description of how properties are serialized in request/response payloads that represent an **EntityType** instance, see Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

An Atom representation of properties outside of the context of an **atom:entry** element is not defined. See EDMSimpleType Property (section 2.2.6.5.3) for details regarding formatting an **EDMSimpleType** property using XML independent from the content of the defining **EntityType**.

2.2.6.2.6 Deferred Content

The serialized representation of an entity and its related entities, identified by NavigationProperties, can be large. For resource conservation purposes (bandwidth, CPU, and so on), a data service generally does not want to return the full graph of entities related to the EntityType instance or set identified in a request URI. For example, a data service ought to defer sending entities represented by any navigation property in a response unless explicitly asked to send those entities via the **$expand** system query option, as described in Expand System Query Option ($expand) (section 2.2.3.6.1.3).

Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) specifies Atom-formatted **EntityType** instances which MUST contain **atom:link** elements for each **NavigationProperty** on the **EntityType**. When these **atom:link** elements are empty, they signify deferred **NavigationProperty** content (for example, the entities represented by the **NavigationProperty** are not serialized inline). For example, using the two **EntityTypes** Customer and Order, as specified in Appendix A: Sample Entity Data Model and CSDL Document (section 6), the default Atom serialization of the Customer instance with EntityKey value of "ALFKI" is shown with deferred **NavigationProperty** content in the Atom-formatted Customer entity listing in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

In the example, the presence of the empty **atom:link** element with **rel** attribute whose value is `http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders` signifies that the value of the Order's **NavigationProperty** is deferred (this is not directly represented in this serialization). In order to retrieve the deferred value(s), a client can make a separate request to the navigation property `URIservice.svc/Customers('ALFKI')/Orders` or explicitly ask that the property be loaded inline via the **$expand** system query option, as described in Expand System Query Option ($expand) (section 2.2.3.6.1.3).

2.2.6.2.6.1 Inline Representation

A request URI can include the **$expand** system query option to explicitly request that the entity or entities represented by a NavigationProperties property be serialized inline (rather than deferred), as described in Expand System Query Option ($expand) (section 2.2.3.6.1.3). The example that follows uses the same data model as the Deferred Content example referenced previously; however, this example shows the value of the Order's **NavigationProperty** serialized inline.

A **NavigationProperty** that represents an EntityType instance or a group of entities and that is serialized inline MUST be placed within a single **m:inline** element that is a child element of the **atom:link** element representing the **NavigationProperty**. Since a **NavigationProperty** identifies a collection of entities or a single entity, the contents of the **m:inline** element will be described in Entity Set (as an Atom Feed Element) (section 2.2.6.2.1) or Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

If **NavigationProperty** represents an **EntityType** instance and that instance is null, an empty **m:inline** element MUST appear under the **atom:link** element that represents **NavigationProperty**. If **NavigationProperty** represents a collection of entities and the collection is empty, an **m:inline** element with a nested **atom:Feed** element with no **atom:Entry** subelements MUST appear under the **atom:link** element that represents **NavigationProperty**. In both cases, the presence of the **m:inline** element indicates that **NavigationProperty** has been expanded but that no content was associated with it.

```
<?xml version="1.0" encoding="utf-8"?>
<entry  xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')" />
  <link
       rel="http://schemas.microsoft.com/ado/2007/08/
       dataservices/relatedlinks/Orders" type="application/xml"
       title="Orders"
       href="Customers('ALFKI')/$links/Orders" />

  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/Orders">
    <m:inline>
      <feed>
        <title type="text">Orders</title>
        <id>http://host/service.svc/Customers('ALFKI')/Orders</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Orders" href="Customers('ALFKI')/Orders" />
        <entry>
          <category term="SampleModel.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Orders(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Orders" href="Orders(1)" />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
```

```
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(1)/Customer" />
            <link
                rel="http://schemas.microsoft.com/ado/2007/08/
                dataservices/relatedlinks/Customer" type="application/xml"
                title="Customer"
                href="Orders(1)/$links/Customer" />

            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/OrderLines" />
            <link
                rel="http://schemas.microsoft.com/ado/2007/08/
                dataservices/relatedlinks/OrderLines" type="application/xml"
                title="OrderLines"
                href="Orders(1)/$links/OrderLines" />
            <content type="application/xml">
              <d:OrderID m:type="Edm.Int32">1</d:OrderID>
              <d:ShippedDate m:type="Edm.DateTime">1997-08-25T00:00:00</d:ShippedDate>
            </content>
          </entry>
          <entry>
            <category term="SampleModel.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
            <id>http://host/service.svc/Orders(2)</id>
            <title type="text" />
            <updated>2008-03-30T21:52:45Z</updated>
            <author>
              <name />
            </author>
            <link rel="edit" title="Orders" href="Orders(2)" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(2)/Customer" />
            <link
                rel="http://schemas.microsoft.com/ado/2007/08/
                dataservices/relatedlinks/Customer" type="application/xml"
                title="Customer"
                href="Orders(2)/$links/Customer" />

            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/OrderLines" />
            <link
                rel="http://schemas.microsoft.com/ado/2007/08/
                dataservices/relatedlinks/OrderLines" type="application/xml"
                title="OrderLines"
                href="Orders(2)/$links/OrderLines" />

            <content type="application/xml">
              <d:OrderID m:type="Edm.Int32">2</d:OrderID>
              <d:ShippedDate m:type="Edm.DateTime">1997-10-03T00:00:00</d:ShippedDate>
            </content>
          </entry>
        </feed>
      </m:inline>
    </link>
    <content type="application/xml">
      <d:CustomerID>ALFKI</d:CustomerID>
      <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
      <d:Address>
        <d:Street>57 Contoso St</d:Street>
        <d:City>Seattle</d:City>
      </d:Address>
      <d:Version>AAAAAAAA+gE=</d:Version>
    </content>
```

```
</entry>
```

Listing: Version 3.0 Atom-formatted Customer Entity with the Order's NavigationProperty Value Formatted Inline

2.2.6.2.7 Service Document

Service Document (section 2.2.3.7.1) specifies that AtomPub, as specified in [RFC5023], defines a service document which describes collections of resources available from a data service. The root URL of a data service that implements the protocol defined in this document MUST identify such a service document. In this service document, a data service MUST represent all available collections in a single **app:workspace** element. See [RFC5023] section 8.3.2 for the definition of the **app:workspace** element and [RFC5023] section 6.1 for the definition of the "app" prefix. Within that workspace, a data service MUST represent each EntitySet in its associated Entity Data Model (EDM), as described in Abstract Data Model (section 2.2.1), as an **app:collection** element, as specified in [RFC5023] section 8.3.3. The URI identifying the **EntitySet** MUST be used as the value of the **href** attribute of the **app:collection** element. The name of the **EntitySet** MAY be used as the value of the **atom:title** element which, as specified in [RFC5023], is a mandatory child element of the **app:collection** element.

The following is an example AtomPub Service Document, as specified in [RFC5023], for the EDM described in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<service xml:base="http://localhost:2032/nw.svc/"
         xmlns:atom="http://www.w3.org/2005/Atom"
         xmlns:app="http://www.w3.org/2007/app"
         xmlns="http://www.w3.org/2007/app">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Customers">
      <atom:title>Customers</atom:title>
    </collection>
    <collection href="Orders">
      <atom:title>Orders</atom:title>
    </collection>
    <collection href="OrderLines">
      <atom:title>OrderLines</atom:title>
    </collection>
  </workspace>
</service>
```

Listing: AtomPub Service Document Describing a Data Service

2.2.6.2.8 Additional Representations

In AtomPub, as specified in [RFC5023], the structured unit of information is an Entry Resource that is represented as an **atom:entry** element and, as specified in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2), is used to represent EntityTypes. A standalone Atom-based representation of the constituent EDM constructs of an EDMSimpleType property is not defined by this document.

The URI-addressing scheme for data services, as defined in URI Format: Resource Addressing Rules (section 2.2.3), does enable addressing the constituent EDM constructs of an EntityType directly. For XML, JSON, and Verbose JSON serialization rules for such resources, see XML Format (section 2.2.6.5), [MS-ODATAJSON], and Verbose JSON Format (section 2.2.6.3), respectively.

2.2.6.2.9 Collection Property

A collection property on an **EntityType** that is not otherwise mapped through a customizable feed property mapping MUST be represented within the **m:properties** element of the **atom:entry** and MUST contain one child element for each collection property of the represented **EntityType** instance. Each child element representing a collection property MUST be defined in the data service namespace (http://schemas.microsoft.com/ado/2007/08/dataservices), and the element name must be the same as the property it represents, as described in Collection Property of Complex Type (section 2.2.6.2.9.1) and Collection of EDMSimple Type (section 2.2.6.5.7). An attribute named "type" (in the Data Service Metadata namespace), as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1), MAY exist on the element representing the collection as a whole.

2.2.6.2.9.1 Collection Property of Complex Type

A property of an **EntityType** that represents a collection of complex types (that is not otherwise mapped through a customizable feed property mapping) MUST be represented within the **m:properties** element of an **atom:content** element or **atom:entry** element, as specified in Collection Property (section 2.2.6.2.9). Each **ComplexType** instance in the collection MUST be represented as a child element of the element that represents the collection as a whole and be named "element" and MUST be defined in the data service namespace (http://schemas.microsoft.com/ado/2007/08/dataservices). Each property of the **ComplexType** instance MUST be represented in the same way as in the serialization of a single **ComplexType**, as described in section 2.2.6.2.3. All properties of the **ComplexType** MUST be serialized to the same parent "element". An attribute named "type" (in the Data Service Metadata namespace, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1)), MAY exist on the element representing the item in the collection.

A collection property of **ComplexType** MUST NOT contain null values of the **ComplexType**. If the collection property is empty, the element representing the collection property as a whole MUST NOT have any child elements.

2.2.6.2.9.2 Collection of EDMSimpleType

A property of an **EntityType** that represents a collection of primitive values (that is not otherwise mapped through a customizable feed property mapping) MUST be serialized within the **m:properties** element of an **atom:content** element or **atom:entry** element, as specified in Collection Property (section 2.2.6.2.9). Each value in the Collection MUST be represented as an element named "element", and "element" MUST be defined in the data service namespace as (http://schemas.microsoft.com/ado/2007/08/dataservices) and be a direct child element of the element representing the collection as a whole. Each **EDMSimpleType** in the collection MUST be represented as described in EDMSimpleType Property (section 2.2.6.2.5).

A collection property of **EDMSimpleType** MUST NOT contain null values of the **EDMSimpleType**. If the collection property is empty, the element representing the collection property as a whole MUST NOT have any child elements.

2.2.6.2.10 Named Resource Streams

An Entry Resource MAY include one or more named resource streams.

When a server responds to a client request to retrieve an Entry Resource that contains named resource streams, the server MUST include information about those named resource streams in the response. That metadata for each named resource stream instance MUST be represented inside atom **atom:link** elements.

The edit link for a named resource stream, if present, MUST be represented as a **link** element that has:

- The **rel** attribute set to http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/ with the name of the named resource stream appended and the **href** attribute set to the edit link for the named resource stream.
- The "**type** attribute set to the Content Type of the named resource stream.
- The **m:etag** attribute, if present, set to the ETag (section 2.2.5.4) value for the named resource stream.

There MAY also be a **link** element for the named resource stream's self link. If this element is present, it MUST have:

- The **rel** attribute set to http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/ with the name of the named resource stream appended.
- The **title** attribute set to the name of the named resource stream.
- The **href** attribute set to the self link for the named resource stream.
- The **type** attribute set to the Content Type of the named resource stream.
- The **m:etag** attribute, if present, set to the ETag value for the named resource stream.

If the edit link and self link are the same for the named resource stream instance, the server MAY optimize payload size by excluding the self link **atom:link** element. In the absence of a self link **atom:link** element for the named resource stream, clients MUST assume that the self link and edit link are the same and that the href of the edit link **atom:link** element MAY be used instead.

Similarly, the server MAY expose the named resource stream instance as read-only by including only the self link. This would retrieve the stream and omit the edit link.

For example, the entry in the following listing indicates that:

- The "Thumbnail" named resource stream might be retrieved from 'http://server/Thumbnail564.jpeg'.
- The "Thumbnail" named resource stream might be updated at "http://server/uploads/Thumbnail564.jpeg".
- The "Thumbnail" named resource stream has a Content Type of "image/jpeg".
- The "PrintReady" named resource stream might be both retrieved and updated at the same URI, namely, "Photos(1)/PrintReady/$value".

```
<entry>
          <category term="SampleModel.Photo"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Photos(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Photos" href="Photos(1)" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/Thumbnail"
            type="image/jpeg"
            title="Thumbnail"
            href="http://server/Thumbnail546.jpeg" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/Thumbnail"
            type="image/jpeg"
            title="Thumbnail"
```

```
            href=" http://server/uploads/Thumbnail546.jpeg"
            m:etag="####" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/PrintReady"
            type="image/png"
            title="PrintReady"
            href="Photos(1)/PrintReady " />
          <content type="application/xml">
            <d:ID m:type="Edm.Int32">1</d:ID>
            <d:Name m:type="Edm.String">Mount Fuji</d:Name>
          </content>
</entry>
```

Listing: AtomPub Entry with Named Resource Stream links

2.2.6.2.11 Links and Subtypes

Entity instances in an **EntitySet** MAY not belong to the same **EntityType** but MUST all be derived from the **EntityType baseType** associated with the **EntitySet**. In addition to the rules described in Entity Set (as an Atom Feed Element (section 2.2.6.2.1), the following rules apply:

- The edit link for an **EntityType**, if present, MUST be represented as a **link** element that SHOULD have an **href** attribute that includes the namespace qualified **EntityType**. For example, http://host/service.svc/Customers/ ('ALFKI2')/ SampleModel.VipCustomer
- There MAY also be a **link** element for the entity's self link. If this element is present, it SHOULD have the **href** attribute set to the namespace-qualified **EntityType**. For example,http://host/service.svc/Customers/ ('ALFKI2')/ SampleModel.VipCustomer
- If an **EntityType** has a **NavigationProperty**, the following rules apply (in addition to those described in Deferred Content (section 2.2.6.2.6)):
  - The link for the **NavigationProperty** on the **EntityType**, if present, MUST be represented as a **link** element that SHOULD have an **href** attribute that includes the namespace qualified **EntityType** on which the **NavigationProperty** exists. For example,http://host/service.svc/Customers/ ('ALFKI2')/ SampleModel.VipCustomer/InHouseStaff
  - The association link that describes the relationship between the related entities MUST be represented as a **link** element that SHOULD have the **href** attribute that includes the namespace qualified **EntityType** on which the **NavigationProperty** exists. For example, http://host/service.svc/Customers/ ('ALFKI2')/ SampleModel.VipCustomer/$links/InHouseStaff
- If an **EntityType** has named resource streams, the following rules apply (in addition to those described in section Named Resource Streams (section 2.2.6.2.10)):
  - The **href** attribute of the, media-resource and edit-media links SHOULD include the namespace-qualified **EntityType** on which the named resource stream exists. For example, http://host/service.svc/Customers/ ('ALFKI2')/ SampleModel.VipCustomer/Logo

2.2.6.2.12 Annotations

In OData 3.0, just as annotations can be applied to metadata constructs in the service metadata document (section 2.2.3.7.2), annotations MAY also be applied to elements in an AtomPub payload. Annotations are represented as **annotation** elements that follow the rules detailed in this section.

Any AtomPub element that represents an Entity Data Model (EDM) construct (for more details, see [MC-CSDL] section 2.2.6.2 and its subsections) MAY be annotated. Parsers MAY choose to ignore annotations.

Rules for Instance Annotations: An instance annotation is represented as an XML element with the name "annotation" in the metadata namespace (see section 2.2.6.1). The **annotation** element MUST have a **term** attribute that specifies the fully qualified name of the term being applied.

If the type of the annotation value is anything other than **Edm.String**, a **type** attribute MUST be present with a value that is equal to the name of the type.

The value of the element that represents a value annotation whose type is a primitive value MUST be an EDM primitive value that is formatted as per section 2.2.6.1.

The value of the element the represents a value annotation whose type is a collection MUST be the individual elements of that collection formatted as direct child elements of the **annotation** element, as specified in Collection Property of Complex Type (section 2.2.6.2.9.1) and Collection of EDMSimple Type (section 2.2.6.5.7).

Each property of an annotation whose type is a **ComplexType** or **EntityType** MUST be represented as a direct child element of the **annotation** element. The name of the child element MUST be the name of the property and MUST be in the data service namespace (for more information, see Common Serialization Rules for XML-based Formats (section 2.2.6.1)). Further formatting rules that MUST be adhered to when representing properties within an annotation are the same as the rules for primitive and complex properties of an **EntityType** or **ComplexType** in an AtomPub payload, as defined in section 2.2.6.2.2 or 2.2.6.2.3, respectively, and their respective subsections.

target Attribute: The **annotation** element MAY have a target attribute to state the target element that is being annotated. Legal values for this annotation are '.' or the name of the target element that is being annotated.

If the value of the target attribute is '.' or if the target attribute is omitted, the target of the annotation is the model element that is represented by the direct parent element that the annotation is within.

If the value of the target attribute is the name of the target element that is being annotated, that element MUST be a sibling of the annotation element. If multiple possible target elements exist, the value of this attribute MUST be a namespace qualified (*namespacePrefix*.*targetElementName*) element name.

When annotating a property, the **annotation** element MUST be a direct child of the **properties** element that is defined in the metadata namespace, and the value of the target attribute MUST specify the property that is being annotated.

When annotating a navigation property, named stream, or other element that is represented by a **link** element, the **annotation** element MUST be a direct child of the **link** element that represents the navigation property.

When annotating an entity, the **annotation** element MUST be a direct child of the **entry** element that represents the entity.

When annotating a feed, the **annotation** element MUST be a direct child of the **feed** element.

When annotating an error, the **annotation** element MUST be a direct child of the **error** element and MAY target the code, message, or innererror, or the error itself if **target** is not specified or is specified with a value of ".".

Instance annotations are not supported when serializing single **EdmSimpleType** properties in XML, as described in 2.2.6.5.3.

The following example shows an annotated collection of the Customer Entities described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) with the following annotations.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

```
<feed xml:base="http://host/service.svc"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Customers</title>
  <id>http://host/service.svc/Customers</id>
  <updated>2009-03-27T23:41:29Z</updated>
  <link rel="self" title="Customers" href="Customers" />
  <!-- instance annotation targetting the collection of Customers -->
  <annotation term="com.contoso.customervocab.setkind">VIPs</annotation>
  <entry>
    <!-- instance annotation targetting the Entity Type instance -->
    <annotation term="com.contoso.customervocab.kind">VIP</annotation>
    <id> http://host/service.svc/Customers('ALFKI')</id>
    <title type="text"></title>
    <updated>2009-03-27T23:41:29Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Customers" href="Customers('ALFKI')" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
      type="application/atom+xml;type=feed" title="Orders"
      href="Customers('ALFKI')/Orders">
      <!-- instance annotation targetting the Orders navigation property -->
      <annotation term="com.contoso.purchaseordervocab.priority"
m:type="Edm.Int">1</annotation>
    </link>
    <!-- Note: navigation properties have been removed to limit the size of this example -->
    <content type="application/xml">
      <m:properties>
        <!-- instance of a structured annotation targetting the CompanyName property -->
        <annotation term="con.contoso.displayvocabulary.displayInfo m:target="CompanyName">
            <d:title m:type="Edm.Boolean">true</d:title>
            <d:order m:type="Edm.Int">1</d:order>
        </annotation>
        <d:CustomerID>ALFKI</d:CustomerID>
        <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
        <d:ContactName> Alfreds Futterkiste </d:ContactName>
        <d:ContactTitle>Sales Representative</d:ContactTitle>
        <d:Address>Obere Str. 578</d:Address>
        <d:City>Toronto</d:City>
        <d:Region m:null="true" />
        <d:PostalCode>12209</d:PostalCode>
        <d:Country>Germany</d:Country>
        <d:Phone>030-0074321</d:Phone>
        <d:Fax>030-0076545</d:Fax>
      </m:properties>
    </content>
  </entry>
</feed>
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xml:base="http://host/service.svc"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns:cust="http://contoso.com/customervocab"
 xmlns:display="http://contoso.com/displayvocab"
 xmlns:po="http://contoso.com/purchaseordervocab"
 xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Customers</title>
  <id>http://host/service.svc/Customers</id>
  <updated>2009-03-27T23:41:29Z</updated>
  <link rel="self" title="Customers" href="Customers" />
  <!-- instance  value annotation targetting the collection of Customers -->
  <cust:setkind m:target=".">VIPs</cust:setkind>
  <entry>
    <!-- instance value annotation targetting the Entity Type instance -->
    <cust:kind m:target="." m:type="Edm.String">VIP</cust:kind>
    <id> http://host/service.svc/Customers('ALFKI')</id>
    <title type="text"></title>
    <updated>2009-03-27T23:41:29Z</updated>
```

```
      <author>
        <name />
      </author>
      <link rel="edit" title="Customers" href="Customers('ALFKI')" />
      <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed" title="Orders"
        href="Customers('ALFKI')/Orders">
        <!-- instance value annotation targetting the Orders navigation property -->
        <po:priority m:target="." m:type="Edm.Int">1</po:priority>
      </link>
      <!-- Note: navigation properties have been removed to limit the size of this example -->
      <content type="application/xml">
        <m:properties>
          <!-- instance of a type annotation targetting the CompanyName property -->
          <display:display m:target="CompanyName">
              <display:title m:type="Edm.Boolean">true</display:title>
              <display:order m:type="Edm.Int">1</display:order>
          </display:display>
          <d:CustomerID>ALFKI</d:CustomerID>
          <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
          <d:ContactName> Alfreds Futterkiste </d:ContactName>
          <d:ContactTitle>Sales Representative</d:ContactTitle>
          <d:Address>Obere Str. 578</d:Address>
          <d:City>Toronto</d:City>
          <d:Region m:null="true" />
          <d:PostalCode>12209</d:PostalCode>
          <d:Country>Germany</d:Country>
          <d:Phone>030-0074321</d:Phone>
          <d:Fax>030-0076545</d:Fax>
        </m:properties>
      </content>
    </entry>
  </feed>
```

2.2.6.3 Verbose JSON Format

Verbose JavaScript Object Notation (JSON) is a lightweight data interchange format based on a subset of the JavaScript Programming Language standard, as specified in [ECMA-262/51]. JSON is a text format that is language independent, but uses conventions that are familiar to programmers of the C-family of languages (C, C++, JavaScript, and so on). Data serialized using JSON can easily be turned into JavaScript objects for programmatic manipulation. JSON notation consists of two structures: a JSON Object (collection of name/value pairs) and a JSON Array (an ordered list of values).

The following subsections define the mapping between Entity Data Model constructs and their serialized representations in the Verbose JSON, as specified in [RFC4627], for use in request/response messages, as specified in Request Types (section 2.2.7).

In all subsections that follow, if a data model construct is not explicitly referenced, then an associated Verbose JSON representation is not defined by this document.

Note Verbose JSON is considered a legacy format. OData 3.0 services MAY<61> support the Verbose JSON format and SHOULD<62> support the preferred OData 3.0 JSON format 2.2.6.6.

2.2.6.3.1 Common Verbose JSON Serialization Rules for All EDM Constructs

Literal values of the Entity Data Model (EDM) primitive types are represented as Verbose JSON literal values, as defined by the rules in the following Common Verbose JSON Serialization Rules for All EDM Constructs table. Grammar rules not defined here are specified in [RFC5234], [RFC4627], or both.

EDM primitive type	ABNF rule for primitive type representation in Verbose JSON payloads	Verbose JSON serialization format (ABNF grammar)
null	**nullLiteral**	`nullLiteral = "null"`
Edm.Binary	**VJsonBinary**	`VJsonBinary = quotation-mark <Base64 encoded value of the EDM.Binary property represented as a JSON string. See [RFC4627] for further details> quotation-mark`

Edm.Boolean	**VJsonBoolean**	`VJsonBoolean = false / true`
		
		`false = ; see [RFC4627] section 2.1`
		
		`true = ; see [RFC4627] section 2.1`

Edm.Byte	**VJsonByte**	`VJsonByte = ; see the byteLiteral rule in ; section 2.2.2`

Edm.DateTime	**VJsonDateTime**	`VJsonDateTime= quotation-mark "\/Date(" ticks [("+" / "-") offset] ")\/" quotation-mark`
		
		`ticks = *DIGIT`
		
		`; ticks is the number of milliseconds since midnight ; January 1, 1970`
		
		`offset = 4DIGIT ; offset represents the number of minutes to add (if preceded by "+") or substract (if preceded by "-") from the time value represented by ticks ; if no offset is specified, the value MUST be interpreted as UTC.`
		
		`;Note: This format is the same used by the ASP.NET ;AJAX framework, described in http://msdn2.microsoft.com/en-`

EDM primitive type	ABNF rule for primitive type representation in Verbose JSON payloads	Verbose JSON serialization format (ABNF grammar)
		;us/library/bb299886.aspx
Edm.Decimal	**VJsonDecimal**	VJsonDecimal = quotation-mark decimalLiteral quotation-mark decimalLiteral = ; see section 2.2.2 quotation-mark = ; see [RFC4627] section 2.5
Edm.Double	**VJsonDouble**	VJsonDouble = doubleLiteral doubleLiteral = ; see section 2.2.2
Edm.Guid	**VJsonGuid**	VJsonGuid = quotation-mark guidLiteral quotation-mark guidLiteral = ; see section 2.2.2
Edm.Int16	**VJsonInt16**	VJsonInt16 = ; see int16Literal in section 2.2.2
Edm.Int32	**VJsonInt32**	VJsonInt32 = ; see int32Literal in section 2.2.2
Edm.Int64	**VJsonInt64**	VJsonInt64 = quotation-mark int64Literal quotation-mark int64Literal = ; see section 2.2.2
Edm.SByte	**VJsonSByte**	VJsonSByte = ; see

EDM primitive type	ABNF rule for primitive type representation in Verbose JSON payloads	Verbose JSON serialization format (ABNF grammar)
		sByteLiteral in section 2.2.2
Edm.Single	**VJsonSingle**	VJsonSingle = singleLiteral singleLiteral = ; see section 2.2.2
Edm.String	**VJsonString**	VJsonString = string string = ; see [RFC4627] section 2.5
Edm.Time	**VJsonTime**	VJsonTime = quotation-mark timeLiteral quotation-mark timeLiteral = ; see section 2.2.2
Edm.DateTimeOffset	**VJsonDateTimeOffset**	VJsonDateTimeOffset = VJsonDateTime ; offset is required
Edm.Stream	**N/A**	namedStreamInVJson= see section 2.2.6.3.14
Edm.Geography	**N/A**	N/A
Edm.GeographyPoint	**pointVJsonLiteral**	pointVJsonLiteral = <See [GeoJSON] representation for a Point>
Edm.GeographyLineString	**lineStringVJsonLiteral**	lineStringVJsonLiteral = <See [GeoJSON] representation for a LineString>
Edm.GeographyPolygon	**polygonVJsonLiteral**	polygonVJsonLiteral = <See [GeoJSON] representation for a Polygon, except as modified below>

EDM primitive type	ABNF rule for primitive type representation in Verbose JSON payloads	Verbose JSON serialization format (ABNF grammar)
		In Ellipsoidal coordinates, all rings are equally interpretable as "outer". Therefore, the rings MUST have their control points in left-foot winding order. This means that the points to the left side of the ring, when traversing in serialized order, are in the polygon, while those to the right side are not. In planar coordinates, where "outer" is well defined, the first ring MUST be the outer ring, in accordance with the JSON standard.
Edm.GeographyCollection	**geoCollectionVJsonLiteral**	geoCollectionVJsonLiteral = <See [GeoJSON] representation for a GeometryCollection>
Edm.GeographyMultiPoint	**multiPointVJsonLiteral**	multiPointVJsonLiteral = <See [GeoJSON] representation for a MultiPoint>
Edm.GeographyMultiLineString	**multiLineStringVJsonLiteral**	multiLineStringVJsonLiteral = <See [GeoJSON] representation for a MultiLineString>
Edm.GeographyMultiPolygon	**multiPolygonVJsonLiteral**	multiPolygonVJsonLiteral = <See [GeoJSON] representation for a MultiPolygon, as modified below> In Ellipsoidal coordinates, all rings are equally interpretable as "outer". Therefore, the rings MUST have their control points in left-foot winding order. This means that the points to the left side of the ring, when traversing in serialized order, are in the polygon, while those to the right side are not. In planar coordinates, where "outer" is well defined, the first ring MUST be the outer ring, in accordance with the

EDM primitive type	ABNF rule for primitive type representation in Verbose JSON payloads	Verbose JSON serialization format (ABNF grammar)
		JSON standard.
Edm.Geometry	**N/A**	N/A
Edm.GeometryPoint	**pointVJsonLiteral**	N/A
Edm.GeometryLineString	**lineStringVJsonLiteral**	N/A
Edm.GeometryPolygon	**polygonVJsonLiteral**	N/A
Edm.GeometryCollection	**geoCollectionVJsonLiteral**	N/A
Edm.GeometryMultiPoint	**multiPointVJsonLiteral**	N/A
Edm.GeometryMultiLineString	**multiLineStringVJsonLiteral**	N/A
Edm.GeometryMultiPolygon	**multiPolygonVJsonLiteral**	N/A

Table: Common Verbose JSON Serialization Rules for All EDM Constructs

2.2.6.3.1.1 Modifications to GeoJSON for Use in OData

Any GeoJSON value that is used in OData SHOULD order the keys with type first, then coordinates, then any other keys. This improves streaming parser performance when parsing values on open types or in other cases where metadata is not present.

The GeoJSON [GeoJSON] standard requires that type **LineString** contains a minimum number of positions in its coordinates collection. This prevents serializing certain valid geospatial values. Therefore, in GeoJSON, the requirement that is stated as "For type 'LineString', the 'coordinates' member must be an array of two or more positions" is replaced with the requirement "For type 'LineString', the 'coordinates' member must be an array of positions" when used in OData.

All other arrays in GeoJSON are allowed to be empty, so no change is necessary. GeoJSON does require that any LinearRing contain a minimum of four positions. That requirement still holds that LinearRings can show up only in other arrays and that those arrays can be empty.

GeoJSON allows multiple types of coordinate reference systems (CRSs). In OData, only one of those types is allowed. In GeoJSON in OData, a CRS MUST be a named CRS. In addition, Open Geospatial Consortium CRS uniform resource names (URNs) are not supported. The CRS identifier MUST be a European Petroleum Survey Group (EPSG) system resource identifier (SRID) legacy identifier.

2.2.6.3.2 Entity Set (as a Verbose JSON Array)

An EntitySet or collection of entities MUST be represented as an array of Verbose JSON objects, with one object for each EntityType instance within the set. A Verbose JSON-based format for **EntityTypes** is defined in Entity Type (as a Verbose JSON object) (section 2.2.6.3.3).

An empty **EntitySet** or collection of entities (one that contains no **EntityType** instances) MUST be represented as an empty JSON array.

The syntax of the Verbose JSON representation of a collection of entities is defined by the grammar listed in this section. The grammar rule "**entitySetInVJson**" defines the Verbose JSON representation of a collection of entities that can be used in all versions of a request payload and OData 1.0 response payloads. The grammar rule "**entitySetInVJson2**" defines the OData 2.0 and OData 3.0 Verbose JSON representations of a collection of entities for response payloads only.

```
; Request and OData 1.0 response Verbose JSON representation of a collection of entities:
entitySetInVJson =  begin-array
                   [entityTypeInVJson *(value-seperator entityTypeInVJson)]
                    end-array

; OData 2.0 and OData 3.0 response Verbose JSON representation of a collection of entities:
entitySetInVJson2 = begin-object
                    [countNVP value-seperator]
                    resultsNVP
                    [value-seperator nextLinkNVP]
                    end-object


resultsNVP      = quotation-mark "results" quotation-mark
                  name-seperator
                  begin-array
                  [entityTypeInVJson *(value-seperator entityTypeInVJson)]
                  end-array

; see section 2.2.6.3.2.1 for additional details
countNVP        = quotation-mark "__count" quotation-mark
                  name-seperator
                  <count value as defined in section 2.2.6.3.2.1>

nextLinkNVP     = quotation-mark "__next" quotation-mark
                  name-seperator
                  quotation-mark
                  resourcePath "?" [skiptokenQueryOp]
                  quotation-mark

entityTypeInVJson = ; see section 2.2.6.3.3
resourcePath      = ; see section 2.2.3.1
sysQueryOption    = ; see section 2.2.3.1
```

Listing: Entity Set Verbose JSON Representation

In response payloads representing a collection of entities, if the server does not include an "entityTypeInVJson" name/value pair (see section 2.2.6.3.3) for every entity in the collection of entities identified by the associated URI, then the JSON array represents a partial collection of entities. In this case, a "nextLinkNVP" name/value pair MUST be included in the JSON array to indicate it represents a partial collection. The URI in the associated "nextLinkNVP" name/value pair MUST have a value equal to the URI, which identifies the next partial set of entities from the originally identified complete set. Such a URI SHOULD include a Skip Token system query option (section 2.2.3.6.1.9) to indicate that the URI addresses the subsequent partial set of entities.

Note The inclusion of a "nextLinkNVP" name/value pair in a Verbose JSON representation of a collection of entities has protocol versioning implications as described in Executing a Received RetrieveValue Request (section 3.2.5.4.2).

2.2.6.3.2.1 InlineCount Representation (for Collections of Entities)

Applies to the OData 2.0 and OData 3.0 protocols

This section defines the semantics of the "countNVP" grammar rule in section 2.2.6.3.2, which is supported only in OData 2.0 and OData 3.0.

A request URI MAY contain an **$inlinecount** system query option to indicate that the count of the number of entities represented by the query after filters have been applied ought to be included in the collection of entities returned from a data service. If such a query string object is present, the response MUST include the "countNVP" name/value pair with the value of the name/value pair equal to the count of the total number of entities addressed by the request URI.

The "countNVP" name/value pair MUST NOT be inside the Inline Representation of a related collection (section 2.2.6.3.9.1).

For example, the count of all Customer Entities using the Customer **EntityType** instance described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in Verbose JSON as shown in the following sample response payload. This example assumes the request URI includes the InlineCount system query option and the Top system query option with a value of 1.

```
{
  d:{
     "__count": "91",
     "results":[
     {
       "__metadata": { "uri": "Customers(\'ALFKI\')",
                       "type": "SampleModel.Customer",
                       "etag": "W/\"X\'000000000000FA01\'\"",
                       "properties" : {
                             "Orders" : {
                                    "associationuri" : "Customers(\'ALFKI\')/$links/Orders"
                             }
                       }
                     },
       "CustomerID": "ALFKI",
       "CompanyName": "Alfreds Futterkiste",
       "Address": { "Street": "57 Contoso St", "City": "Seattle" },
       "Version": "AAAAAAAA+gE=",
       "Orders": {
          "results": [
            {
               "__metadata": { "uri": "Orders(1)",
                               "type": "SampleModel.Order",
                               "properties" : {
                                 "Customer" : {
                                   "associationuri" : "Orders(1)/$links/Customer",
                                 },
                                 "OrderLines" : {
                                   "associationuri" : "Orders(1)/$links/OrderLines",
                                 }
                               }
                             },
               "OrderID": 1,
               "ShippedDate": "\/Date(872467200000)\/",
               "Customer":   { "__deferred": { "uri": "Orders(1)/Customer"} },
               "OrderLines": { "__deferred": { "uri": "Orders(1)/OrderLines"} }
            },
            {
               "__metadata": { "uri": "Orders(2)",
                               "type": "SampleModel.Order",
                               "properties" : {
                                 "Customer" : {
                                   "associationuri" : "Orders(2)/$links/Customer",
            .                    },
                                 "OrderLines" : {
                                   "associationuri" : "Orders(2)/$links/OrderLines",
                                 }
                               }
                             },
               "OrderID": 2,
               "ShippedDate": "\/Date(875836800000)\/",
               "Customer":   { "__deferred": { "uri": "Orders(2)/Customer"} },
               "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines"} }
            }
          ]
       }
     ]
  }
}
```

Listing: OData 3.0 Verbose JSON-Formatted InlineCount representation

2.2.6.3.2.2 Entity Set (as a Verbose JSON Array) with Actions

Applies to the OData 3.0 protocol

In OData 3.0, it is possible to advertise actions (section 2.2.1.3) that are bound to the definition of the feed (or **EntitySet**) in the feed.

A new action's name/value pair MAY be included as a property of the JSON object that is the value of the optional "__metadata" JSON object. The "__metadata" JSON object is a peer of the results name/value pair that actually holds the array of entities.

The value of the action's name/value pair is a JSON object that contains name/value pairs for each action that the server advertises as bindable to the definition of the feed.

For each action, the name MUST be the Action Metadata URL (section 2.2.1.3.1) that identifies the action and the value MUST be an array of JSON objects. Any number of JSON objects is allowed in this array. Each object in this array MUST have at least two name/value pairs: title and target. The order of these name/value pairs is insignificant.

The target name/value pair MUST be included and MUST contain a URL. This is the URL to which clients issue an Invoke Action request (section 2.2.7.5.1) to invoke the action. The binding parameter is assumed to be bound to the encompassing feed definition. Therefore, if the client invokes the action via the target URL, the request body MUST not include a value for the binding parameter.

The title name/value pair MUST be included and MUST contain a simple string that is used as a simple but not necessarily unique name for the action. Generally, servers SHOULD specify a value that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

Actions advertised in the feed MUST be interpreted as being bound to the definition of the feed and not to the items that are represented in the feed.

Actions that are advertised in the feed MUST be advertised only if the server can fully encode the action, the resource path, and the appropriate system query options that define the feed.

System Query Options (section 2.2.3.6.1) that change the membership of the feed MUST be considered part of the feed definition. In practice, this means that the target URL that is used to invoke the action MUST encode the following system query options if they are used to define the feed:

- $filter (section 2.2.3.6.1.4)
- $expand (section 2.2.3.6.1.3)
- $orderby (section 2.2.3.6.1.6)
- $skip (section 2.2.3.6.1.7)
- $top (section 2.2.3.6.1.8)

The remaining system query options, generally, do not define the feed and do not need to be encoded in the target of the action:

- $format (section 2.2.3.6.1.5)
- $skiptoken (section 2.2.3.6.1.9)
- $inlinecount (section 2.2.3.6.1.10)
- $select (section 2.2.3.6.1.11)

2.2.6.3.2.3 Entity Set (as a Verbose JSON Array) with Functions

Applies to the OData 3.0 protocol

In OData 3.0, the functions name/value pair MAY be included as a property JSON object that is the value of the optional "__metadata" JSON object. The value of the functions name/value pair is a JSON object that contains name/value pairs for each function that the server advertises as bindable to the definition of the feed.

For each function, the name MUST be the function metadata URL (section 2.2.1.4.1) that identifies the function or a set of bindable function overloads and the value MUST be an array of JSON objects. Any number of JSON objects is allowed in this array. Each object in this array MUST have a least two name/value pairs: title and target. The order of these name/value pairs is insignificant.

The target name/value pair MUST be included and MUST contain a URL. This is the URL to which clients issue an Invoke Function request (section 2.2.7.5.2) to invoke the function. The binding parameter is assumed to be bound to the encompassing feed definition. Therefore, if the client invokes the function, the invoke request URL MUST not include a value for the binding parameter by appending the value as a parameter to the target URL.

The title name/value pair MUST be included and MUST contain a simple string that is used as a simple but not necessarily unique name for the function. Generally, servers SHOULD specify a value that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

The function metadata URL MUST identify only functions that are bindable to the current feed definition. If overloads exist that cannot be bound to the current feed definition, individual **m:Function** elements SHOULD be returned that each have a function metadata URL that identifies a specific bindable overload.

Functions advertised in the feed MUST be interpreted as being bound to the definition of the feed and not to the items that are represented in the feed.

Functions that are advertised in the feed MUST only be advertised if the server can fully encode the function, the resource path, and the appropriate system query options (section 2.2.3.6.1) that define the feed.

System query options (section 2.2.3.6.1) that change the membership of the feed MUST be considered part of the feed definition. In practice, this means that the target URL used to invoke the function MUST encode the following system query options if they are used to define the feed:

- ($filter) (section 2.2.3.6.1.4)
- ($expand) (section 2.2.3.6.1.3)
- ($orderby) (section 2.2.3.6.1.6)
- ($skip) (section 2.2.3.6.1.7)
- ($top) (section 2.2.3.6.1.8)

The remaining system query options, generally do not define the feed and do need not to be encoded in the target of the function:

- ($format) (section 2.2.3.6.1.5)
- ($skiptoken) (section 2.2.3.6.1.9)
- $inlinecount) (section 2.2.3.6.1.10)
- ($select) (section 2.2.3.6.1.11)

2.2.6.3.3 Entity Type (as a Verbose JSON Object)

An instance of an EntityType MUST be serialized as a Verbose JSON object.

Each property on the **EntityType** MUST be represented as a name/value pair, as specified in [RFC4627], within the object. Alternatively, if the **EntityType** instance being represented is identified with a URI that includes a Select system query option (section 2.2.3.6.1.11), the prior rule is relaxed such that only the properties identified by the **$select** query option MUST be represented by name/value pairs. The name in the name/value pair is the name of the property as defined on the **EntityType**, and the value of the pair is the value of the property. The order name/value pairs that appear within a JSON object MUST be considered insignificant. Name/value pairs not representing a property defined on the **EntityType** SHOULD NOT be included. The following subsections describe additional formatting rules for each type of property defined on an **EntityType**.

The Verbose JSON serialization of an **EntityType** instance MAY<63> include a name/value pair named "__metadata". This name/value pair is not data, but instead, by convention defined in this document, specifies the metadata for the **EntityType** instance that the JSON object represents. The ordering of this name/value pair with respect to other name/value pairs that represent properties that are defined on the **EntityType** is insignificant. In OData 1.0 and OData 2.0, the value of the "__metadata" property contains seven name/value pairs: "uri", "type," "etag", "edit_media", "media_src", "media_etag", and "content_type". In OData 3.0, four more name/value pairs are added: "properties", "actions", "functions", and "id". The order of these name/value pairs is insignificant. The value of the "uri" name/value pair MUST be the Canonical URIs identifying the **EntityType** instance represented by the JSON object.

The "type" name/value pair MUST be included if the **EntityType** of the **EntityType** instance represented by the JSON object is part of an inheritance hierarchy, as described in [MC-CSDL] (section 1). If the **EntityType** is not part of an inheritance hierarchy, the "type" name/value pair MAY be included. The value of the "type" name/value pair MUST be the namespace qualified name, as specified in [MC-CSDL], of the **EntityType** of the instance that the JSON object represents.

The "etag" name/value pair MAY be included. When included, it MUST represent the concurrency token associated with the **EntityType** instance ETag (section 2.2.5.4) and MUST be used instead of the ETag HTTP header defined in ETag (section 2.2.5.4), which, as specified in [RFC2616], is used to represent a single entity when multiple entities are present in a single payload.

The "media_src" and "content_type" name/value pairs MUST be included and the "edit_media" and "media_etag" name/value pairs MAY be included if the entity being represented is a Media Link Entry. For example, the description of the entity type as shown in the data services' CSDL document includes the **HasStream="true"** attribute as defined in section 2.2.3.7.2. If the entity being represented is not a Media Link Entry, the "edit_media", "media_src", "media_etag", and "content_type" name/value pairs cannot be included.

The "id" name/value pair can be included if the server is using OData 2.0 and has to be included if the server is using OData 3.0.

The value of the "edit_media" name/value pair MUST be a URI that is equivalent to the value of the "href" attribute on an **atom:linkrel="edit-media"** AtomPub element if the entity was to be represented by the AtomPub [RFC5023] format, instead of Verbose JSON. The value of the "media_src" name/value pair MUST be a URI that is equivalent to the value of the "src" attribute on the **atom:content** AtomPub element if the entity was to be represented using the AtomPub [RFC5023] format, instead of Verbose JSON. The value of the "content_type" name/value pair MUST be equivalent to the value of the "type" attribute on the **atom:content** AtomPub element if the entity was to be represented using the AtomPub [RFC5023] format, instead of Verbose JSON. The value of the "media_etag" name/value pair MUST be equal to the value of the concurrency token associated with the Media Resource identified by the "edit_media" and/or "media_src" name/value pairs.

The value of the "properties" name/value pair MAY contain a JSON object for each **NavigationProperty**. Each **NavigationProperty** is serialized as name/value pairs in which the value

is a JSON object that contains a single name/value pair, with the name equal to the name of the **NavigationProperty** and a value equal to the URI that can be used to manage the relationship between the related entities.

The JSON object representing the **EntityType** SHOULD also contain representations of the properties defined on the **EntityType**. Each **EDMSimpleType**, **ComplexType**, and **NavigationProperty** defined on the **EntityType** MUST be formatted according to the directives in sections EDMSimpleType Property (section 2.2.6.3.8), Complex Type (section 2.2.6.3.4), and Navigation Property (section 2.2.6.3.6).

The syntax of the Verbose JSON representation of an entity is defined by the grammar listed in this section. The grammar rule "entityTypeInVJson" defines the representation of an entity that can be used in both request and response payloads.

```
; OData 1.0, OData 2.0, and OData 3.0 Verbose JSON representation of an entity:
entityTypeInVJson  = entityTypeBody

entityTypeBody      = begin-object
                      (
                          (entityTypeProperty)
                          *( value-seperator ( entityTypeProperty / entityTypeProperty ))
                          [ value-seperator metadataNVP]
                      )
                      / metadataNVP
                      end-object

metadataNVP         =  quotation-mark "__metadata" quotation-mark
                       name-seperator
                       begin-object
                       ( uriNVP
                         [value-seperator idNVP]
                         [value-seperator typeNVP]
                         [value-seperator etagNVP]
                         [value-separator mleMetadata])
                         [propmetadataNVP]
                         [actionsNVP]
                         [functionsNVP]
                       /
                       ( typeNVP
                         [value-seperator etagNVP])
                         [propmetadataNVP]
                       /
                         etagNVP
                         [propmetadataNVP]
                       /
                         propmetadataNVP
                       end-object

entityTypeProperty  = entityPropertyInVJson
                        /entityCTInVJson
                        /deferredNavProperty
                        /namedSteamInVJson

uriNVP              = quotation-mark "uri" quotation-mark
                      name-seperator
                      quotation-mark resourcePath quotation-mark

; OData 3.0 Verbose JSON representation of Actions:
actionsNVP          = quotation-mark "actions" quotation-mark
                      name-seperator
                      begin-object
                      [ actionNVP
                      *(value-seperator actionNVP) ]
                      end-object

actionNVP           = quotation-mark actionUri quotation-mark
                      name-seperator
```

```
                      begin-object
                      titleNVP
                      value-seperator targetNVP
                      end-object

; OData 3.0 Verbose JSON representation of Functions:
functionsNVP        = quotation-mark "functions" quotation-mark
                      name-seperator
                      begin-object
                      [ functionNVP
                      *(value-seperator functionNVP) ]
                      end-object

functionNVP         = quotation-mark functionUri quotation-mark
                      name-seperator
                      begin-object
                      titleNVP
                      value-seperator targetNVP
                      end-object

titleNVP            = quotation-mark "title" quotation-mark
                      name-seperator
                      quotation-mark titleValue quotation-mark

titleValue          = *pchar        ; section 3.3 of [RFC3986]

targetNVP           = quotation-mark "target" quotation-mark
                      name-seperator
                      quotation-mark targetUrl quotation-mark

targetUrl           = ; a Url that can be used as the target for
                      ; either an Invoke Action Request (section 2.2.7.5.1) or
                      ; an Invoke Function Request (section 2.2.7.5.2)

actionUrl           = ; an Action Metadata URL (section 2.2.1.3.1)

functionUrl         = ; a Function Metadata URL (section 2.2.1.4.1)

; OData 3.0 Verbose JSON representation of the URI representing the relationship between ;
related entities:
propmetadataNVP     = quotation-mark "properties" quotation-mark
                      name-seperator
                      begin-object
                      (
                         entityTypeProperty
                         name-seperator
                         begin-object
                        (
                           quotation-mark "associationuri" quotation-mark
                           name-seperator
                           quotation-mark resourcePath quotation-mark
                        )
                      )
                      end-object

typeNVP             = quotation-mark "type" quotation-mark
                      name-seperator
                      quotation-mark entityType quotation-mark

etagNVP             = quotation-mark "type" quotation-mark
                      name-seperator
                      quotation-mark entityTag quotation-mark

editMediaNVP        = quotation-mark "edit_media" quotation-mark
                      name-seperator
                      quotation-mark resourcePath quotation-mark

mediaSrcNVP         = quotation-mark "media_src" quotation-mark
                      name-seperator
                      quotation-mark resourcePath quotation-mark
```

```
contentTypeNVP      = quotation-mark "content_type" quotation-mark
                      name-seperator
                      quotation-mark contentType quotation-mark

idNVP               = quotation-mark "id" quotation-mark
                      name-seperator
                      quotation-mark resourcePath quotation-mark


mleMetadata         =
                      media_srcNVP
                      value-seperator contentTypeNVP
                      [value-seperator etagNVP]
                      [value-seperator editMediaNVP]

deferredNavProperty = entityNavProperty name-seperator
                      begin-object
                      quotation-mark "__deferred" quotation-mark
                      name-seperator
                      begin-object
                      uriNVP
                      end-object
                      end-object


contentType         = <An IANA-defined [IANA-MMT] content type>

resourcePath               = ; section 2.2.3.1
entityCTInVJson            = ; section 2.2.6.3.4
entityPropertyInVJson      = ; section 2.2.6.3.8
entityPropertyValueInVJson = ; section 2.2.6.3.8
entityType                 = ; section 2.2.3.1
entityNavProperty          = ; section 2.2.3.1
entityTag                  = ; section 2.2.5.4

begin-object               = ; [RFC4627] section 2
name-seperator             = ; [RFC4627] section 2
value-seperator            = ; [RFC4627] section 2
value                      = ; [RFC4627] section 2.1
```

Listing: OData 3.0 Entity Type Verbose JSON Representation

For example, the Customer EntityType instance described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in Verbose JSON, as shown in the following listing.

```
{
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders"
             } },
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\"",
                   "properties" : {
                         "Orders" : {
                                "associationuri" : "Customers(\'ALFKI\')/$links/Orders"
                         }
                   }
                 }
}
```

Listing: OData 3.0 Verbose JSON-formatted Customer Entity

2.2.6.3.3.1 Entity Type (as a Verbose JSON Object) with Actions

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, the actions name/value pair MAY be included. The value is a JSON object that contains name/value pairs of each action that the server looks for to advertise as bindable to the entity.

For each action, the name MUST be the action metadata URL (section 2.2.1.3.1) that identifies the action and the value MUST be an array of JSON objects. Any number of JSON objects is allowed in this array. Each object in this array MUST have at least two name/value pairs: title and target. The order of these name/value pairs is insignificant.

The target name/value pair MUST be included and contain a URL. When a client issues an Invoke Action request (section 2.2.7.5.1), the value is to be used as the URL. The binding parameter is assumed to be bound to the encompassing entity. Therefore, if the client invokes the action via the target URL, it MUST not include a value for the binding parameter in the request body.

The title name/value pair MUST be included and contain a simple string that is used as a simple but not necessarily unique name for the action. Generally, servers SHOULD specify a value that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

Actions can also be selectively requested or omitted by using different Select system query option (section 2.2.3.6.1.11) combinations.

If the **EntityType** JSON Object is retrieved as part of a feed (see section 2.2.6.3.2) and it is expensive to identify whether an action that is requested, either implicitly or explicitly, by using Select system query option (section 2.2.3.6.1.11) can be bound to a particular entity, a server SHOULD advertise the action and fail later if the action is invoked and found to be unavailable.

2.2.6.3.3.2 Entity Type (as a Verbose JSON Object) with Functions

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, the function's name/value pair MAY be included. The value is a JSON object which contains name/value pairs of each function the server advertises as bindable to the entity.

For each function, the name MUST be the function metadata URL (section 2.2.1.4.1) that identifies the function and the value MUST be an array of JSON objects. Any number of JSON objects is allowed in this array. Each object in this array MUST have a least two name/value pairs: title and target. The order of these name/value pairs is insignificant.

The target name/value pair MUST be included and MUST contain a URL. When a client issues an Invoke Function request (section 2.2.7.5.2), this is the URL to use. The binding parameter is assumed to be bound to the encompassing entity. Therefore, if the client invokes the function via the target URL, it MUST not include a value for the binding parameter by appending a parameter to the target URL.

The title name/value pair MUST be included and MUST contain a simple string that is used as a simple but not necessarily unique name for the function. Generally, servers SHOULD specify a value that would be easily understood by any user because the title is likely to be used by clients to display options to an end user.

The function metadata URL MUST only identify functions that are bindable to the current **EntityType**. If overloads exist that cannot be bound to the current **EntityType**, individual **m:Function** elements SHOULD be returned that each have a function metadata URL that identifies a specific bindable overload.

If a function to be advertised has overloads that can all be bound to the current **EntityType**, the server SHOULD expose a single function that contains a name equal to the function metadata URL that identifies all the overloads.

Functions can also be selectively requested or omitted by using different Select system query option (section 2.2.3.6.1.11) combinations.

If the **EntityType** JSON Object is retrieved as part of a Feed (see section 2.2.6.2.1) and it is expensive to identify whether a function that is requested explicitly by using a Select system query option (section 2.2.3.6.1.11) is available, a server SHOULD advertise the requested function and fail later if the function is invoked and found to be unavailable.

2.2.6.3.4 Complex Type

An instance of a **ComplexType** MUST be represented as a Verbose JSON object. Each declared property defined on the **ComplexType** MUST be represented as a name/value pair within the JSON object. Additional name/value pairs that do not represent a declared property of the **ComplexType** SHOULD NOT be included. The name in the name/value pair MUST equal the name of the declared property on the **ComplexType** and the value of the pair MUST equal the value of the property. The order name/value pairs that appear within the JSON object MUST be considered insignificant.

The syntax of the Verbose JSON representation of a **ComplexType** is defined by the grammar listed in this section. The grammar rule "entityCTInVJson" defines the Verbose JSON representation of a **ComplexType** that can be used in both request and response payloads.

```
entityCTInVJson = begin-object
                  entityCTBody
                  end-object

entityCTBody    = quotation-mark entityComplexProperty quotation-mark
                  name-seperator
                  entityCTValue

entityCTMetadata = quotation-mark "__metadata" quotation-mark
                   name-seperator
                   begin-object
                   [typeNVP]
                   end-object

entityCTValue   = begin-object
                   [
                     (
                       entityPropertyInVJson /
                       entityCTBody /
                       entityCTMetadata
                     )
                     *(
                       (value-seperator entityPropertyInVJson) /
                       (value-seperator entityCTBody)
                     )
                  ]
                  end-object

entityPropertyInVJson   = ; see section 2.2.6.3.8

typeNVP                  = ;see section 2.2.6.3.3
```

Listing: ComplexType Verbose JSON Representation

2.2.6.3.5 Collection of Complex Type Instances

A collection of ComplexType instances MUST be represented as an array of JSON objects. Each object in the array represents a single **ComplexType** instance as specified in Complex Type (section 2.2.6.3.4).

The syntax of the Verbose JSON representation of a collection of **ComplexType** instances is defined by the grammar listed in this section. The grammar rule "entityCollCTInVJson" defines the Verbose JSON representation of a collection of **ComplexType** instances that can be used in all versions of request payloads and OData 1.0 response payloads. The grammar rule "entityCollCTInVJson2" defines the OData 2.0 Verbose JSON representation of a collection of **ComplexType** instances for response payloads only. The grammar rule "entityCollCTInVJson3" defines the OData 3.0 Verbose JSON representation of a collection of **ComplexType** instances for response payloads only.

```
; Request and OData 1.0 response Verbose JSON representation of a collection of ComplexType
instances:
entityCollCTInVJson =  begin-array
                       entityCTValue      ; see section 2.2.6.3.4
                       *(value-seperator entityCTValue)
                       end-array

; OData 2.0 response Verbose JSON representation of a collection of ComplexType instances;
entityCollCTInVJson2 = begin-object
                       resultsNVP
                       [(value-seperator entityCTMetadata)] ; see section 2.2.6.3.4
                       end-object

; OData 3.0 response Verbose JSON representation of a collection of ComplexType instances:
entityCollCTInVJson3 = begin-object
                       [collMetadataNVP value-seperator]
                       resultsNVP
                       end-object

collMetadataNVP =   quotation-mark "__metadata" quotation-mark
                    name-seperator
                    begin-object
                    [collComplexTypeNVP]
                    end-object

collComplexTypeNVP = quotation-mark "type" quotation-mark
                    name-seperator
                    quotation-mark "Collection(" complexTypeNVP ")" quotation-mark

resultsNVP          = quotation-mark "results" quotation-mark
                      name-seperator
                      begin-array
                      entityCTValue      ; see section 2.2.6.3.4
                      *(value-seperator entityCTValue)
                      end-array
```

2.2.6.3.6 Navigation Property

The default representation of a **NavigationProperty** is as a JSON name/value pair. The name is equal to "__deferred" and the value is a JSON object that contains a single name/value pair with the name equal to "uri". The value of the "uri" name/value pair MUST be a URI relative to the service root URI, as specified in Service Root (section 2.2.3.2), that identifies the **NavigationProperty**.

The syntax of a **NavigationProperty**, represented within a JSON object, is shown using the grammar rule "deferredNavProperty" in the Entity Type Verbose JSON Representation listing in Entity Type (as a JSON Object) (section 2.2.6.3.3).

OData 3.0 adds another JSON object with the name "properties" to the "__metadata" object that contains an array of objects, each of which SHOULD have the name of a **NavigationProperty** in the entity. Each object has one name/value pair with the name "associationuri". The value of the

"associationuri" name/value pair MUST be a URI that represents the association between the related entities.

The syntax of the OData 3.0 properties object is shown by using the grammar rule "propmetadataNVP" in the Entity Type Verbose JSON Representation listing in Entity Type (as a JSON Object) (section 2.2.6.3.3).

2.2.6.3.7 Collection of EDMSimpleType Values

A collection of EDMSimpleType values MUST be represented as an array of JSON primitives. Each element in the array represents a single primitive type value.

The syntax of the Verbose JSON representation of a collection of **EDMSimpleType** values is defined by the grammar listed in this section. The grammar rule "entityCollPrimValueInVJson" defines the Verbose JSON representation of a collection of **EDMSimpleType** values that can be used in all versions of request payloads and OData 1.0 response payloads. The grammar rule "entityCollPrimValueInVJson2" defines the OData 2.0 Verbose JSON representation of a collection of **EDMSimpleType** values for response payloads only. The grammar rule "entityCollPrimValueInVJson3" defines the OData 3.0 Verbose JSON representation of a collection of **EDMSimpleType** values for response payloads only.

```
; Request and OData 1.0 response Verbose JSON representation of a collection of EDMSimpleType
values:
entityCollPrimValueInVJson =  begin-array
                              entityPropertyValueInVJson
                              ; see section 2.2.6.3.8
                              *(value-seperator entityPropertyValueInVJson)
                              end-array

; OData 2.0 response Verbose JSON representation of a collection of EDMSimpleType values:
entityCollPrimValueInVJson2 = quotation-mark "results" quotation-mark
                              name-seperator
                              begin-array
                              entityPropertyValueInVJson
                              ; see section 2.2.6.3.8
                              *(value-seperator entityPropertyValueInVJson)
                              end-array

; OData 3.0 response Verbose JSON representation of a collection of EDMSimpleType values:
entityCollPrimValueInVJson3 = begin-object
                        [collEdmSimpleMetadataNVP value-seperator]
                        resultsNVP
                        end-object

collEdmSimpleMetadataNVP =   quotation-mark "__metadata" quotation-mark
                     name-seperator
                     begin-object
                     [collEdmSimpleTypeNVP]
                     end-object

collEdmSimpleTypeNVP = quotation-mark "type" quotation-mark
                     name-seperator
                     quotation-mark "Collection(" edmSimpleTypeNVP ")" quotation-mark

edmSimpleTypeNVP =; EDM SimleType name
```

2.2.6.3.8 EDMSimpleType Property

A property of type EDMSimpleType MUST be represented as a JSON name/value pair. The name in the name/value pair MUST be equal to the name of the Entity Data Model (EDM) property and the value MUST be set to the value of the property. The value is formatted, as specified in Common Verbose JSON Serialization Rules for All EDM Constructs (section 2.2.6.3.1).

When represented as part of the Verbose JSON representation of an **EntityType** or **ComplexType**, the syntax of an **EDMSimpleType** property formatted in Verbose JSON is as follows.

```
entityPropertyInVJson =  quotation-mark entityProperty quotation-mark
                         name-seperator
                         entityPropertyValueInVJson

entityPropertyValueInVJson = <EDMSimple type serialized as per section 2.2.6.3.1>
```

When represented as a standalone construct, the syntax of the Verbose JSON representation of an **EDMSimpleType** is defined by the grammar listed in this section. The grammar rule "entityPropertyInVJson" defines the Verbose JSON representation of an **EDMSimpleType** property that can be used in all versions of request payloads and OData 1.0 response payloads. The grammar rule "entityPropertyInVJson2" defines the OData 2.0 and OData 3.0 Verbose JSON representations of an **EDMSimpleType** property for response payloads only.

```
; Request and OData 1.0 response Verbose JSON representation of a property:
entityPropertyInVJson  = quotation-mark entityProperty quotation-mark
                         name-seperator
                         entityPropertyValueInVJson

; OData 2.0 and OData 3.0 response Verbose JSON representation of a property:
entityPropertyInVJson2 = quotation-mark "results" quotation-mark
                         name-seperator
                         begin-object
                         quotation-mark entityProperty quotation-mark
                         name-seperator
                         entityPropertyValueInVJson
                         end-object
```

2.2.6.3.9 Deferred Content

The serialized representation of an entity and its related entities, identified by NavigationProperties, can be large. To conserve resources (bandwidth, CPU, and so on), it is generally not a good idea for a data service to return the full graph of entities related to the EntityType instance or set identified in a request URI. For example, a data service ought to defer sending entities represented by any navigation property in a response unless explicitly asked to send those entities via the **$expand** system query option, as described in Expand System Query Option ($expand) (section 2.2.3.6.1.3).

In Verbose JSON-formatted **EntityType** instances (see Entity Type (as a JSON Object) (section 2.2.6.3.3)), **NavigationProperties** serialized as name/value pairs in which the value is a JSON object containing a single name/value pair with the name "__deferred" and a value that is a JSON object containing a single name/value pair with the name "uri" and a string value, which is a URL that can be used to retrieve the deferred content, signify deferred **NavigationProperty** content (for example, the entities represented by the **NavigationProperty** are not serialized inline). For example, using the two **EntityTypes** Customer and Order, as described in Appendix A: Sample Entity Data Model and CSDL Document (section 6), the default Verbose JSON serialization (with deferred **NavigationProperty** content) of the Customer instance with EntityKey value of "ALFKI" is shown in Entity Type (as a JSON object) (section 2.2.6.3.3).

In the example, the presence of the "__deferred" name/value pair signifies that the value of the Orders **NavigationProperty** is not directly represented on the JSON object in this serialization. In order to obtain the deferred value(s), a client would make a separate request directly to the navigation property URI (`service.svc/Customers('ALFKI')/Orders`) or explicitly ask that the property be serialized inline via the **$expand** system query option, as described in Expand System Query Option ($expand) (section 2.2.3.6.1.3).

2.2.6.3.9.1 Inline Representation

As described in Expand System Query Option ($expand) (section 2.2.3.6.1.3), a request URI can include the **$expand** system query option to explicitly request the entity or entities represented by a NavigationProperty be serialized inline, rather than deferred. The example shown in this section uses the same data model as the Deferred Content example referenced in the preceding section. However, the following example shows the value of the Orders **NavigationProperty** serialized inline.

A **NavigationProperty** that is serialized inline MUST be represented as a name/value pair on the JSON object with the name equal to the **NavigationProperty** name. If the **NavigationProperty** identifies a single EntityType instance, the value MUST be a JSON object representation of that **EntityType** instance, as specified in Entity Type (as a JSON object) (section 2.2.6.3.3). If the **NavigationProperty** represents an EntitySet, the value MUST be as specified in Entity Set (as a JSON array) (section 2.2.6.3.2).

```
                    "etag": "W/\"X\'000000000000FA01\'""
{
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle",
     "Location": {
       "crs": {
         "type": "name",
         "properties": { "name": "EPSG:4326" }
       },
       "type": "Point", "coordinates": [-127.9324, 49.2345]
     }
   },
   "Version": "AAAAAAAA+gE=",
   "Orders":
       {
         Results: [
         {
            "__metadata": { "uri": "Orders(1)",
                            "type": "SampleModel.Order"
                          },
            "OrderID": 1,
            "ShippedDate": "\/Date(872467200000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(1)/Customer"} },
            "OrderLines": { "__deferred": { "uri": "Orders(1)/OrderLines"} },
            "__metadata": { "uri": "Orders(1)",
                            "type": "SampleModel.Order",
                            "properties" : {
                               "Customer" : {
                                  "associationuri" : "Orders(1)/$links/Customer"
                               },
                               "OrderLines" : {
                                  "associationuri" : "Orders(1)/$links/OrderLines"
                               }
                             }
                         }


         },
         {
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } },
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } },
            "__metadata": { "uri": "Orders(2)",
                            "type": "SampleModel.Order",
                            "properties" : {
                               "Customer" : {
                                  "associationuri" : "Orders(2)/$links/Customer"
                               },
                               "OrderLines" : {
                                  "associationuri" : "Orders(2)/$links/OrderLines"
                               }
                             }
```

```
                        }


            }
      ] ,
      "__metadata": { "uri": "Customers(\'ALFKI\')",
                      "type": "SampleModel.Customer",
                      "etag": "W/\"X\'000000000000FA01\'\"",
                      "properties" : {
                            "Orders" : {
                                  "associationuri" : "Customers(\'ALFKI\')/$links/Orders"
                            }
                      }
                    }

}
```

Listing: OData Verbose 3.0 JSON-Formatted Customer Entity with the Orders Navigation Property Value Formatted Inline

2.2.6.3.10 Links

Links represent unidirectional associations or one direction of a bidirectional association between EntityType instances. In the Verbose JSON format, Links are serialized as an array of URIs, each of which identifies a single linked entity.

When represented in Verbose JSON, Links MUST be formatted, as shown in the table in the following listing, ABNF Grammar for Links Represented in Verbose JSON, using one JSON object containing a single "uri" name/value pair per Link. The value of the "uri" name/value pair on each object MUST equal the absolute, canonical URI representing the linked-to **EntityType** instance.

The syntax of the Verbose JSON representation of a collection of links is defined by the grammar listed in this section. The grammar rule "linkCollVJson" defines the Verbose JSON representation of a collection of links that can be used in all versions of a request payload and OData 1.0 response payloads. The grammar rule "linkCollVJson2" defines the OData 2.0 and OData 3.0 Verbose JSON representations of a collection of links for response payloads only.

```
; Request and OData 1.0 response Verbose JSON representation of a collection of  links
linkCollVJson       =  begin-array
                       *linkVJson
                       end-array

; OData 2.0 and OData 3.0 response Verbose JSON representation of a collection of links
linkCollVJson2       =  begin-object
                       [countNVP value-seperator]
                       linkCollResultsNVP
                       [value-seperator nextUriNVP]
                       end-object

linkCollResultsNVP  =  quotation-mark "results" quotation-mark
                       name-seperator
                       begin-array
                       *linkVJson
                       end-array

countNVP            =  ; see section 2.2.6.3.2

nextUriNVP          =  ; see section 2.2.6.3.2 for ABNF.
                       ; see section 2.2.6.5.5.2 for AtomPub Format.

;Grammar rules common to OData 1.0, OData 2.0, and OData 3.0
linkVJson            =  begin-object
                       [linkUriNVP
```

```
                        *(value-seperator linkUriNVP) ]
                       end-object

linkUriNVP          =  quotation-mark "uri" quotation-mark
                       name-seperator
                       quotation-mark dataServiceNqo-URI quotation-mark
                       ; see section 2.2.3.1
```

Listing: ABNF Grammar for Links Represented in Verbose JSON

For example, when using the sample model and instance data, as described in Appendix A: Sample Entity Data Model and CSDL Document (section 6), the Links from the Customer with EntityKey "ALFKI" to Order instances is represented as shown in the following Example of Links Formatted by Using OData 1.0 Verbose JSON Representation listing. Each URI in the array identifies a single Order that is associated with the Customer.

```
[
  {"uri": "http://host/service.svc/Orders(1)"},
  {"uri": "http://host/service.svc/Orders(2)"}
]
```

Listing: Example of Links Formatted by Using OData 1.0 Verbose JSON Representation

2.2.6.3.11 InlineCount Representation (for Collections of Links)

Applies to the OData 2.0 and OData 3.0 protocols

This section defines the semantics of the "countNVP" grammar rule in section 2.2.6.3.10, which is supported only in the OData 2.0 and OData 3.0 protocols.

A request URI MAY contain an **$inlinecount** system query option to indicate that the count of the number of links represented by the query is included in the collection of links returned from the data service. If such a query string token is present, the response MUST include the countNVP name/value pair (before any linkURINVP name/value pairs) with the value of the name/value pair equal to the count of the total number of links addressed by the request URI.

2.2.6.3.12 Service Document

Service Document specifies that AtomPub, as specified in [RFC5023], defines a Service Document that describes collections of resources available from a data service. The root URL of a data service that implements the protocol defined in this document MUST identify such a service document. This section defines a Verbose JSON representation, as specified in [RFC4627], of the data provided in an AtomPub, as specified in [RFC5023] Service Document. For a description of the contents of a Service Document for which this section defines a Verbose JSON serialization, see Service Document.

The syntax of a Verbose JSON-serialized Service Document is as shown in the grammar that follows:

```
VJsonServiceDocument = begin-object
                       quotation-mark "EntitySets" quotation-mark
                       name-seperator
                       begin-array

                       entitySetName   ; One for each Entity Set in the data service
                                       ; as defined by the Entity Sets shown in the
                                       ; CSDL document returned from the data
                                       ; service's $metadata endpoint
                       *("," entitySetName)

                       end-array
                       end-object
```

```
entitySetName       = <A JSON string literal (quoted) equal to the name of an Entity Set
                      in the data model associated with the data service>
```

The following is an example Verbose JSON, as specified in [RFC4627], representation of the information provided by an AtomPub, as specified in [RFC5023], Service Document for the Entity Data Model (EDM) that is described in Appendix A: Sample Entity Data Model and CSDL Document.

```
{
  "EntitySets": [
     "Customers",
     "Orders",
     "OrderDetails"
  ]
}
```

Listing: Verbose JSON Service Document Describing a Data Service

2.2.6.3.13 Collection Property

Applies to the OData 3.0 protocol

In the OData 3.0 protocol, a collection property of a **ComplexType** or **EDMSimpleType** MUST be represented in the same way as collections of a **ComplexType**, as described in Collection of Complex Type Instances (section 2.2.6.3.5). Similarly, collections of **EDMSimpleTypes** MUST be represented the same as collections of **EDMSimpleTypes**, as described in Collection of EDMSimpleType Values (section 2.2.6.3.7). A "__metadata" object with a name/value pair called “type” MAY be present to specify the **EDMSimpleType** or the **ComplexType**.

```
{
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\"",
                    "properties" : {
                              "Orders" : {
                                "associationuri" : " Customers(\'ALFKI\')/$links/Orders "
                              }
                     }
                  },
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
  "EmailAddresses": {"__metadata": { "type": "Collection(Edm.String)" },
             "results":["mike@company.com", "mike2@company.com"]},
  "AlternateAddresses":{"__metadata": { "type": "Collection(SampleModel.Address)" },
             "results":[{"Street": "123 contoso street", "Apartment": "508"},
                        {"__metadata": { "type": "SampleModel.EAddress"},
                         "Street": "834 1st street" }
                        ]},
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
```

Listing: Verbose JSON-Formatted Customer Entity with Collection Properties

2.2.6.3.14 Named Resource Streams

When an EntityType includes named resource streams, any representations of whole **EntityType** instances MUST include named resource stream instances by using the grammar listed in this section. The grammar rule "namedStreamInVJson" specifies how a named resource stream instance is

represented. This rule is referenced, in turn, by the rule "entityPropertyInVJson" that specifies how **EntityType** instance members MUST be represented when representing an **EntityType** instance in Verbose JSON. See Entity Type (as a JSON object) (section 2.2.6.3.3) for more information on how **EntityType** instances are represented in Verbose JSON.

```
namedStreamInVJson = quotation-mark entityNamedStream quotation-mark
                     name-seperator
                     entityNamedStreamInVJson

entityNamedStreamInVJson = begin-object mediaResourceNVP end-object

mediaResourceNVP =  quotation-mark "__mediaresource" quotation-mark
                    name-seperator
                    begin-object
                    mleMetadata
                    [value-seperator etagNVP]
                    end-object
                    ; defined in section 2.2.6.3.3

mleMetadata      =
                    mediaSrcNVP
                    value-seperator contentTypeNVP
                    [value-seperator editMediaNVP]
                    ; defined in section 2.2.6.3.3

etagNVP          =  quotation-mark "media_etag" quotation-mark
                    name-seperator
                    quotation-mark entityTag quotation-mark
                    ; defined in section 2.2.6.3.3

editMediaNVP     =  quotation-mark "edit_media" quotation-mark
                    name-seperator
                    quotation-mark resourcePath quotation-mark
                    ; defined in section 2.2.6.3.3

mediaSrcNVP      =  quotation-mark "media_src" quotation-mark
                    name-seperator
                    quotation-mark resourcePath quotation-mark
                    ; defined in section 2.2.6.3.3

contentTypeNVP   =  quotation-mark "content_type" quotation-mark
                    name-seperator
                    quotation-mark contentType quotation-mark
                    ; defined in section 2.2.6.3.3

entityNamedStream = ; see section 2.2.3.6.1.11
entityTypeProperty=;  see section 2.2.6.3.3
```

When a named resource stream is present on the declaring **EntityType** and not excluded from the results explicitly by using **$select**, the named resource stream instance MUST be present in the Verbose JSON representation.

The "media_src" and "content_type" name/value pairs MUST be included.

The "edit_media" name/value pair MAY be included if the named resource stream instance can be updated.

The "media_etag" name/value pair MAY contain an ETag (ETag (section 2.2.5.4)). When the ETag is included, the value MUST be the value of the ETag for the named resource stream instance.

The value of the "edit_media" name/value pair MUST be a URI that can be used to replace the existing stream with a HTTP PUT request. The value of the "media_src" name/value pair MUST be a URI that can be used to retrieve the stream of bytes with a GET request.

The value of the "content_type" name/value pair MUST specify the content type of the binary stream (as specified in [RFC2616]) that is represented by the "edit_media" URI . If the "media_etag" name/value pair is present, the value MUST be the ETag (ETag (section 2.2.5.4)) value for the named resource stream retrieved from the "edit_media" URI.

```
{
   "__metadata": { "uri": "Photos(1)",
                   "type": "SampleModel.Photo",
                 },
   "ID": 1,
   "Name": "Mount Fuji",
   "Thumbnail": {
        "__mediaresource": {
             "edit_media": " http://server/uploads/Thumbnail546.jpg ",
             "media_src": "http://server/Thumbnail546.jpeg ",
             "content-type": "img/jpeg",
             "media_etag": "####"
        }
   },
   "PrintReady": {
        "__mediaresource": {
             "edit_media": "Photos(1)/PrintReady",
             "media_src": "Photos(1)/PrintReady ",
             "content-type": "img/png",
        }
   }
}
```

Listing: Entity with Named Resource Streams Formatted in Verbose JSON

2.2.6.3.15 Links and Subtypes

Entity instances in an **EntitySet** MAY not belong to the same **EntityType** but MUST all be derived from the **EntityType** baseType associated with the **EntitySet**. In the OData 3.0 protocol, in addition to the rules described in Entity Type (as a JSON object) (section 2.2.6.3.3), the value of the **ResourcePath** for the following grammar rules (described in Entity Type (as a JSON object) (section 2.2.6.3.3),) SHOULD include the **EntityType** instance (as described in Resource Path (resourcePath) (section 2.2.3.3)):

- uriNVP
- associationuri
- editMediaNVP
- mediaSrcNVP

2.2.6.3.16 Annotations

Instance annotations are not supported in the Verbose JSON format.

2.2.6.4 Raw Format

The data service URI addressing scheme, as specified in URI Format: Resource Addressing Rules (section 2.2.3), enables directly addressing the "raw" value (see URI 4 and URI 5 in Resource Path: Semantics (section 2.2.3.5)) of **EDMSimpleType** properties defined on an **EntityType** or

ComplexType. This allows the constituent parts of an **EntityType** to be identified independent of the rest of the **EntityType** and without any wrapping syntax.

2.2.6.4.1 EDMSimpleType Property

By default, the raw value (identified via URIs with resource paths ending in "$value") of any **EDMSimpleType** property (except those of type **Edm.Binary**) SHOULD be represented using the text/plain media type and MUST be serialized as specified in Common Serialization Rules for XML-based Formats (section 2.2.6.1). A **$value** request for a property that is NULL SHOULD result in a "404 Not Found" response. A data service MAY<64> customize the media type used for any property. The raw value of an **Edm.Binary** property MUST be an unencoded byte stream.

If the value of the property to be serialized is null, see Common Serialization Rules for XML-based Formats (section 2.2.6.1), because the representation is format specific.

2.2.6.5 XML Format

The data service URI addressing scheme, specified in URI Format: Resource Addressing Rules (section 2.2.3), enables the constituent parts of an **EntityType** and associations between **EntityTypes** to be identified directly. This allows interaction with a specific piece of data or relationship, independent of the rest of the **EntityType**. Servers responding to requests that identify a constituent part of an **EntityType** instance MUST respond with an XML-based serialization of that part's value, as specified in this section, unless the request URI's resource path ends in "$value" (in which case it is to use the format defined in Raw Format (section 2.2.6.4)).

The serializations defined in the following subsections MUST be identified with the application/xml media type or text/xml media types.

2.2.6.5.1 Complex Type

A ComplexType property, defined on an EntityType, MUST be represented in the same way as it is within in the Atom-based format, as specified in section Complex Type (section 2.2.6.2.3); however, the XML element representing the **ComplexType** instance as a whole MUST be the root of the XML document (for example, not a child element, as described in section Complex Type (section 2.2.6.2.3)). For example, the Address property of type CAddress (a **ComplexType**) in the sample model (see Appendix A: Sample Entity Data Model and CSDL Document (section 6)) is represented in the following listing.

If the value of the **ComplexType** is null, the element MUST be empty and MUST include the **m:null** attribute set to true, as required by the Common Serialization Rules for XML-based Formats (section 2.2.6.3.1).

```
<?xml version="1.0" encoding="utf-8"?>
<Address xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">
  <Street>57 Contoso St</Street>
  <City>Seattle</City>
</Address>
```

Listing: XML-formatted Complex Type

2.2.6.5.2 Collection of Complex Type Instances

A collection of ComplexType instances is represented in XML as a single XML document with the root element of the document equal to the same name of the service operation returning the **ComplexType** instances. The root element and all its child elements MUST exist in the Data Service Metadata namespace, as specified in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

Each **ComplexType** instance in the collection is represented as a child element of the root element and named "element". An attribute named "type" (in the Data Service Metadata namespace, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1), MUST exist on the element. The value of this attribute specifies the namespace qualified type name of the **ComplexType**.

Each property of the **ComplexType** instance is represented in the same way as in the XML serialization of a single **ComplexType**, as described in Complex Type (section 2.2.6.2.3).

2.2.6.5.3 EDMSimpleType Property

Properties of type EDMSimpleType MUST be represented as a single (root) XML element in the data service namespace, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1), with the same name as the property. The element MAY have an **m:type attribute** to specify the Entity Data Model (EDM) type of the property. If the **m:type** attribute is missing, the EDM type of the property MUST be assumed to be Edm.String. The text value of the element MUST be equal to the value of the property. The property value is formatted, as described in the EDM Primitive Type Formats for XML Element Values table in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

2.2.6.5.4 Collection of EDMSimpleType Values

A collection of EDMSimpleType values MUST be represented as a single XML document with the root element of the document equal to the name of the service operation, action, or function returning the values. The root element and all its child elements MUST exist in the Data Service Namespace, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

Each value in the collection is represented as a child element of the root element and be named "element". The text value of the XML element is formatted, as described in the EDM Primitive Type Formats for XML Element Values table in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

2.2.6.5.5 Links

Links represent unidirectional associations or one direction of a bidirectional association between EntityTypes. Using the Customer and Order **EntityTypes**, associations, and instance sample data, as described in Appendix A: Sample Entity Data Model and CSDL Document (section 6), the set of links from the Customer with EntityKey "ALFKI" to Order instances are represented by a set of URIs, with each URI in the set identifying a single Order that is linked to the Customer. Such link information MUST be serialized as an XML document that conforms to the XML schema [XMLSCHEMA1/2] shown in the following XML Schema for a Set of Links Represented Using XML listing. In the serialization, one URI element MUST exist for each link, with the text value of the element equal to the canonical URI of the linked-to EntityType instance. Additionally, the target namespace MAY be server-specific, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://schemas.microsoft.com/ado/2007/08/dataservices">
  <xsd:element name="links">
    <xsd:complexType mixed="false">
      <xsd:sequence>
        <xsd:element name="uri" type="xsd:string" minOccurs="0"
                     maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing: XML Schema for a Set of Links Represented Using XML

A single link, which is not part of a set, MUST be serialized as an XML document that conforms to the XML schema [XMLSCHEMA1/2] shown in following XML Schema for a Single Link Represented Using XML listing. The definition of the URI element in this case is unchanged from above. The targetNamespace in the XSD MAY be server-specific, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://schemas.microsoft.com/ado/2007/08/dataservices">
   <xsd:element name="uri" type="xsd:string" minOccurs="1"
                maxOccurs="1"/>
</xsd:schema>
```

Listing: XML Schema for a Single Link Represented Using XML

For example, using the sample model and instance data, as described in Appendix A: Sample Entity Data Model and CSDL Document (section 6), the links from the Customer with EntityKey entity "ALFKI" to Order instances are represented as follows.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<links xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">
  <uri>http://host/service.svc/Orders(1)</uri>
  <uri>http://host/service.svc/Orders(2)</uri>
</links>
```

Listing: Example of Links Formatted as XML

2.2.6.5.5.1 InlineCount Representation (for Collections of Links)

Applies to the OData 2.0 and OData 3.0 protocols

This section defines an extended representation of a collection of links from that described in section 2.2.6.5.5. This representation of a collection of links is supported only in the OData 2.0 and OData 3.0 protocols.

A request URI MAY contain an **$inlinecount** system query option to indicate that the count of the number of links represented by the query is to be included in the collection of links returned by a data service.

The count value included in the result MUST be enclosed in an **m:count** element which MUST be the first child element of the root **links** element.

2.2.6.5.5.2 Next Page (for Collections of Links)

A response containing a collection of links MAY include a **next** element in the Data Service Namespace (section 2.2.6.1) as the final child element of the root **links** element if the server only returned a subset (or single page) of the collection of links.

The value of the **next** element MUST be a URI that identifies the next page of the links in the collection; often the URI will include a Skip Token system query option (section 2.2.3.6.1.9).

2.2.6.5.6 Collection of Complex Type

In the OData 3.0 protocol, a collection property of complex types, defined on an **EntityType**, MUST be represented in the same way that it is in the Atom-based format, as specified in section Collection Property (section 2.2.6.2.9). However, the XML element that represents the collection instance as a whole MUST be the root of the XML document (not a child element, as described in Collection Property of Complex Type (section 2.2.6.2.9.1)). For example, the Address property of type AlternateAddresses

(a **ComplexType**) in the sample model (see Appendix A: Sample EDM and CSDL Document (section 6)) is represented in the following listing.

```
<?xml version="1.0" encoding="utf-8"?>
<d: AlternateAddresses m:type="Collection(SampleModel.Address)"
             xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
             xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <d:element m:type="Collection(SampleModel.EAddress)>
        <d:Street>123 contoso street</d:Street>
    </d:element>
    <d:element >
        <d:Street>834 1st street</d:Street>
        <d:Apartment>102</d:Apartment>
    </d:element>
</d: AlternateAddresses>
```

Listing: XML Formatted Collection of Complex Type

2.2.6.5.7 Collection of EDMSimpleType

In the OData 3.0 protocol, a collection property of simple types, defined on an **EntityType**, MUST be represented in the same way that it is in the Atom-based format, as specified in section 2.2.6.2.9.2. However, the XML element that represents the collection instance as a whole MUST be the root of the XML document (not a child element, as described in section 2.2.6.2.9.2. For example, the EmailAddress property of type string (an **EDMSimpleType**) in the sample model (see Appendix A: Sample EDM and CSDL Document (section 6)) is represented in the following listing.

```
<?xml version="1.0" encoding="utf-8"?>
<d:EmailAddresses m:type="Collection(Edm.String)"
                 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
                 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <d:element>alternate1@company.com</d:element>
    <d:element>alternate2@company.com</d:element>
</d:EmailAddresses>
```

Listing: XML Formatted Collection of EDMSimpleType

2.2.6.6 Preferred OData 3.0 JSON Format

OData 3.0 services SHOULD<65> support the preferred OData 3.0 JSON format as defined in [MS-ODATAJSON].

2.2.7 Request Types

This document defines requests that a client can send to a data service.

The request types defined in this document either extend the request types defined in AtomPub, as specified in [RFC5023], by providing additional rules for each type. Or, they add additional types, in addition to those defined in [RFC5023].

In general, this document adopts the protocol semantics of AtomPub, as specified in [RFC5023], but extends AtomPub to allow the use of alternate formats (such as JSON as specified in [RFC4627]), in addition to Atom, and defines a URI addressing scheme for the abstract data model used in this document. This document's data model maps 1-to-1 to the model constructs defined in AtomPub and defines additional constructs not present in the AtomPub model.

As specified in [RFC5023] and extended in this section, the requests from the client and the corresponding responses from the server are exchanged using HTTP request methods. Each request

type defined is mapped to an HTTP request method (for example, GET, POST, and so on) and HTTP request URI pair.

In general, the request types defined allow clients to:

- Retrieve, edit, and delete Entity Data Model (EDM) constructs represented as data service or AtomPub resources using HTTP's GET, PUT, PATCH, and DELETE methods. For details, see Retrieve Request Types (section 2.2.7.2), Update Request Types (section 2.2.7.3), and Delete Request Types (section 2.2.7.4).
- Insert new EntityType instances into an EntitySet represented as an AtomPub Collection. See Insert Request Types (section 2.2.7.1) for details.
- Invoke a data service service operation. For further details see Invoke Request (section 2.2.7.5).
- Package many requests using a batch request type. See Batch Request (section 2.2.7.6) for details.
- Issue any of the above non-batch operations using a technique commonly referred to as POST tunneling (Tunneled Requests (section 2.2.7.7)).

This section defines the syntax rules for each request type. Any ABNF syntax rules that are not specified in [RFC5234] or [RFC4627] use the extensions defined in [RFC2616]. The following are common ABNF syntax rules used throughout this section.

```
HTTP-Header-Types   = *((general-header
                         ; see section 4.5 of [RFC2616]
                         / request-header
                         ; see section 5.3 of [RFC2616]
                         / entity-header) CRLF )
                         ; see section 7.1 of [RFC2616]
```

Listing: Common Grammar Rules for Request Types

2.2.7.1 Insert Request Types

This section defines all the insert request types a client can send to a data service. All insert requests use the HTTP POST request method. The type of insert action is further defined by the request URI used in a POST request.

InsertEntity Request (section 2.2.7.1.1) defines the InsertEntity request type that enables a client to insert a new **EntityType** instance into an **EntitySet**.

InsertLink Request (section 2.2.7.1.2) defines the InsertLink request type that is used to add a new link between **EntityTypes** instances.

2.2.7.1.1 InsertEntity Request

The purpose of the InsertEntity request is to enable a new EntityType instance, potentially with new related entities, to be inserted into an EntitySet. The base rules and semantics of this request type are defined by AtomPub, as specified in [RFC5023] section 5.3 – Creating a Resource, and, as described in Abstract Data Model (section 2.2.1), Entity Data Model (EDM) constructs are mapped directly to data model concepts used in AtomPub. For example, **EntityTypes** are AtomPub Entry Resources and collections of entities (entity sets and so on) are AtomPub collections. This section adds constraints to those defined in AtomPub for this request type.

As specified in [RFC5023] section 9.2, insert requests use the HTTP POST method and the request URI has to represent an AtomPub collection. Because a collection maps to a conceptual schema definition

language (CSDL) in an EDM, the HTTP request line URI MUST be any valid data service URI, as defined in URI Format: Resource Addressing Rules (section 2.2.3), which identifies a collection of entities.

[RFC5023] section 9.2 states that the request body in the POST might be an AtomPub Entry Resource (which maps to an **EntityType** instance in an EDM) represented as an Atom Entry document. This document extends this rule to allow additional representations of an Entry Resource to be posted to a URI representing a collection. This document defines three such representations of **EntityTypes** that map to Entry Resources: AtomPub as defined in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2), JSON as defined in Entity [ODataJSON4.0] section 6, and Verbose JSON as defined in Entity Type (as a Verbose JSON Object) (section 2.2.6.3.3).

When the request body is a representation of an Entry Resource (Entity Type in EDM terms), the client MAY specify whether the resource/entity is to be automatically linked to other already existing entities in the data service. For example, a new order entity might need to be bound to an existing customer entity in a customer relationship management focused data service. Such linking is to be supported only if the **EntityType** of the to-be-inserted entity defines a NavigationProperty which associates the new entity and the to-be-related entity.

To bind the new entity to one or more (as defined by the cardinality of the **NavigationProperty**) existing entities, the client MUST include the required binding information in the representation of the associated **NavigationProperty** in the request payload.

To bind the new entity to an existing entity by using the Atom format, the **NavigationProperty** MUST be represented, as specified in Navigation Property (section 2.2.6.2.4), with one exception: the **href** attribute of the **atom:link** element has to represent the URI of the entity to be linked to.

To bind the new entity to an existing entity by using the preferred OData 3.0 JSON format, see [ODataJSON4.0] section 8.5.

To bind the new entity to an existing entity by using the Verbose JSON format, the **NavigationProperty** MUST be represented using the inline representation of a **NavigationProperty**, as specified in Inline Representation (section 2.2.6.3.9.1), with the inlined entities represented using only the "__metadata" name/value pair (the properties of each inlined entity SHOULD NOT be provided).

In addition to supporting the insertion of a new **EntityType** instance (E1) into an **EntitySet**, this request type allows inserting new entities related to E1 (described by a **NavigationProperty** on the **EntityType** associated with E1) using a single InsertEntity request. For example, in a customer relationship management focused data service, a new customer entity and new related order entities could be inserted by using a single InsertEntity request. This form of an InsertEntity request is also known as a "deep insert".

To insert a new **EntityType** instance (E1) and related entities, the related entities MUST be represented using the inline representation of the **NavigationProperty** (associated with E1) that identifies the link to the (to-be-inserted) related entities, as described in Inline Representation (section 2.2.6.3.9.1) and Inline Representation (section 2.2.6.2.6.1).

The syntax of an InsertEntity request is defined as follows.

```
insertEntity-Req         = insertEntity-ReqLine
                           insertEntity-ReqHeaders
                           CRLF
                           insertEntity-ReqBody

insertEntity-ReqLine     = "POST"
                           SP entitySetUri insertEntity-QueryOps
                           SP HTTP-Version CRLF

insertEntity-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                           [MaxDataServiceVersion]     ; see section 2.2.5.7
                           [Content-Type]              ; see section 2.2.5.2
```

```
                              [Prefer]                    ; see section 2.2.5.9
                              *(HTTP-Header-Types)

entitySetUri              = <Any Resource Path which identifies collection of
                             entities>
                            ; see section 2.2.3 and section 2.2.3.5 -- URI1 & URI6

insertEntity-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
                            ; see section 2.2.3.1

insertEntity-ReqBody      = <Entity in JSON format as per [ODataJSON4.0] section 6>
                            / <Entity Type in Verbose JSON format as per section 2.2.6.3.3>
                            / <Entity Type in Atom format as per section 2.2.6.2.2>
                            / <Entity Type in Atom Format with Customizable Feeds Property
                               Mapping as per section 2.2.6.2.2.1>
```

The syntax of a response to a successful InsertEntity request is defined as follows.

```
insertEntity-Resp         = Status-Line    ; see [RFC2616] section 6.1.1
                            insertEntity-RespHeaders
                            CRLF
                            insertEntity-RespBody

insertEntity-RespHeaders  = DataServiceVersion   ; see section 2.2.5.3
                            [ETag]               ; see section 2.2.5.4
                            [Preference-Applied] ; see section 2.2.5.10
                            [DataServiceId]      ; see section 2.2.5.11

                            <Location header, described in [RFC5023] section 9.2>
                            "Content-Type: "
                            <One of the Media types which defines a representation of
                             an Entity Type>     ; see section 2.2.6
                            *(HTTP-Header-Types)

insertEntity-RespBody     = <Entity in JSON format as per [ODataJSON4.0] section 6>
                            / <Entity Type in Atom format as per section 2.2.6.2.2>
                            / (begin-object
                              quotation-mark "d" quotation-mark
                              name-seperator
                              entityTypeInVJson
                              end-object)
                             ; see section 2.2.6.3.3
```

The syntax of an error response is shown in Error Response (section 2.2.8.1).

2.2.7.1.1.1 Examples

See Appendix A: Sample Entity Data Model and CSDL Document (section 6) for the sample model and data used in this section.

Example 1: Insert a new Customer and bind it to existing Orders with key values 1 and 2 by using the Atom format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml
DataServiceVersion: 1.0
Accept: application/atom+xml
Content-Length: nnn

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
```

```
     xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
     xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
     xmlns="http://www.w3.org/2005/Atom">
  <id>http://host/service.svc/Customers('ASDFG')</id>
  <title type="text" />
  <updated>2008-12-07T8:00:00Z</updated>
  <author>
    <name />
  </author>
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/related/Orders"
        href="Orders(1)" />
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/related/Orders"
        href="Orders(2)" />
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ASDFG</d:CustomerID>
    <d:CompanyName>Contoso Widgets</d:CompanyName>
    <d:Address>
      <d:Street>58 Contoso St</d:Street>
      <d:City>Seattle</d:City>
    </d:Address>
   </m:properties>
  </content>
</entry>
```

HTTP Response:

```
HTTP/1.1 201 Created
Date: Fri, 12 Dec 2008 17:17:11 GMT
Location: http://host/service.svc/Customers('ASDFG')
Content-Type: application/atom+xml;type=entry
DataServiceVersion: 3.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08
            /dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ASDFG')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ASDFG')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ASDFG')/Orders" />
       xmlns:d="http://schemas.microsoft.com/ado/2007/08
/dataservices/relatedlinks  /dataservices/related/Orders"
        type="application/xml"
        title="Orders"
        href="Customers('ASDFG')/$links/Orders" />

  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ASDFG</d:CustomerID>
    <d:CompanyName>Contoso Widgets</d:CompanyName>
    <d:Address>
```

```
      <d:Street>58 Contoso St</d:Street>
      <d:City>Seattle</d:City>
    </d:Address>
    <d:Version>AAAAAAAA+gE=</d:Version>
   </m:properties>
  </content>
</entry>
```

Example 2: Insert a new Customer and bind it to existing Orders with key values 1 and 2 by using the Verbose JSON format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
Content-Length: nnn

{
   "__metadata": { "uri": "Customers(\'ASDFG\')" },
   "CustomerID": "ASDFG",
   "CompanyName": "Contoso Widgets",
   "Address": { "Street": "58 Contoso St", "City": "Seattle" },
   "Orders": [
      { "__metadata": {"uri": "Order(1)"} },
      { "__metadata": {"uri": "Order(2)"} }
   ]
}
```

HTTP Response (Version 3.0):

```
HTTP/1.1 201 Created
Date: Fri, 12 Dec 2008 17:17:11 GMT
Location: http://host/service.svc/Customers('ASDFG')
Content-Type: application/json;odata=verbose
DataServiceVersion: 3.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

{
 "d":
    {
      "__metadata": { "uri": "Customers(\'ASDFG\')",
                      "type": "SampleModel.Customer",
                      "etag": "W/\"X\'000000000000FA01\'\"",
                            "properties" : {
                              "Orders" : {
                                "associationuri" : "Customers(\'ASDFG\')/$links/Orders"
                              }
                            }

                    },
      "CustomerID": "ASDFG",
      "CompanyName": "Contoso Widgets",
      "Address": { "Street": "58 Contoso St", "City": "Seattle" },
      "Version": "AAAAAAAA+gE=",
      "Orders":  { "__deferred": { "uri": "Customers(\'ASDFG\')/Orders"} }
    }
}
```

Example 3: Insert a new Customer and two new related orders by using the Atom format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml
DataServiceVersion: 1.0
Accept: application/atom+xml
Content-Length: nnn

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://host/service.svc/Customers('ASDFG')</id>
  <title type="text" />
  <updated>2008-03-30T21:52:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/related/Orders"
        href="Customers('ASDFG')/Orders">
    <m:inline>
      <feed>
        <title type="text">Orders</title>
        <id>http://host/service.svc/Customers('ASDFG')/Orders</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Orders"
        href="Customers('ASDFG')/Orders" />
        <entry>
          <id>http://host/service.svc/Orders(3)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <content type="application/xml">
           <m:properties>
            <d:OrderID>3</d:OrderID>
            <d:ShippedDate>
               2008-03-30T21:52:45Z</d:ShippedDate>
           </m:properties>
          </content>
        </entry>
        <entry>
          <id>http://host/service.svc/Orders(4)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <content type="application/xml">
           <m:properties>
            <d:OrderID>4</d:OrderID>
            <d:ShippedDate>2008-03-30T21:52:45Z</d:ShippedDate>
           </m:properties>
          </content>
       </entry>
      </feed>
    </m:inline>
   </link>
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ASDFG</d:CustomerID>
    <d:CompanyName>Contoso Widgets</d:CompanyName>
    <d:Address>
      <d:Street>58 Contoso St</d:Street>
      <d:City>Seattle</d:City>
```

```
    </d:Address>
   </m:properties>
  </content>
</entry>
```

HTTP Response (Version 3.0):

```
HTTP/1.1 201 Created
Date: Fri, 12 Dec 2008 17:17:11 GMT
Location: http://host/service.svc/Customers('ASDFG')
Content-Type: application/atom+xml;type=entry
DataServiceVersion: 3.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08
            /dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ASDFG')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ASDFG')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ASDFG')/Orders" />
  <link rel="http://schemas.microsoft.com/ado/2007/08
  /dataservices/relatedlinks/Orders"
        type="application/xml"
        title="Orders"
        href="Customers('ASDFG')/$links/Orders" />
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ASDFG</d:CustomerID>
    <d:CompanyName>Contoso Widgets</d:CompanyName>
    <d:Address>
      <d:Street>58 Contoso St</d:Street>
      <d:City>Seattle</d:City>
    </d:Address>
    <d:Version m:type="Edm.Binary">AAAAAAAA+gE=</d:Version>
   </m:properties>
  </content>
</entry>
```

Example 4: Insert a new Customer and two new related orders by using the Verbose JSON.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
Content-Length: nnn

{
```

```
   "CustomerID": "ASDFG",
   "CompanyName": "Contoso Widgets",
   "Address": { "Street": "58 Contoso St", "City": "Seattle" },
   "Orders": [
         {
            "OrderID": 1,
            "ShippedDate": "\/Date(872467200000)\/"
         },
         {
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/"
         }
   ]
}
```

HTTP Response (Version 3.0):

```
HTTP/1.1 201 Created
Date: Fri, 12 Dec 2008 17:17:11 GMT
Location: http://host/service.svc/Customers('ASDFG')
Content-Type: application/json;odata=verbose
DataServiceVersion: 3.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

{"d":
 {
   "__metadata": { "uri": "Customers(\'ASDFG\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\"",
                   "properties" : {
                              "Orders" : {
                                "associationuri" : "Customers(\'ASDFG\')/$links/Orders"
                              }
                            }

                 },
   "CustomerID": "ASDFG",
   "CompanyName": "Contoso Widgets",
   "Address": { "Street": "58 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ASDFG\')/Orders"} }
 }
}
```

2.2.7.1.2 InsertLink Request

The purpose of the InsertLink request is to enable a new link to be created between two EntityType instances. AtomPub, as specified in [RFC5023], does not define a request of this type. Therefore, this request type is not based on an AtomPub-defined request, as specified in [RFC5023].

An InsertLink Request MUST use the HTTP POST method and the URI specified by the client in the HTTP request line MUST be a valid data service URI, as specified in URI Format: Resource Addressing Rules (section 2.2.3), which identifies the collection of links between an **EntityType** instance and an EntitySet. The **EntitySet** MUST be related to the instance by a NavigationProperty on the instance's **EntityType**, as specified in the URI7 grammar rule in Resource Path: Semantics (section 2.2.3.5). For example, by using Appendix A: Sample Entity Data Model and CSDL Document (section 6), the following is a valid URI for requests of this type:
`http://host/service.svc/Customers('ALFKI')/$links/Orders`.

In this context, the **EntityType** instance that is identified by the resource path segment immediately prior to the "$links" segment in the request URI is referred to as the source entity. Requests of this type MUST contain a request body (formatted as a link according to [MS-ODATAJSON] section 2.1.23,

formatted according to the "linkVJson" rule in EDMSimpleType Property (section 2.2.6.3.8), or formatted according to the XML schema for a single link in **EDMSimpleType** property) that contains a URI that identifies the entity to be linked to from the source entity.

If the new link defined by a request of this type represents one direction of a bidirectional association, inserting the link (one direction of the bidirectional association) implies the opposite direction is also inserted.

If an InsertLink request is successful, the response MUST have a 204 status code, as specified in [RFC2616], and contain an empty response body.

If the InsertLink request is not successful (an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an InsertLink request is defined as follows.

```
insertLink-Req          =   insertLink-ReqLine
                            insertLink-ReqHeaders
                            CRLF
                            insertLink-ReqBody

insertLink-ReqLine      =   "POST"
                            SP entityTypeInstanceLinksUri insertLink-QueryOps
                            SP HTTP-Version
                            CRLF

insertLink-ReqHeaders   =   [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            [Content-Type]              ; see section 2.2.5.2
                            [Prefer]                    ; see section 2.2.5.9
                            *(HTTP-Header-Types)

insertLink-ReqBody      =   <JSON representtation of a single link per [MS-ODATAJSON]
                             section 2.1.23>
                            / linkVJson                    ; see section 2.2.6.3.10
                            / <XML Representation of a single link as per the XML
                               Schema in section 2.2.6.5.5>

entityTypeInstanceLinksUri = <Any Resource Path identifying a collection of Links
                              where the final URI segment is a navProperty
                              (section 2.2.3.1) representing an Entity Set>
                              ; see section 2.2.3.5 -- URI7

insertLink-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
                            ; see section 2.2.3.1
```

The syntax of a response to a successful InsertLink request is defined as follows.

```
insertLink-Resp         =    Status-Line              ; see [RFC2616] section 6.1.1
                             insertLink-RespHeaders
                             CRLF

insertLink-RespHeaders  =    DataServiceVersion       ; see section 2.2.5.3
                             [ETag]                   ; see section 2.2.5.4
                             [Preference-Applied]     ; see section 2.2.5.10
                             *(HTTP-Header-Types)
```

2.2.7.1.3 InsertMediaResource Request

The purpose of the InsertMediaResource request is to enable a BLOB (in other words, a media resource) along with an associated EntityType instance which, potentially with new related entities, is to be inserted into an EntitySet. The base rules and semantics of this request type are defined by

AtomPub, as specified in [RFC5023] section 9.6 - Media Resources and Media Link Entries, and, as described in Abstract Data Model (section 2.2.1), Entity Data Model (EDM) constructs are mapped directly to data model concepts used in AtomPub. If an **EntityType** instance is created via an InsertMediaResource request, the created entity represents an AtomPub Media Link Entry (MLE) and the associated BLOB represents the media resource described by the MLE.

The remainder of this section adds constraints to those defined in AtomPub for this request type.

As specified in [RFC5023] section 9.6, insert requests of this type use the HTTP POST method and the request URI has to represent an AtomPub collection. Because a collection maps to an **EntitySet** (that is, a collection of entities) in an EDM, the HTTP request line URI MUST be any URI that identifies a collection of entities. If the **EntityType** associated with the implicitly created MLE defines a concurrency token, a successful response MUST contain an ETag header with a value equal to the value of the concurrency token of the MLE created.

The syntax of an InsertMediaResource request is defined as follows.

```
insertMR-Req             = insertEntity-ReqLine     ; section 2.2.7.1.1
                           insertMR-ReqHeaders
                           CRLF
                           insertMR-ReqBody

insertMR-ReqHeaders  =   [DataServiceVersion]    ; see section 2.2.5.3
                         [MaxDataServiceVersion] ; see section 2.2.5.7
                         [Accept]                ; see section 2.2.5.1
                         [Content-Type]          ; see section 2.2.5.2
                         [Prefer]                ; see section 2.2.5.9
                         [<Slug header as defined in [RFC5023] section 9.7>]
                         *(HTTP-Header-Types)


entitySetUri             = <Any data service URI which identifies a collection of entities>

insertEntity-QueryOps    = ; section 2.2.7.1.1

insertMR-ReqBody         = <Any valid HTTP request body> ; see [RFC5023] section 9.6
```

The syntax of a response to a successful InsertMediaResource request is defined as follows.

```
insertMR-Resp             = Status-Line          ; see [RFC2616] section 6.1.1
                            insertEntity-RespHeaders
                            CRLF
                            insertEntity-RespBody

insertMR-RespHeaders      = DataServiceVersion   ; see section 2.2.5.3
                            [ETag]               ; see section 2.2.5.4
                            <Location header, as described in [RFC5023] section 9.6>
                            "Content-Type: "
                            <One of the Media types which defines a representation of
                             an Entity Type>     ; see section 2.2.6
                            [Preference-Applied] ; see section 2.2.5.10
                            [DataServiceId]      ; see section 2.2.5.11
                            *(HTTP-Header-Types)

insertEntity-RespBody     = <Media Entity in JSON as per [ODataJSON4.0] section 10>
                            / <Entity Type representing a Media Link Entry in Atom format as
                              per section 2.2.6.2.2>
                            / (begin-object
                               quotation-mark "d" quotation-mark
                               name-seperator
                               entityTypeInVJson
                               end-object)
                              ; see section 2.2.6.3.3
                              ; the entityTypeInVJson representation MUST include all name
```

```
                              ; value/pairs denoted in the mleMetadata rule
```

The syntax of an error response is shown in Error Response (section 2.2.8.1).

2.2.7.2 Retrieve Request Types

2.2.7.2.1 RetrieveEntitySet Request

A RetrieveEntitySet request is used by a client to retrieve the entries in an AtomPub collection, as specified in [RFC5023], that maps to an EntitySet in the abstract data model used in this document, as described in Abstract Data Model (section 2.2.1). The base rules and semantics of this request type are defined by AtomPub, as specified in [RFC5023] section 5.2 -- Listing Collection Members. This section adds constraints to those defined in AtomPub for this request type.

According to [RFC5023] section 5.2, requests of this type MUST use the HTTP GET method and the URI specified by the client in the HTTP request line has to represent an AtomPub collection. Because a collection maps to an **EntitySet** in an Entity Data Model (EDM), the HTTP request line URI MUST be equal to any valid data service URI that identifies an **EntitySet**, as specified in URI Format: Resource Addressing Rules (section 2.2.3).

[RFC5023] section 9.2 states that the response body from such a request has to be an Atom Feed document. This document extends this rule to allow additional representations of a collection (or **EntitySet**) to be retrieved by a data service client. This document defines three representations of **EntitySet**: AtomPub as specified in Entity Set (as an Atom Feed Element) (section 2.2.6.2.1), Collection of Entities as specified in [ODataJSON4.0] section 12, and Verbose JSON as specified in Entity Set (as a Verbose JSON Array) (section 2.2.6.3.2).

If a RetrieveEntitySet request was successful, the response payload MUST contain the requested representation of the entities in the **EntitySet** identified in the request URI. The payload of such a response MUST be formatted using AtomPub, JSON, or Verbose JSON, according to the rules defined in AtomPub Format (section 2.2.6.2), JSON format [ODataJSON4.0] section 12, and Verbose JSON Format (section 2.2.6.3), respectively.

If the RetrieveEntitySet request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveEntitySet request is defined as follows:

```
retrieveEntitySet-Req       = retrieveEntitySet-ReqLine
                              retrieveEntitySet-ReqHeaders
                              CRLF

retrieveEntitySet-ReqLine   = "GET"
                              SP entitySetUri retrieveEntitySet-
                                 QueryOps
                              SP HTTP-Version
                              CRLF

retrieveEntitySet-ReqHeaders = [DataServiceVersion]
                               ; see section 2.2.5.3
                               [MaxDataServiceVersion]
                               ; see section 2.2.5.7
                               [Accept]
                               ; see section 2.2.5.1
                               *(HTTP-Header-Types)

entitySetUri                  = ; see section 2.2.7.1.1

retrieveEntitySet-QueryOps    = ["?" (customQueryOption /sysQueryOption)
                                  *("&"customQueryOption / sysQueryOption)]
```

```
                                ; see section 2.2.3.1 & section 2.2.3.6.1.5
```

The syntax of a response to a successful RetrieveEntitySet request is defined as follows:

```
retrieveEntitySet-Resp       = Status-Line
                               ; see [RFC2616] section 6.1.1
                                 retrieveEntitySet-RespHeaders
                                 CRLF
                                 retrieveEntitySet-RespBody

retrieveEntitySet-RespHeaders= DataServiceVersion
                               ; see section 2.2.5.3
                               [Content-Type]
                               ; see section 2.2.5.2
                               [ETag]
                               ; see section 2.2.5.4
                               *(HTTP-Header-Types)

retrieveEntitySet-RespBody   =  <Collectons of entities in JSON as per [ODataJSON4.0]
                                 section 12>
                                / <Entity Set formatted as per section
                                2.2.6.2.1>
                                / (begin-object
                                   quotation-mark "d" quotation-mark
                                   name-seperator
                                   (entitySetInVJson / entitySetInVJson2)
                                   end-object)
                                   ; see section 2.2.6.3.2
```

2.2.7.2.2 RetrieveEntity Request

A RetrieveEntity request is used by a client to retrieve an AtomPub entry resource, as specified in [RFC5023], and potentially related entities that map to EntityType instances, as described in Abstract Data Model (section 2.2.1).

Requests of this type MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST represent an AtomPub entry resource. Because an entry resource maps to an **EntityType** in an Entity Data Model (EDM), the HTTP request line URI MUST be any valid data service URI that identifies an **EntityType** instance, as defined in URI Format: Resource Addressing Rules (section 2.2.3). A client will typically obtain such a URI after parsing one of the **EntityType** instances serialized in the response payload from a prior RetrieveEntitySet request, as specified in Retrieve Request Types (section 2.2.7.2). When using the AtomPub format in a RetrieveEntitySet request, such a URI is typically obtained from the "edit" or "self" URIs described in AtomPub, as specified in [RFC5023] section 11.1 (The "edit" Link Relation), [RFC4287] section 4.2.7.2, and Entity Type (as an Atom Entry Element) (section 2.2.6.2.2). Using Verbose JSON in the RetrieveEntitySet request, the URI is obtained from the metadata name/value pair of a JSON object representing an entity, as described in Entity Type (as a Verbose JSON object) (section 2.2.6.3.3). For information about obtaining the URI for the RetrieveEntitySet request in the preferred OData 3.0 JSON format, see [ODataJSON4.0] section 4.5.8.

AtomPub [RFC5023] describes retrieving an AtomPub entry resource, which maps to an **EntityType** instance in an EDM, in an HTTP response payload that MUST be represented as an Atom entry document, as specified in [RFC4287] section 4.1.2. This document extends that behavior to allow additional representations of an entry resource to be retrieved by a data service client. A client states the desired response payload format by using the Accept (section 2.2.5.1) request header. This document defines three representations of **EntityTypes**: entity type as an Atom Entry element (section 2.2.6.2.2), entity as a JSON object ([ODataJSON4.0] section 6), and entity type as a Verbose JSON object (section 2.2.6.3.3).

If the RetrieveEntity request was successful, the response payload MUST contain the requested representation of the **EntityType** instance identified in the request URI. The payload of such a

response MUST be formatted by using Atom, JSON, or Verbose JSON according to the rules defined in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2), Entity (as a JSON object) ([ODataJSON4.0] section 6), and Entity Type (as a Verbose JSON object) (section 2.2.6.3.3).

If the RetrieveEntity request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveEntity request is defined as follows:

```
retrieveEntity-Req       = retrieveEntity-ReqLine
                           retrieveEntity-ReqHeaders
                           CRLF

retrieveEntity-ReqLine   = "GET"
                           SP entityTypeInstanceUri retrieveEntity-
                              QueryOps
                           SP HTTP-Version
                           CRLF

retrieveEntity-ReqHeaders = [DataServiceVersion]
                            ; see section 2.2.5.3
                            [MaxDataServiceVersion]
                            ; see section 2.2.5.7
                            [If-None-Match]
                            ; see section 2.2.5.6
                            [Accept]
                            ; see section 2.2.5.1
                            *(HTTP-Header-Types)

entityTypeInstanceUri = ; see section 2.2.7.4.1

retrieveEntity-QueryOps   = ["?"(customQueryOption /sysQueryOption)
                             *("&" customQueryOption / sysQueryOption)]
```

The syntax of a response to a successful RetrieveEntity request is defined as follows:

```
retrieveEntity-Resp         =  Status-Line
                               ; see [RFC2616] section 6.1.1
                               retrieveEntity-RespHeaders
                               CRLF
                               retrieveEntity-RespBody

retrieveEntity-RespHeaders  =  DataServiceVersion
                               ; see section 2.2.5.3
                               [Content-Type]
                               ; see section 2.2.5.2
                               [ETag]
                               ; see section 2.2.5.4
                               *(HTTP-Header-Types)

; Responses including related entities were requesting using the
; $expand query string operator defined in section 2.2.3.6.1

retrieveEntity-RespBody     =  <Entity (possibly with Expanded Navigation Property)
                                formatted by using JSON as per [ODataJSON4.0]
                                sections 6 and 8.3>
                               / <Entity Type instance (possibly with
                                  related instances) formatted by using
                                  Atom as per sections
                                   2.2.6.2.2 and 2.2.6.2.6.1>
                               / (begin-object
                                  quotation-mark "d" quotation-mark
                                  name-seperator
                                  entityTypeInVJson
                                  end-object)
                                  ; see section 2.2.6.3.3 &
```

```
; 2.2.6.3.9.1
```

2.2.7.2.3 RetrieveComplexType Request

The purpose of the RetrieveComplexType request is to enable the value of a **ComplexType** property on an **EntityType** instance to be retrieved by a client. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an entry resource. As such, this request type is not based on an AtomPub-defined [RFC5023] request.

A RetrieveComplexType request MUST use the HTTP GET method and the URI specified in the HTTP request line MUST be a valid data service URI that identifies a **ComplexType** property on an **EntityType** instance, as specified in URI Format: Resource Addressing Rules (section 2.2.3).

If the RetrieveComplexType request was successful, the response MUST have a 200 status code as specified in [RFC2616]. The payload of such a response MUST be formatted using XML, JSON, or Verbose JSON, according to the rules defined in Complex Type (section 2.2.6.5.1), Complex Value ([ODataJSON4.0] section 7.2), and Complex Type (section 2.2.6.3.4), respectively.

If the RetrieveComplexType request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveComplexType request is defined as follows:

```
retrieveCT-Req          = retrieveCT-ReqLine
                          retrieveCT-ReqHeaders
                          CRLF

retrieveCT-ReqLine      = "GET"
                          SP entityTypeInstanceCTPropertyUri retrieveCT-QueryOps
                          SP HTTP-Version
                          CRLF

retrieveCT-ReqHeaders   = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-None-Match]             ; see section 2.2.5.6
                          [Accept]                    ; see section 2.2.5.1
                          *(HTTP-Header-Types)

entityTypeInstanceCTPropertyUri = <Any Resource Path identifying a ComplexType
                                   property on an Entity Type instance>
                                   ; see section 2.2.3 and section 2.2.3.5 -- URI3

retrieveCT-QueryOps      = ["?" (customQueryOption / formatQueryOp)
                           *("&" customQueryOption)]
                           ; see section 2.2.3.1 & section 2.2.3.6.1.5
```

The syntax of a response to a successful RetrieveComplexType request is defined as follows:

```
retrieveCT-Resp         =  Status-Line                ; see [RFC2616] section 6.1.1
                           retrieveCT-RespHeaders
                           CRLF
                           retrieveCT-RespBody

retrieveCT-RespHeaders  =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           *(HTTP-Header-Types)

retrieveCT-RespBody     =  <Complex value formatted by using JSON as per [ODataJSON4.0]
                            section 7.2>
                           / <ComplexType property value formatted by using XML as per
                            section 2.2.6.5.1>
```

```
/(begin-object
  quotation-mark "d" quotation-mark
  name-seperator
  (entityCTInVJson
  end-object)
  ; see section 2.2.6.3.4
```

2.2.7.2.4 RetrievePrimitiveProperty Request

The purpose of the RetrievePrimitiveProperty request is to enable the value of an **EDMSimpleType** property on an **EntityType** instance (or one of its constituent **ComplexType** instances) to be retrieved by a client. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an Entry Resource. As such, this request type is not based on any AtomPub-defined [RFC5023] request.

A RetrievePrimitiveProperty request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a property of type **EDMSimpleType** on an **EntityType** instance or **ComplexType** instance, as described in URI Format: Resource Addressing Rules (section 2.2.3).

If the RetrievePrimitiveProperty request was successful, the response MUST have a 2*xx* status code as specified in [RFC2616]. The payload of such a response MUST be formatted using XML, JSON, or Verbose JSON, as defined in EDMSimpleType Property (section 2.2.6.5.3), Primitive Value ([MS-ODATAJSON] section 2.1.17), and EDMSimpleType Property (section 2.2.6.3.8), respectively.

If the RetrievePrimitiveProperty request is not successful (for example, an error occurred while processing the request) the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrievePrimitiveProperty request is defined as follows:

```
retrievePP-Req          = retrievePP-ReqLine
                          retrievePP-ReqHeaders
                          CRLF

retrievePP-ReqLine      = "GET"
                          SP entityTypeInstancePropertyUri retrievePP-QueryOps
                          SP HTTP-Version
                          CRLF

retrievePP-ReqHeaders   = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-None-Match]             ; see section 2.2.5.6
                          [Accept]                    ; see section 2.2.5.1
                          *(HTTP-Header-Types)

entityTypeInstancePropertyUri = ; see section 2.2.7.4.3

retrievePP-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
                          ; see section 2.2.3.1
```

The syntax of a response to a successful RetrievePrimitiveProperty request is defined as follows:

```
retrievePP-Resp         =  Status-Line                ; see [RFC2616] section 6.1.1
                           retrievePP-RespHeaders
                           CRLF
                           retrievePP-RespBody

retrievePP-RespHeaders  =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           [Content-Type]             ; see section 2.2.5.2
                           *(HTTP-Header-Types)
```

```
retrievePP-RespBody     =  <Property value formatted in JSON  as per [ODataJSON4.0]
                            section 7>
                           / <Property value formatted in XML as per section
                            2.2.6.5.3>
                           / (begin-object
                              quotation-mark "d" quotation-mark
                              name-seperator
                              begin-object
                              (entityPropertyInVJson / entityPropertyInVJson2)
                              end-object
                              end-object)
                              ; see section 2.2.6.3.8
```

2.2.7.2.5 RetrieveValue Request

The purpose of the RetrieveValue Request is to enable the raw value of an **EDMSimpleType** property on an **EntityType** instance to be retrieved by a client. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an Entry Resource. As such, this request type is not based on any AtomPub-defined [RFC5023] request.

A RetrieveValue Request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies the raw value of a property (of type **EDMSimpleType**) on an **EntityType** instance, as specified in URI Format: Resource Addressing Rules (section 2.2.3).

If the RetrieveValue Request was successful, the response MUST have a 2*xx* status code, as specified in [RFC2616], and the response body MUST be formatted as specified in EDMSimpleType Property (section 2.2.6.4.1).

If the RetrieveValue Request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveValue Request is defined as follows:

```
retrieveValue-Req         = retrieveValue-ReqLine
                            retrieveValue-ReqHeaders
                            CRLF

retrieveValue-ReqLine     = "GET"
                            SP entityTypeInstancePropertyUri
                               "/$value"
                               retrieveValue-QueryOps
                            SP HTTP-Version
                            CRLF

retrieveValue-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            [If-None-Match]                 ; see section 2.2.5.6
                            *(HTTP-Header-Types)

entityTypeInstancePropertyUri = ; see section 2.2.7.4.3

retrieveValue-QueryOps    = ["?" customQueryOption *("&" customQueryOption)]
                            ; see section 2.2.3.6
```

The syntax of a response to a successful RetrieveValue Request is defined as follows:

```
retrieveValue-Resp       = Status-Line                ; see [RFC2616] section 6.1.1
                           retrieveValue-RespHeaders
                           CRLF
                           retrieveValue-RespBody
```

```
retrieveValue-RespHeaders= DataServiceVersion          ; see section 2.2.5.3
                           [ETag]                      ; see section 2.2.5.4
                           [Content-Type]              ; see section 2.2.5.2
                           *(HTTP-Header-Types)

retrieveValue-RespBody   = <Property value formatted as per section 2.2.6.4.1>
```

2.2.7.2.6 RetrieveCollectionProperty Request

The purpose of the RetrieveCollectionProperty request is to enable the value of a collection property on an **EntityType** instance to be retrieved by a client. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an entry resource. As such, this request type is not based on any AtomPub-defined [RFC5023] request.

A RetrieveCollectionProperty Request MUST use the HTTP GET method. Additionally, the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a property of type collection, as described in URI Format: Resource Addressing Rules (section 2.2.3).

If the RetrieveCollectionProperty request is successful, the response MUST have a 2*xx* status code, as specified in [RFC2616]. The payload of such a response MUST be formatted by using XML as defined in Collection of EDMSimpleType Values (section 2.2.6.5.3), JSON as defined in Collection of Primitive Values ([ODataJSON4.0] section 7.3), or Verbose JSON as defined in Collection of EDMSimpleType Values (section 2.2.6.3.7).

If the RetrieveCollectionProperty request is not successful, the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveCollectionProperty request is defined as follows:

```
retrieveCollection-Req          = retrieveCollection-ReqLine
                          retrieveCollection-ReqHeaders
                          CRLF

retrieveCollection-ReqLine      = "GET"
                          SP entityTypeInstanceCollectionPropertyUri
                          SP HTTP-Version
                          CRLF

retrieveCollection-ReqHeaders   = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-None-Match]             ; see section 2.2.5.6
                          [Accept]                    ; see section 2.2.5.1
                          *(HTTP-Header-Types)

entityTypeInstanceCollectionPropertyUri = <Any Resource Path identifying a Collection
                                  property on an Entity Type instance>
                                  ; see section 2.2.3 and section 2.2.3.5 – URI18
```

The syntax of a response to a successful RetrieveCollectionProperty request is defined as follows:

```
retrieveCollection-Resp         =  Status-Line                ; see [RFC2616] section 6.1.1
                           retrieveCollection-RespHeaders
                           CRLF
                           retrieveCollection-RespBody

retrieveCollection-RespHeaders  =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                      ; see section 2.2.5.4
                           [Content-Type]              ; see section 2.2.5.2
                           *(HTTP-Header-Types)

retrieveCollection-RespBody     =  <Collection of primitive values and collection of complex
```

```
                                values formatted in JSON as per [ODataJSON4.0]
                                sections 7.3 and 7.4, respectively>
                               / <Collection property values formatted in
                                  Verbose JSON as per sections 2.2.6.3.5 and 2.2.6.3.7>
                               / <Collection property values formatted in XML as
                                  per sections 2.2.6.5.6 and 2.2.6.5.7>
```

2.2.7.2.7 RetrieveServiceMetadata Request

The purpose of the RetrieveServiceMetadata Request is to enable a client to retrieve the conceptual schema definition language (CSDL) document, as specified in [MC-CSDL], describing the data model associated with the data service. AtomPub, as specified in [RFC5023], does not define the usage of an Entity Data Model, as specified in [MC-CSDL], with data services. As such, this request type is not based on any AtomPub-defined [RFC5023] request.

A RetrieveServiceMetadata Request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI, that identifies the $metadata endpoint of a data service, as specified in URI Syntax (section 2.2.3.1).

If the RetrieveServiceMetadata Request was successful, the response MUST have a 200 status code, as specified in [RFC2616], and the response body MUST be formatted as specified in Conceptual Schema Definition Language Document for Data Services (section 2.2.3.7.2). If the RetrieveServiceMetadata Request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1). The version number returned as the value of the DataServiceVersion response header MUST match the value of the **DataServiceVersion** attribute (section 2.2.3.7.2) in the returned EDMX [MC-EDMX] document.

The syntax of a RetrieveServiceMetadata Request is defined as follows:

```
retrieveServiceMD-Req         = retrieveServiceMD-ReqLine
                                retrieveServiceMD-ReqHeaders
                                CRLF

retrieveServiceMD-ReqLine     = "GET"
                                SP serviceRootNoHost
                                "/$metadata"
                                retrieveServiceMD-QueryOps
                                SP HTTP-Version
                                CRLF

retrieveServiceMD-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                                [MaxDataServiceVersion]     ; see section 2.2.5.7
                                *(HTTP-Header-Types)

serviceRootNoHost = <the ServiceRoot section of the data service URI
                    beginning after  the definition of the host>
                     ; see section 2.2.3.2

retrieveServiceMD-QueryOps    = ["?" customQueryOption *("&" customQueryOption)]
                                 ; see section 2.2.3.6
```

The syntax of a response to a successful RetrieveServiceMetadata Request is defined as follows:

```
retrieveServiceMD-Resp       = Status-Line           ; see [RFC2616] section 6.1.1
                               retrieveServiceMD-RespHeaders
                               CRLF
                               retrieveServiceMD-RespBody

retrieveServiceMD-RespHeaders= DataServiceVersion          ; see section 2.2.5.3
                               [Content-Type]              ; see section 2.2.5.2
                               *(HTTP-Header-Types)
```

```
retrieveServiceMD-RespBody   = <CSDL-based document describing the data model
                                defining the data service>
                               ; see section 2.2.3.7.2
```

2.2.7.2.8 RetrieveServiceDocument Request

The purpose of the RetrieveServiceDocument request is to enable a client to retrieve the Service Document describing the collection of resources exposed by a data service, as described in Service Document (section 2.2.3.7.1).

AtomPub, as specified in [RFC5023], describes the retrieval of an AtomPub Service Document in an HTTP response payload. This document extends that behavior to allow additional representations of the Service Document to be retrieved by a data service client. A client states the desired response payload format by using the Accept (section 2.2.5.1) request header. This document defines three such representations of Service Documents: Service Document (section 2.2.6.2.7) for AtomPub, Service Document ([MS-ODATAJSON] section 2.1.16) for JSON, and Service Document (section 2.2.6.3.12) for Verbose JSON.

A RetrieveServiceDocument Request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies the service root (section 2.2.3.2).

If the RetrieveServiceDocument request was successful, the response MUST have a 200 status code, as specified in [RFC2616], and the response body MUST be formatted as specified in Service Document (section 2.2.6.2.7) for AtomPub-based requests, Service Document ([MS-ODATAJSON] section 2.1.16) for JSON-based requests, and Service Document (section 2.2.6.3.12) for Verbose JSON-based requests. If the RetrieveServiceDocument request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveServiceDocument request is defined as follows:

```
retrieveServiceDocument-Req = retrieveServiceDocument-ReqLine
                              retrieveServiceDocument-ReqHeaders

retrieveServiceDocument-ReqLine = "GET"
                                  SP serviceRoot
                                  retrieveServiceDocument-QueryOps
                                  SP HTTP-Version
                                  CRLF

retrieveServiceDocument-ReqHeaders = [DataServiceVersion]    ; see section 2.2.5.3
                                     [MaxDataServiceVersion] ; see section 2.2.5.7
                                     [Accept]                ; see section 2.2.5.1
                                     *(HTTP-Header-Types)

serviceRoot                         = ; see section 2.2.3.2

retrieveServiceDocument-QueryOps    = ["?" customQueryOption
                                       *("&" customQueryOption)]
                                      ; see section 2.2.3.6
```

The syntax of a response to a successful RetrieveServiceDocument request is defined as follows:

```
retrieveServiceDocument-Resp = Status-Line        ; see [RFC2616] section 6.1.1
                               retrieveServiceDocument-RespHeaders
                               CRLF
                               retrieveServiceDocument-RespBody

retrieveServiceDocument-RespHeaders = DataServiceVersion ; see section 2.2.5.3
```

```
                                    [Content-Type]     ; see section 2.2.5.2
                                    *(HTTP-Header-Types)

retrieveServiceDocument-RespBody   = <Service Document formatted by using AtomPub
                                      as per section 2.2.6.2.7>
                                    /<Service Document formatted by using JSON as per
                                     [MS-ODATAJSON] (section 2.1.16)>
                                    /(begin-object
                                      quotation-mark "d" quotation-mark
                                      name-seperator
                                      VJsonServiceDocument
                                      end-object)
                                      ; see section 2.2.6.3.11
```

2.2.7.2.9 RetrieveLink Request

The purpose of the RetrieveLink request is to enable the links representing the relationships from one EntityType instance to another or from one **EntityType** instance to all others in a specified EntitySet to be retrieved by a client. AtomPub, as specified in [RFC5023], does not define a request of this type. Therefore, this request type is not based on an AtomPub-defined [RFC5023] request.

A RetrieveLink request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies the links from one **EntityType** instance to another or from one **EntityType** instance to all other entities in a specified **EntitySet**, as described in the URI7 grammar rule in Resource Path: Semantics (section 2.2.3.5).

If the RetrieveLink request was successful, the response MUST have a 2*xx* status code, as specified in [RFC2616]. The response payload MUST contain a representation of the set of links or single link identified by the request URI. A client states the desired response payload format using the Accept (section 2.2.5.1) request header. This document defines three such formats for links: XML, JSON, and Verbose JSON, as defined in Links (section 2.2.6.5.5), links formatted as a resource reference ([MS-ODATAJSON] section 2.1.23), and Links (section 2.2.6.3.10), respectively.

If the RetrieveLink request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveLink Request is defined as follows:

```
retrieveLink-Req          = retrieveLink-ReqLine
                            retrieveLink-ReqHeaders
                            CRLF

retrieveLink-ReqLine      = "GET"
                            SP entityTypeInstanceLinksUri retrieveLink-QueryOps
                            SP HTTP-Version
                            CRLF

retrieveLink-ReqHeaders   = [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            [Accept]                    ; see section 2.2.5.1
                            *(HTTP-Header-Types)

entityTypeInstanceLinksUri = <Any Resource Path identifying a single Link or collection
                              of Links>
                              ; see section 2.2.3 and section 2.2.3.5 -- URI7

retrieveLink-QueryOps     = ["?" (customQueryOption / formatQueryOp)
                             *("&" customQueryOption)]
                            ; see section 2.2.3.1
```

The syntax of a response to a successful RetrieveLink Request is defined as follows:

```
retrieveLink-Resp         =  Status-Line              ; see [RFC2616] section 6.1.1
                             retrieveLink-RespHeaders
                             CRLF
                             retrieveLink-RespBody

retrieveLink-RespHeaders  =  DataServiceVersion       ; see section 2.2.5.3
                             *(HTTP-Header-Types)

retrieveLink-RespBody     =  <Representation of the links or link addressed in the
                              request URI formatted as per [MS-ODATAJSON] section 2.1.23>
                             / <Representation of the links or link addressed in the
                                request URI formatted as per section 2.2.6.5.5>
                             / quotation-mark "d" quotation-mark
                               name-seperator
                               <Representation of the links or link addressed in the
                                request URI formatted as per section 2.2.6.3.10>
```

2.2.7.2.10 RetrieveCount Request

Applies to the OData 2.0 and OData 3.0 protocols

The purpose of the RetrieveCount request is to enable the count of a collection of **EntityType** instances or a single **EntityType** instance to be retrieved by the client. AtomPub, as specified in [RFC5023], does not define a request of this type. Therefore, this request type is not based on an AtomPub-defined [RFC5023] request.

A RetrieveCount Request MUST use the HTTP GET method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a collection of **EntityType** instances, as specified in URI Format: Resource Addressing Rules (section 2.2.3).

If the RetrieveValue Request was successful, the response MUST have a 2*xx* status code, as specified in [RFC2616], and the response body MUST be formatted as specified in EDMSimpleType Property (section 2.2.6.4.1).

If the RetrieveValue Request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The RetrieveCount Request is supported only in the OData 2.0 and OData 3.0 protocols.

The syntax of a RetrieveCount Request is defined as follows:

```
retrieveCount-Req         = retrieveCount-ReqLine
                            retrieveCount-ReqHeaders
                            CRLF

retrieveCount-ReqLine     = "GET"
                            SP entitySetUri [/ entityTypeInstanceUri]
                               "/$count"
                               retrieveCount-QueryOps
                            SP HTTP-Version
                            CRLF

retrieveCount-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            *(HTTP-Header-Types)

entitySetUri                 = ; see section 2.2.7.1.1

entityTypeInstanceUri = ; see section 2.2.7.4.1

retrieveCount-QueryOps    = ["?" customQueryOption
                                /sysQueryOption *("&"
                                 customQueryOption
                                / sysQueryOption)]
```

```
                                ; see section 2.2.3.1
```

The syntax of a response to a successful RetrieveCount Request is defined as follows:

```
retrieveCount-Resp       = Status-Line                ; see [RFC2616] section 6.1.1
                           retrieveCount-RespHeaders
                           CRLF
                           retrieveCount-RespBody

retrieveCount-RespHeaders= DataServiceVersion         ; see section 2.2.5.3
                           [Content-Type]             ; see section 2.2.5.2
                           *(HTTP-Header-Types)

retrieveCount-RespBody   = <Count value formatted as per section 2.2.6.4.1>
```

2.2.7.2.11 Retrieve Request Containing a Customizable Feed Mapping

Applies to the OData 2.0 and OData 3.0 protocols

In OData 2.0 and OData 3.0, it is possible to map the value of a property on an **EntityType** to another location in the feed. A retrieve request made to an **EntityType** or to a collection of **EntityType** instances with a property mapping has the same format as a request made to an **EntityType** or collection of **EntityType** instances without a property mapping. The response to a retrieve request made to an **EntityType** or to a collection of **EntityType** instances with a property mapping MUST be formatted according to the rules defined in Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping (section 2.2.6.2.2.1).

The format of the URI in a Retrieve request does not change when the request is being made to an **EntityType** instance or collection of **EntityType** instances with a customizable feed property mapping. System query options and service operations that accept a property name as a parameter MUST treat the value of the parameter as identifying the name of a property identified on an **EntityType** and not the mapped location of that property.

2.2.7.2.12 RetrieveMediaResource Request

A RetrieveMediaResource Request is used by a client to retrieve an AtomPub Media Resource, as specified in [RFC5023] section 9.6, that maps to a BLOB associated with an EntityType instance, as described in Abstract Data Model (section 2.2.1). A request of this type is defined by AtomPub [RFC5023] section 9.6. This section adds constraints to those defined in AtomPub for this request type.

If the RetrieveMediaResource Request was successful, the response payload MUST contain the requested representation of the media resource identified in the request URI. If the RetrieveMediaResource Request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a RetrieveMediaResource Request is defined as follows:

```
retrieveMR-Req           = retrieveMR-ReqLine
                           retrieveEntity-ReqHeaders
                           CRLF

retrieveMR-ReqLine       = "GET"
                           SP entityTypeInstanceMRUri
                              retrieveEntity-QueryOps
                           SP HTTP-Version
                           CRLF

retrieveEntity-ReqHeaders = ; section 2.2.7.2.2
```

```
retrieveEntity-QueryOps   = ; section 2.2.7.2.2

entityTypeInstanceMRUri   = ; see section 2.2.7.3.5
```

The syntax of a response to a successful RetrieveMediaResource Request is defined as follows:

```
retrieveMR-Resp             =  Status-Line
                               ; see [RFC2616] section 6.1.1
                               retrieveEntity-RespHeaders
                               CRLF
                               retrieveMR-RespBody

retrieveEntity-RespHeaders  =  ; section 2.2.7.2.2

retrieveMR-RespBody         =  <Any valid HTTP response body>
```

2.2.7.3 Update Request Types

2.2.7.3.1 UpdateEntity Request

An UpdateEntity Request is used by a client to update an existing AtomPub Entry Resource, as specified in [RFC5023], that maps to an EntityType instance in the abstract data model used in this document, as described in Abstract Data Model (section 2.2.1). The base rules and semantics of this request type are defined by AtomPub, as specified in [RFC5023] section 5.4. This section adds constraints to those defined in AtomPub for this request type.

As in [RFC5023] section 5.4.2, requests of this type use the HTTP PUT method, and the URI specified by the client in the HTTP request line represents an AtomPub entry resource. Given that an entry resource maps to an **EntityType** instance in an Entity Data Model (EDM), the HTTP request line URI MUST be equal to any valid data service URI that identifies an **EntityType** instance, as described in URI Format: Resource Addressing Rules (section 2.2.3). Additionally, this specification allows UpdateEntity requests to use the MERGE method (as defined in PATCH/MERGE (section 2.2.4.1)) or PATCH method (as defined in [RFC5789]) instead of PUT to specify that a merge-based update is to be performed.

[RFC5023] section 5.4.2 states that the request body can be an AtomPub entry resource (that maps to an **EntityType** instance in an EDM) that is represented as an Atom entry document. This document extends this rule to allow additional representations of an entry resource to be used. This document defines three such representations: entity type as an Atom Entry element (section 2.2.6.2.2), entity as a JSON object ([ODataJSON4.0] section 6), and entity type as a Verbose JSON object (section 2.2.6.3.3). A client SHOULD specify the representation used in the request body by including a Content-Type header in the request.

When the request body is a representation of an Entry Resource (**EntityType** instance in EDM terms), the client can specify if the resource/entity (in addition to being updated) is to be automatically linked to other already existing entities in the data service that are related through associations such that the opposite association end has a cardinality of 1 or 0-or-1. Such linking MUST only be supported if the **EntityType** of the to-be-edited entity defines a NavigationProperty that identifies a single **EntityType** instance. For example, an existing employee entity might need to be rebounded to an existing manager entity in a human resources management focused data service.

To rebind the entity to an existing entity, the client MUST include the required binding information in the representation of the associated **NavigationProperty** in the request payload.

To bind the entity to an existing entity using the Atom format, the **NavigationProperty** MUST be represented as specified in Navigation Property (section 2.2.6.2.4) with one exception: the **href** attribute of the **atom:link** element is equal the URI that identifies the existing entity that is the target of the link.

To bind the new entity to an existing entity by using the preferred OData 3.0 JSON format, see [ODataJSON4.0] section 8.5.

To bind the new entity to an existing entity by using the Verbose JSON format, the **NavigationProperty** MUST represent the entity that is the target of the link by using the Inline Representation (section 2.2.6.3.9.1) of a **NavigationProperty** with the single inlined entity represented using only the "__metadata" name/value pair (the properties of the inlined entity SHOULD NOT be provided). If the inlined entity's properties are provided, they MUST be ignored by the data service.

If the UpdateEntity request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have 0 bytes in the response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that MUST be formatted the same as the response body to a RetrieveEntity request (section 2.2.7.2.2).

If the UpdateEntity request is not successful (for example, if an error occurs during the request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateEntity request is defined as follows:

```
updateEntity-Req         = updateEntity-ReqLine
                           updateEntity-ReqHeaders
                           CRLF
                           updateEntity-ReqBody

updateEntity-ReqLine     = ("PUT" / "MERGE"/ "PATCH")
                           SP entityTypeInstanceUri updateEntity-QueryOps
                           SP HTTP-Version CRLF

updateEntity-ReqHeaders  = [DataServiceVersion]     ; see section 2.2.5.3
                           [MaxDataServiceVersion]  ; see section 2.2.5.7
                           [If-Match]               ; see section 2.2.5.5
                           [Content-Type]           ; see section 2.2.5.2
                           [Prefer]                 ; see section 2.2.5.9
                           *(HTTP-Header-Types)

entityTypeInstanceUri = ; see section 2.2.7.4.1

updateEntity-QueryOps    = ["?" (customQueryOption / filterQueryOp) *("&" customQueryOption)]
                           ; see section 2.2.3.1

updateEntity-ReqBody     = <Entity in JSON format as per [ODataJSON4.0] section 6>
                           / <entityTypeInVJson       ; see section 2.2.6.3.3>
                           / <Entity Type in Atom format as per section 2.2.6.2.2>
                           / <Entity Type with Property Mapping in Atom format as per section
                               2.2.6.2.2.1>
```

The syntax of a response to a successful UpdateEntity request is defined as follows:

```
updateEntity-Resp        = Status-Line           ; see [RFC2616] section 6.1.1
                           updateEntity-RespHeaders
                           CRLF
                           [updateEntity-RespBody]

updateEntity-RespHeaders  = DataServiceVersion   ; see section 2.2.5.3
                            [ETag]               ; see section 2.2.5.4
                            [Preference-Applied]  ; see section 2.2.5.10
                            *(HTTP-Header-Types)
updateEntity-RespBody     =  <Entity (possibly with expanded navigation properties) formatted
                              by using JSON as per [ODataJSON4.0] sections 6 and 8.3>
                              / <Entity Type instance (possibly with
```

```
      related instances) formatted by using
      Atom as per sections 2.2.6.2.2 and 2.2.6.2.6.1>
   / (begin-object
      quotation-mark "d" quotation-mark
      name-seperator
      entityTypeInVJson
      end-object)
      ; see section 2.2.6.3.3 &
      ; 2.2.6.3.9.1
```

2.2.7.3.1.1 Example

See Appendix A: Sample Entity Data Model and CSDL Document (section 6) for the sample model and data used in this section.

The example below updates an existing Order entity and rebinds it to Customer with EntityKey value "ALFKI".

HTTP Request:

```
PUT /service.svc/Orders(1) HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
Accept: application/json;odata=verbose
Content-Length: nnn

{
   "__metadata":{ "uri": "Orders(1)" },
   "OrderID": 1,
   "ShippedDate": "\/Date(872467200000)\/",
   "Category" : { __metadata: {uri:"/Customers('ALFKI')" } }
}
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Length: nnn
DataServiceVersion: 1.0
ETag: W/"X'000000000000FA01'"
```

2.2.7.3.2 UpdateComplexType Request

The purpose of the UpdateComplexType request is to enable the value of a **ComplexType** instance to be updated. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an entry resource. As such, this request type is not based on an AtomPub-defined [RFC5023] request.

An UpdateComplexType request MUST use the HTTP PUT, PATCH, or MERGE method, as specified in PATCH/MERGE (section 2.2.4.1). Additionally, the URI specified by the client in the HTTP request line MUST be a valid data service URI, that identifies a **ComplexType** instance, as specified in URI Format: Resource Addressing Rule (section 2.2.3) .

The payload of the request MAY be formatted using the XML, JSON, or Verbose JSON format, according to the rules defined in Complex Type (section 2.2.6.5.1), Complex Value ([ODataJSON4.0] section 7.2), and Complex Type (section 2.2.6.3.4), respectively. A server accepting a request of this type using the PUT method MUST replace the value of the **ComplexType** addressed via the request URI with the value provided in the payload. A server accepting a request of this type using the MERGE or PATCH method MUST merge the property values of the **ComplexType** addressed via the request URI with the property values that are provided in the payload, as specified in PATCH/MERGE (section 2.2.4.1).

If the UpdateComplexType request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have 0 bytes in the response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that MUST be formatted the same as a response body to an RetrieveComplexType request (section 2.2.7.2.3).

If the UpdateComplexType request is not successful (for example, if an error occurs during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateComplexType request is defined as follows:

```
updateCT-Req            = updateCT-ReqLine
                          updateCT-ReqHeaders
                          CRLF
                          updateCT-ReqBody

updateCT-ReqLine        = "PUT" / "MERGE" / "PATCH"
                          SP entityTypeInstanceCTUri updateCT-QueryOps
                          SP HTTP-Version
                          CRLF

updateCT-ReqHeaders     = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-Match]                  ; see section 2.2.5.5
                          [Content-Type]              ; see section 2.2.5.2
                          [Prefer]                    ; see section 2.2.5.9
                          *(HTTP-Header-Types)

updateCT-ReqBody        = <Complex value formatted in JSON as per [ODataJSON4.0]
                           section 7.2>
                          / <ComplexType value formatted in XML as
                             per section 2.2.6.5.1>
                          / (begin-object entityCTInVJson end-object)
                             ; see section 2.2.6.3.4

entityTypeInstanceCTUri = <Any Resource Path identifying a Complex Type instance>
                                ; see section 2.2.3 and section 2.2.3.5 -- URI3

updateCT-QueryOps       = ["?" customQueryOption *("&" customQueryOption)]
```

The syntax of a response to a successful UpdateComplexType request is defined as follows:

```
updateCT-Resp           =  Status-Line                ; see [RFC2616] section 6.1.1
                           updateCT-RespHeaders
                           CRLF
                           [updateCT-RespBody]

updateCT-RespHeaders    =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           [Preference-Applied]           ; see section 2.2.5.10
                           *(HTTP-Header-Types)

updateCT-RespBody        =  <Complex value formatted by using JSON as per [ODataJSON4.0]
                            section 7.2>
                            /<ComplexType property value formatted by using XML as per
                              section 2.2.6.5.1>
                            /(begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             (entityCTInVJson / entityCTInVJson2)
                             end-object)
                             ; see section 2.2.6.3.4
```

2.2.7.3.3 UpdatePrimitiveProperty Request

The purpose of the UpdatePrimitiveProperty request is to enable the value of an EDMSimpleType property on an EntityType instance to be updated. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an Entry Resource. As such, this request type is not based on an AtomPub-defined [RFC5023] request.

An UpdatePrimitiveProperty request MUST use either the HTTP PUT, PATCH, or MERGE (as specified in PATCH/MERGE (section 2.2.4.1)) and the URI specified by the client in the HTTP request line MUST be a valid data service URI, as specified in URI Format: Resource Addressing Rules (section 2.2.3), which identifies a property (of type **EDMSimpleType**) on an **EntityType** instance (or on one of the ComplexType properties of that **EntityType** instance).

The payload of the request MAY be formatted by using XML, JSON, or Verbose JSON, according to the rules defined in EDMSimpleType Property (section 2.2.6.5.3), Primitive Value ([MS-ODATAJSON] section 2.1.17), and EDMSimpleType Property (section 2.2.6.3.8), respectively. A server receiving a request of this type MUST replace the value of the property addressed via the request URI with the property value provided in the payload. The server MUST perform the same behavior whether the request uses the MERGE, PATCH, or PUT method.

If the property addressed is one of the properties which define the **EntityKey** for the addressed entity, then the request MUST be considered invalid.

If the UpdatePrimitiveProperty request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have 0 bytes in the response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that includes the updated entity that MUST be formatted the same as a response body to a RetrievePrimitiveProperty request (section 2.2.7.2.4).

If the UpdatePrimitiveProperty request is not successful (for example, if an error occurs during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdatePrimitiveProperty request is defined as follows:

```
updatePP-Req            = updatePP-ReqLine
                          updatePP-ReqHeaders
                          CRLF
                          updatePP-ReqBody

updatePP-ReqLine        = "PUT" / "MERGE" / "PATCH"
                          SP entityTypeInstancePropertyUri updatePP-QueryOps
                          SP HTTP-Version
                          CRLF

updatePP-ReqHeaders     = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-Match]                  ; see section 2.2.5.5
                          [Content-Type]              ; see section 2.2.5.2
                        [Prefer]                    ; see section 2.2.5.9
                          *(HTTP-Header-Types)

updatePP-ReqBody        = <Primitive value formatted in JSON as per {MS-ODATAJSON]
                           section 2.1.17>
                          / <Property value formatted in XML as per section 2.2.6.5.3>
                          / (begin-object entityPropertyInVJson end-object)
                             ; see section 2.2.6.3.8

entityTypeInstancePropertyUri = ; see section 2.2.7.4.3

updatePP-QueryOps       = ["?" customQueryOption *("&" customQueryOption)]
```

The syntax of a response to a successful UpdatePrimitiveProperty request is defined as follows:

```
updatePP-Resp           =  Status-Line                 ; see [RFC2616] section 6.1.1
                           updatePP-RespHeaders
                           CRLF
                           [updatePP-RespBody]

updatePP-RespHeaders    =  DataServiceVersion          ; see section 2.2.5.3
                           [ETag]                      ; see section 2.2.5.4
                           [Preference-Applied]            ; see section 2.2.5.10
                           *(HTTP-Header-Types)
updatePP-RespBody     =  <Primitive value formatted in JSON as per {MS-ODATAJSON]
                          section 2.1.17>
                          / <Property value formatted in XML as per section
                             2.2.6.5.3>
                          / (begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             begin-object
                             (entityPropertyInVJson / entityPropertyInVJson2)
                             end-object
                             end-object)
                             ; see section 2.2.6.3.8
```

2.2.7.3.4 UpdateCollectionProperty Request

The purpose of the UpdateCollectionProperty request is to enable the value of a collection instance to be updated. AtomPub, as specified in [RFC5023], does not define operations on subcomponents of an Entry Resource. As such, this request type is not based on an AtomPub-defined [RFC5023] request.

UpdateCollectionProperty request MUST use the HTTP PUT method. Additionally, the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a collection property instance, as specified in URI Format: Resource Addressing Rule (section 2.2.3). HTTP PATCH and MERGE methods are not valid with the UpdateCollectionProperty request.

The payload of the request MAY be formatted by using XML, JSON, or Verbose JSON format, according to the rules defined in Collection of Complex Type Instances (section 2.2.6.5.2) and Collection of EDMSimpleType Values (section 2.2.6.5.4) for XML, Collection of Primitive Values (section 7.3) and Collection of Complex Values (section 7.4) in [ODataJSON4.0] for JSON, or Collection of Complex Type Instances (section 2.2.6.3.5) and Collection of EDMSimpleType Values (section 2.2.6.3.7) for Verbose JSON. A server accepting a request of this type by using the PUT method MUST replace the entire value of the collection type addressed via the request URI with the value provided in the payload.

If the UpdateCollectionProperty request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], if the response contains no response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that MUST be formatted the same as the response body to a RetrieveCollectionProperty request (section 2.2.7.2.6).

If the UpdateCollectionProperty request is not successful (for example, if an error occurs during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateCollectionProperty request is defined as follows:

```
updateCollection-Req            = updateCollection-ReqLine
                          updateCollection-ReqHeaders
                          CRLF
                          updateCollection-ReqBody

updateCollection-ReqLine        = "PUT"
```

```
                          SP entityTypeInstanceCollectionUri
                          SP HTTP-Version
                          CRLF

updateCollection-ReqHeaders     = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-Match]                  ; see section 2.2.5.5
                          [Content-Type]              ; see section 2.2.5.2
                          [Prefer]                    ; see section 2.2.5.9
                          *(HTTP-Header-Types)

updateCollection-ReqBody        = <Collection of primitive values or collection of
                                   complex values formatted in JSON
                                   as per [ODataJSON4.0] sections 7.3 and 7.4>
                                  / <Collection of values formatted in Verbose JSON as
                                     per sections 2.2.6.3.5 or 2.2.6.3.7>
                                  / <Collection of values formatted in XML as
                                     per sections 2.2.6.5.6 and 2.2.6.5.7>

entityTypeInstanceCollectionUri = <Any Resource Path identifying a Collection instance>
                                ; see section 2.2.3 and section 2.2.3.5 – URI8
```

The syntax of a response to a successful UpdateComplexType Request is defined as follows:

```
updateCollection-Resp           =  Status-Line     ; see [RFC2616] section 6.1.1
                                      updateCollection-RespHeaders
                                      CRLF
                                      [updateCollection-RespBody]

updateCollection-RespHeaders    =  DataServiceVersion          ; see section 2.2.5.3
                                      [Preference-Applied]           ; see section 2.2.5.10
                                      *(HTTP-Header-Types)

updateCollection-RespBody     =   <Collection of primitive values or collection of
                                    complex values formatted in JSON
                                    as per [ODataJSON4.0] sections 7.3 and 7.4>
                                   / <Collection of values formatted in Verbose JSON
                                      as per sections 2.2.6.3.5 or 2.2.6.3.7>
                                   / <Collection of values formatted in XML
                                      as per sections 2.2.6.5.6 and 2.2.6.5.7>
```

2.2.7.3.5 UpdateValue Request

The purpose of the UpdateValue request is to enable the value of an **EDMSimpleType** property on an **EntityType** instance to be updated. AtomPub [RFC5023] does not define operations on subcomponents of an Entry Resource. As such, this request type is not based on an AtomPub-defined [RFC5023] request.

An UpdateValue request MUST use the HTTP PUT, PATCH, or MERGE method (as specified in PATCH/MERGE (section 2.2.4.1)). Additionally, the URI specified by the client in the HTTP request line MUST be a valid data service URI, as specified in URI Format: Resource Addressing Rules (section 2.2.3), that identifies the raw value of a property (of type **EDMSimpleType**) on an **EntityType** instance (or on one of the **ComplexType** properties of that **EntityType** instance).

The payload of the request MUST be formatted according to the rules defined in EDMSimpleType Property (section 2.2.6.4.1). A server receiving a request of this type MUST replace the value of the property addressed via the request URI with the value provided in the payload. The server MUST perform the same behavior whether the request uses the HTTP MERGE, PATCH, or PUT method.

If the UpdateValue request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have a 0 byte response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY

have a 200 response code, as specified in [RFC2616], and a response body that includes the updated entity that MUST be formatted the same as a response body to a RetrieveValue request (section 2.2.7.2.5).

If the UpdateValue request is not successful (for example, if an error occurs during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateValue request is defined as follows:

```
updateValue-Req         = updateValue-ReqLine
                          updateValue-ReqHeaders
                          CRLF
                          updateValue-ReqBody

updateValue-ReqLine     = "PUT" | "MERGE"| "PATCH"
                          SP entityTypeInstancePropertyUri
                             "/$value"
                             updateValue-QueryOps
                          SP HTTP-Version
                          CRLF

updateValue-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-Match]                  ; see section 2.2.5.5
                          [Content-Type]              ; see section 2.2.5.2
                          [Prefer]                    ; see section 2.2.5.9
                          *(HTTP-Header-Types)

updateValue-ReqBody     = <Property value formatted as per section 2.2.6.4.1>

entityTypeInstancePropertyUri = ; see section 2.2.7.4.3

updateValue-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
```

The syntax of a response to a successful UpdateValue request is defined as follows:

```
updateValue-Resp         = Status-Line                ; see [RFC2616] section 6.1.1
                           updateValue-RespHeaders
                           CRLF
                           [updateValue-RespBody]

updateValue-RespHeaders =  DataServiceVersion          ; see section 2.2.5.3
                           [ETag]                      ; see section 2.2.5.4
                           [Preference-Applied]           ; see section 2.2.5.10
                           *(HTTP-Header-Types)

updateValue-RespBody   = <Property value formatted as per section 2.2.6.4.1>
```

2.2.7.3.6 UpdateLink Request

The purpose of the UpdateLink request is to enable a Link to be established between two EntityType instances. AtomPub [RFC5023] does not define a request of this type. Therefore, this request type is not based on an AtomPub-defined [RFC5023] request.

An UpdateLink request MUST use the HTTP PUT, PATCH, or MERGE method (as specified in PATCH/MERGE (section 2.2.4.1)). Addtionally, the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a single link between two **EntityType** instances, as described in URI Format: Resource Addressing Rules (section 2.2.3).

For example, when using the model described in Appendix A: Sample Entity Data Model and CSDL Document (section 6), `http://host/service.svc/Orders(1)/$links/Customer` is a valid URI for

updating which Customer is associated with Orders(1). However, `http://host/service.svc/Customers('ALFKI')/$links/Orders(1)` is not valid for an UpdateLink request because the navigation property on Customer identifies a collection of Orders.

In this context, the **EntityType** instance identified by the resource path segment immediately prior to the "$links" segment in the request URI is referred to as the source entity. Requests of this type MUST contain a request body (formatted as a resource reference according to [MS-ODATAJSON] section 2.1.23, formatted according to the "linkVJson" rule in Links (section 2.2.6.3.10), or formatted according to the XML schema for a single link in Links (section 2.2.6.5.5)) that contains a URI that identifies the **EntityType** instance to be linked to from the source entity.

If the UpdateEntity request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have 0 bytes in the response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that MUST be formatted the same as a response body to an RetrieveLink request (section 2.2.7.2.9).

If the UpdateLink request is not successful (for example, if an error occurs during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateLink request is defined as follows:

```
updateLink-Req          =   updateLink-ReqLine
                            updateLink-ReqHeaders
                            CRLF
                            updateLink-ReqBody

updateLink-ReqLine      =   "PUT" / "MERGE"/ "PATCH
                            SP entityTypeInstanceSingleLinkUri updateLink-QueryOps
                            SP HTTP-Version
                            CRLF
                            updateLink-ReqBody

updateLink-ReqHeaders   =   [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            [Content-Type]              ; see section 2.2.5.2
                            [Prefer]                    ; see section 2.2.5.9
                            *(HTTP-Header-Types)

updateLink-ReqBody      =   <Resource reference formatted in JSON as per
                             [MS-ODATAJSON] section 2.1.23>
                            / linkVJson             ; see section 2.2.6.3.10
                            / <XML representation of a single link as per the XML
                               Schema in section 2.2.6.5.3>

entityTypeInstanceSingleLinkUri =  ; section 2.2.7.4.2

updateLink-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
                            ; see section 2.2.3.1
```

The syntax of a response to a successful UpdateLink request is defined as follows:

```
updateLink-Resp         =    Status-Line              ; see [RFC2616] section 6.1.1
                             updateLink-RespHeaders
                             CRLF
                             [updateLink-RespBody]

updateLink-RespHeaders  =    DataServiceVersion       ; see section 2.2.5.3
                             [Preference-Applied]         ; see section 2.2.5.10
                             *(HTTP-Header-Types)

updateLink-RespBody     =    <Representation of the links or link addressed in the
```

```
                                request URI formatted as per [MS-ODATAJSON] section 2.1.23>
                               / <Representation of the links or link addressed in the
                                  Request URI formatted as per section 2.2.6.5.5 >
                               / quotation-mark "d" quotation-mark
                                 name-seperator
                                 <Representation of the links or link addressed in the
                                  request URI formatted as per section 2.2.6.3.10>
```

2.2.7.3.7 UpdateMediaResource Request

An UpdateMediaResource request is used by a client to update an existing AtomPub media resource, as specified in [RFC5023] section 9.6. The AtomPub media resource maps to BLOB, which is described by an EntityType instance in the abstract data model used in this document, as described in Abstract Data Model (section 2.2.1). The base rules and semantics of this request type are defined by AtomPub, as specified in [RFC5023] section 9.6. This section adds constraints to those defined in AtomPub for this request type.

As in [RFC5023] section 9.6, requests of this type use the HTTP PUT method, and the URI specified by the client in the HTTP request line has to represent an AtomPub media resource. Given that media resources are described by Media Links Entries, which map to **EntityType** instances in an Entity Data Model (EDM), the HTTP request line URI MUST be equal to any valid URI that identifies a media resource associated with an existing **EntityType** instance. The MERGE and PATCH methods defined in this document are not supported for requests of this type.

If the UpdateMediaResource request is successful, the response in OData 1.0 and OData 2.0 MUST have a 204 response code, as specified in [RFC2616], and have 0 bytes in the response body, unless a Prefer header (section 2.2.5.9) has been specified to request content. When an OData 1.0 or OData 2.0 request contains a Prefer header value that requests content or when OData 3.0 is used, the response MAY have a 200 response code, as specified in [RFC2616], and a response body that MUST be formatted the same as the response body to an InsertMediaResource request (section 2.2.7.1.3).

If the UpdateMediaResource request is not successful (for example, an error occurred during request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an UpdateMediaResource request is defined as follows:

```
updateMR-Req             = updateMREntity-ReqLine
                           updateEntity-ReqHeaders
                           CRLF
                           updateEntity-ReqBody

updateMREntity-ReqLine   = "PUT"
                           SP entityTypeInstanceMRUri updateEntity-QueryOps
                           SP HTTP-Version CRLF

updateEntity-ReqHeaders  = [Prefer]                    ; see section 2.2.5.9
                           ; see section 2.2.7.3.1

entityTypeInstanceMRUri = <Any Resource Path identifying the Media Resource described by an
                           Entity Type instance represnting a Media Link Entry>
                           ; see section 2.2.3 and section 2.2.3.5 – URI17

updateEntity-QueryOps    = ; see section 2.2.7.3.1

updateMR-ReqBody         = <Any valid HTTP request body> ; see [RFC5023] section 9.6
```

The syntax of a response to a successful UpdateMediaResource request is defined as follows:

```
updateMR-Resp            = Status-Line           ; see [RFC2616] section 6.1.1
                           updateMR-RespHeaders
                           CRLF
```

```
                          [updateMR-RespBody]

updateMR-RespHeaders    = DataServiceVersion   ; see section 2.2.5.3
                          [ETag]                ; see section 2.2.5.4
                          [Preference-Applied]      ; see section 2.2.5.10
                          *(HTTP-Header-Types)

updateMR-RespBody       = <Media Entity in JSON format as per [ODataJSON4.0] section 10>
                          / <Entity Type representing a Media Link Entry in Atom format as
                             per section 2.2.6.2.2>
                          / (begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             entityTypeInVJson
                             end-object)
                            ; see section 2.2.6.3.3
                            ; the entityTypeInVJson representation MUST include all name
                            ; value/pairs denoted in the mleMetadata rule
```

2.2.7.3.8 Update Request Containing a Customizable Feed Property Mapping

Applies to the OData 2.0 and OData 3.0 protocols

In OData 2.0 and OData 3.0, it is possible to map the value of a property on an **EntityType** to another location in the feed. If the **EntityType** instance contained in an UpdateEntity, UpdateComplexType, or UpdatePrimitiveType Request has a property mapping defined on it and the Content-Type is AtomPub, then the request body MUST be formatted by using the **EntityType** formatting rules defined in Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping (section 2.2.6.2.2.1).

When sending an Update Request to an **EntityType** instance that has a customizable feed property mapping, the client MAY include the property value in an **m:properties** element as described in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2). If the property value is included in the **m:properties** element (as described in section 2.2.6.2.2), the data service MUST use that value when updating the **EntityType** instance.

2.2.7.4 Delete Request Types

2.2.7.4.1 DeleteEntity Request

The purpose of the DeleteEntity request is to enable an **EntityType** instance to be deleted from a data service. The base rules and semantics of this request type are defined by AtomPub, as specified in [RFC5023] section 9.4 -- Deleting Resources with DELETE. As described in Abstract Data Model (section 2.2.1), Entity Data Model constructs are mapped directly to data model concepts used in AtomPub. For example, **EntityTypes** are AtomPub Entry Resources and **EntitySets** are AtomPub collections. This section adds constraints to those defined in AtomPub for this request type.

As in [RFC5023] section 9.4, DeleteEntity requests MUST use the **HTTP DELETE** method, and the URI specified by the client in the HTTP request line addresses an AtomPub Member Resource. Because an Entry Resource (subtype of Member Resource) maps to an **EntityType** instance in an Entity Data Model, the HTTP request line URI MUST be any valid data service URI which identifies a single **EntityType** instance, as specified in URI Format: Resource Addressing Rules (section 2.2.3).

A DeleteEntity request MAY<66> cause additional side effects (for example, cascading deletes) to the entities in a data service.

If a DeleteEntity request is sent to a URI that identifies an entity that represents a Media Link Entry, then as part of deleting the entity, the associated Media Resource SHOULD also be deleted.

A DeleteEntity request MUST have an empty (0 bytes) payload. If the DeleteEntity request was successful, the response MUST have a 204 (No Content) status code, as specified in [RFC2616]. If the

DeleteEntity request is not successful, the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a DeleteEntity request is defined as follows:

```
deleteEntity-Req         = deleteEntity-ReqLine
                           deleteEntity-ReqHeaders
                           CRLF

deleteEntity-ReqLine     = "DELETE"
                            SP entityTypeInstanceUri deleteEntity-QueryOps
                            SP HTTP-Version CRLF

deleteEntity-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                           [MaxDataServiceVersion]     ; see section 2.2.5.7
                           [If-Match]                  ; see section 2.2.5.5
                           *(HTTP-Header-Types)

entityTypeInstanceUri    = <Any Resource Path identifying an Entity Type instance>
                           ; see section 2.2.3 and section 2.2.3.5 -- URI2 & URI6

deleteEntity-QueryOps    = ["?" customQueryOption *("&" customQueryOption)]
                           ; see section 2.2.3.1
```

The syntax of a response to a successful DeleteEntity request is defined as follows:

```
deleteEntity-Resp        = Status-Line    ; see [RFC2616] section 6.1.1
                           deleteEntity-RespHeaders
                           CRLF

deleteEntity-RespHeaders  = DataServiceVersion             ; see section 2.2.5.3
                            *(HTTP-Header-Types)
```

2.2.7.4.2 DeleteLink Request

The purpose of the DeleteLink request is to enable an existing link between two EntityType instances to be removed. AtomPub [RFC5023] does not define a request of this type. Therefore, this **request** type is not based on an AtomPub-defined [RFC5023] request.

A DeleteLink request MUST use the HTTP DELETE method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies a single link between two **EntityType** instances, as specified in URI Format: Resource Addressing Rules (section 2.2.3). For example, using Appendix A: Sample Entity Data Model and CSDL Document (section 6), the following are valid URIs for requests of this type:
`http://host/service.svc/Customers('ALFKI')/$links/Orders(1)` and
`http://host/service.svc/Orders(1)/$links/Customer`.

DeleteLink request MUST contain 0 bytes in the payload. If a DeleteLink request is successful, the response MUST have a 204 status code, as specified in [RFC2616], and an empty response body.

If the DeleteLink request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a DeleteLink request is defined as follows:

```
deleteLink-Req          =   deleteLink-ReqLine
                            deleteLink-ReqHeaders
                            CRLF

deleteLink-ReqLine      =   "DELETE"
                            SP entityTypeInstanceSingleLinkUri deleteLink-QueryOps
```

```
                              SP HTTP-Version
                              CRLF

deleteLink-ReqHeaders   =     [DataServiceVersion]
                              ; see section 2.2.5.3
                              [MaxDataServiceVersion]
                              ; see section 2.2.5.7
                              *(HTTP-Header-Types)

entityTypeInstanceSingleLinkUri = <Any Resource Path identifying a single link>
                                    ; see section 2.2.3 and section 2.2.3.5 -- URI7

deleteLink-QueryOps     = ["?" customQueryOption *("&" customQueryOption)]
                              ; see section 2.2.3.1
```

The syntax of a response to a successful DeleteLink request is defined as follows:

```
deleteLink-Resp         =     Status-Line              ; see [RFC2616] section 6.1.1
                              deleteLink-RespHeaders
                              CRLF

deleteLink-RespHeaders  =     DataServiceVersion       ; see section 2.2.5.3
                              *(HTTP-Header-Types)
```

2.2.7.4.3 DeleteValue Request

The purpose of the DeleteValue Request is to enable an **EDMSimpleType** property of an **EntityType** instance to be set to null. AtomPub [RFC5023] does not define operations on subcomponents of an Entry Resource. As such this request type is not based on an AtomPub-defined [RFC5023] request.

DeleteValue request MUST use the HTTP DELETE method and the URI specified by the client in the HTTP request line MUST be a valid data service URI, as defined in URI Format: Resource Addressing Rules (section 2.2.3), that identifies the raw value of a nullable property (of type **EDMSimpleType**) on an **EntityType** instance or on one of the **ComplexType** properties of that **EntityType** instance.

A DeleteValue request sent to a URI identifying a property for which null is not a valid value SHOULD be considered to be a malformed request and the server is to respond with the HTTP 4*xx* range of status codes.

A DeleteValue request MUST contain 0 bytes in the payload. If the DeleteValue request was successful, the response MUST have a 204 (No Content) status code, as specified in [RFC2616]. If the DeleteValue request is not successful, the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of a DeleteValue request is defined as follows:

```
deleteValue-Req         = deleteValue-ReqLine
                          deleteValue-ReqHeaders
                          CRLF

deleteValue-ReqLine     = "DELETE"
                          SP entityTypeInstancePropertyUri
                             "/$value"
                             deleteValue-QueryOps
                          SP HTTP-Version
                          CRLF

deleteValue-ReqHeaders  = [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          [If-Match]                  ; see section 2.2.5.5
                          *(HTTP-Header-Types)
```

```
entityTypeInstancePropertyUri = <Any Resource Path identifying a EDMSimpleType
                                 property on an Entity Type instance>
                                ; see section section 2.2.3 and section 2.2.3.5 -- URI4 &
URI5

deleteValue-QueryOps      = ["?" customQueryOption *("&" customQueryOption)]
```

The syntax of a response to a successful DeleteValue request is defined as follows:

```
deleteValue-Resp         = Status-Line                ; see [RFC2616] section 6.1.1
                           deleteValue-RespHeaders
                           CRLF

deleteValue-RespHeaders = DataServiceVersion          ; see section 2.2.5.3
                           *(HTTP-Header-Types)
```

2.2.7.5 Invoke Request Types

The purpose of the Invoke request is to enable a client to call a **FunctionImport**, as specified in [MC-CSDL] section 2.1.15, that is exposed by a service operation in a data service. Note that this excludes function imports that are used to specify either actions (section 2.2.1.3) or functions (section 2.2.1.4). AtomPub [RFC5023] does not define operations of this type. As such, this request type is not based on any AtomPub-defined [RFC5023] request.

Invoke Request MUST use the HTTP method specified by the data service's metadata document, as described in Conceptual Schema Definition Language Document for Data Services (section 2.2.3.7.2), and the URI specified by the client in the HTTP request line MUST be a URI which identifies a service operation, as described in Resource Path: Semantics (section 2.2.3.5).

If the **FunctionImport**, as specified in [MC-CSDL], that is exposed by a service operation requires input parameters, those parameters MUST be provided, as specified in Service Operation Parameters (section 2.2.3.6.3).

If the Invoke request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an Invoke request is defined as follows:

```
invoke-Req    =     invoke-ReqLine
                    invoke-ReqHeaders
                    CRLF

invoke-ReqLine      =     <Any standard or custom HTTP method>
                          ; The method supported for a particular
                          ; service operation
                          ; is defined by the httpMethod CSDL annotation
                          ; defined in section 2.2.3.7.2
                          SP serviceOperationUri invoke-QueryOps
                          SP HTTP-Version
                          CRLF

invoke-ReqHeaders   =     [DataServiceVersion]        ; see section 2.2.5.3
                          [MaxDataServiceVersion]     ; see section 2.2.5.7
                          *(HTTP-Header-Types)

serviceOperationUri =     <Any Resource Path identifying a Service Operation>
                           ; see section 2.2.3 and section 2.2.3.5

invoke-QueryOps      = ["?" (sysQueryOption / customQueryOption / serviceOpParam)
                         *("&" (customQueryOption / serviceOpParam) )]
                         ; see section 2.2.3.1 & section 2.2.3.6.3
```

The syntax of a response to a successful Invoke request is defined as follows:

```
invoke-Resp         =      Status-Line                 ; see [RFC2616] section 6.1.1
                           invoke-RespHeaders
                           CRLF
                           invoke-RespBody

invoke-RespHeaders  =      DataServiceVersion          ; see section 2.2.5.3
                           [ETag]                      ; see section 2.2.5.4
                           *(HTTP-Header-Types)

invoke-RespBody     =      invoke-RespBody-collEt
                           / invoke-RespBody-et
                           / invoke-RespBody-collCt
                           / invoke-RespBody-ct
                           / invoke-RespBody-collPrim

; a Function Import which returns a collection of entities
invoke-RespBody-collEt   = retrieveEntitySet-RespBody ; see section 2.2.7.2.1


; a Function Import which returns an entity
invoke-RespBody-et       = retrieveEntity-RespBody    ; see section 2.2.7.2.2

; a Function Import which returns a collection of complex types
invoke-RespBody-collCt   = <Collection of complex values formatted by using JSON
                            as per [ODataJSON4.0] section 7.4>
                           /<Collection of complex type instances formatted by using XML
                             as per section 2.2.6.5.2>
                           /(begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             entityCollCTInVJson
                             end-object)
                             ; see section 2.2.6.3.5

; a Function Import which returns a single Complex Type
invoke-RespBody-ct       = retrieveCT-RespBody        ; see section 2.2.7.2.3

; a Function Import which returns a collection of primitives
invoke-RespBody-collPrim = <Collection of primitive values formatted by using JSON
                            as per [ODataJSON4.0] section 7.3>
                           /<Collection of primitive type values formatted by using XML
                            as per section 2.2.6.5.4>
                           /(begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             entityCollPrimValueInVJson)
                             end-object
                             ; see section 2.2.6.3.7

quotation-mark          = ; see [RFC4627] section 2.5

; a Function Import which returns a single primitive type value
invoke-RespBody-prim     = retrievePP-RespBody        ; see section 2.2.7.2.4
```

2.2.7.5.1 Invoke Action Request

Applies to the OData 3.0 protocol

The purpose of the Invoke Action request is to enable a client to call a **FunctionImport**, as specified in [MC-CSDL] section 2.1.15, that is exposed as an action (section 2.2.1.3) in a data service.

An Invoke Action request MUST use an HTTP POST. Additionally, the URI specified by the client in the HTTP request line MUST be a URI that identifies an action, as described in Resource Path: Semantics (section 2.2.3.5).

If the **FunctionImport**, as specified in [MC-CSDL], that is exposed as an action requires input parameters other than the binding parameter, those parameters MUST be provided in the body of the request by using the application/json or application/json;odata=verbose Content-Type, as specified in Action Parameters (section 2.2.3.6.5).

If the Invoke Action request is not successful (for example, an error occurred while processing the request), the response MUST be formatted according to Error Response (section 2.2.8.1).

In particular, if the Invoke Action request includes an If-Match (section 2.2.5.5) header that contains an ETag for the resource against which the request is bound, servers MUST check that version of the ETag against the latest server known value. If the server discovers the client's version is out of date, the server MUST fail the request with a 412 Precondition Failed response (section 3.2.8).

The OData protocol provides a way to pass action parameters in JSON and Verbose JSON formats only.

The syntax of an Invoke Action request is defined as follows:

```
invoke-Req                = invokeAction-ReqLine
                            invokeAction-ReqHeaders
                            CRLF
                            invokeAction-Parameters

invokeAction-ReqLine      = "POST"
                            SP actionUri
                            SP HTTP-Version
                            CRLF

invokeAction-ReqHeaders   = [DataServiceVersion]        ; see section 2.2.5.3
                            [MaxDataServiceVersion]     ; see section 2.2.5.7
                            [If-Match]                  ; see section 2.2.5.5
                            ["Content-Type" WSP "=" WSP "application/json;odata=verbose"
CRLF]
                            *(HTTP-Header-Types)

actionUri                 = <Any Resource Path identifying an Action>
                            ; see section 2.2.3 and section 2.2.3.5

invokeAction-Parameters   = <Action parameters in JSON as per [ODataJSON4.0] section 17>
                            / begin-object
                            [ actionParameterNVP
                            *(value-seperator actionParameterNVP) ]
                            end-object

actionParameterNVP        = quotation-mark actionParameterName quotation-mark
                            name-seperator
                            actionParameterValue

actionParameterName       = *pchar
                            ; the name of a parameter to the Action
                            ; as defined by the corresponding FunctionImport.

actionParameterValue      = "null" | typeInVJson

typeInVJson                = <Any type (or collection) serialized in Verbose JSON>
                            ; formatted as per the rules described in section 2.2.6.3
```

The syntax of a response to a successful Invoke Action request is defined as follows:

```
invokeAction-Resp         = Status-Line        ; see [RFC2616] section 6.1.1
                            invoke-RespHeaders ; section 2.2.7.5
                            CRLF
                            [invoke-RespBody]  ; missing if the Action has no ReturnType
```

2.2.7.5.2 Invoke Function Request

Applies to the OData 3.0 protocol

The purpose of the Invoke Function request is to enable a client to call a **FunctionImport**, as specified in [MC-CSDL] section 2.1.15, that is exposed as a function (section 2.2.1.4) in a data service.

An Invoke Function request MUST use the HTTP GET method. Additionally, the URI specified by the client in the HTTP request line MUST be a URI that identifies one or more functions, as described in Resource Path: Semantics (section 2.2.3.5).

If the **FunctionImport** element(s), as specified in [MC-CSDL], that is exposed by one or more functions requires input parameters other than the binding parameter, those parameters MUST be provided, as specified in Function Parameters (section 2.2.3.6.4).

If the Invoke Function request is not successful (for example, an error occurred during the request processing), the response MUST be formatted according to Error Response (section 2.2.8.1).

The syntax of an Invoke Function request is defined as follows:

```
invokeFunction-Req    = invoke-ReqLine
                        invoke-ReqHeaders
                        CRLF

invoke-ReqLine        = "GET"
                        SP functionUri invoke-QueryOps
                        SP HTTP-Version
                        CRLF

invoke-ReqHeaders     = [DataServiceVersion]        ; see section 2.2.5.3
                        [MaxDataServiceVersion]     ; see section 2.2.5.7
                        *(HTTP-Header-Types)

functionUri           = <Any Resource Path identifying one or more Functions>
                        ; see section 2.2.3 and section 2.2.3.5

invoke-QueryOps       = ["?" (sysQueryOption / customQueryOption / serviceOpParam)
                        *("&" (customQueryOption / serviceOpParam) )]
                        ; see section 2.2.3.1 & section 2.2.3.6.3
```

The syntax of a response to a successful Invoke request is defined as follows:

```
invokeFunction-Resp = invoke-Resp
                      ; see section 2.2.7.5
```

2.2.7.6 Batch Request

The request types defined in the preceding subsections provide mechanisms for a client to query and manipulate resources exposed by a data service whereby each request type maps to a single HTTP request/response exchange. Such request types integrate deeply with HTTP, as specified in [RFC2616], allowing a client to leverage the services (for example, caching) provided by the HTTP infrastructure deployed at large.

While the request types noted above address many common data service scenarios, as described in Protocol Examples (section 4), use cases exist where it is beneficial to enable a client of a data service to "batch" up a group of requests and send that Batch to the data service in a single HTTP request. This section defines a Batch request type that reduces the number of roundtrips to a data service for applications that need to make numerous requests and a change set syntax as a way to logically group a set of requests in a single unit within a batch.

A data service MAY<67> support requests of this type. If a data service does not implement support for a Batch request, it has to return a 4*xx* response code in the response to any Batch request sent to it.

Batch request MUST use the HTTP POST method and the URI specified by the client in the HTTP request line MUST be a valid data service URI that identifies the batch endpoint of a data service, see URI9 in Resource Path: Semantics (section 2.2.3.5).

Query operations and change sets MAY<68> be intermixed within a Batch request and occur in any order. A query operation MUST consist of a single Retrieve request, as defined in Retrieve Request Types (section 2.2.7.2), or of an Invoke Request (section 2.2.7.5) that uses the HTTP GET method. Change sets MUST consist of one or more of the following request types:

- Insert Request Types (section 2.2.7.1)
- Update Request Types (section 2.2.7.3)
- Delete Request Types (section 2.2.7.4)
- Invoke Request (section 2.2.7.5)), by using the HTTP PUT, POST, PATCH, MERGE, or DELETE methods.

Request types within a batch MUST be performed by the server with the same semantics used when the request type is used outside of the context of a batch.

The order of change sets and query operations within a Batch request is significant, as it states the order in which the data service MUST process the components of a Batch. The order of requests within a change set MUST NOT be significant such that a data service can process the requests in a change set in any order. All operations in a change set represent a single change unit so the server MUST successfully process and apply all the requests in the change set or else apply none of the requests in the change set. It is up to the data service to define rollback semantics to undo any requests within a change set that have been applied before another request in that same change set failed and thereby honor this all-or-nothing requirement.

A Batch request is represented by using a single HTTP POST request with a payload of the multipart/mixed media type, as specified in [RFC2046]. The request MUST use Multipurpose Internet Mail Extensions (MIME) version 1.0 and include a Content-Type header that conforms to the **contentTypeMime** rule in the grammar shown in the following Batch request headers listing. Each change set and/or query operation in a Batch request is represented as a distinct MIME part, separated by the boundary defined in the Content-Type header of the request. Implementers SHOULD follow the recommendations for boundary generation described in [RFC2046] section 5.1.

Preambles and Epilogues in the MIME payload, as defined in [RFC2046], are valid but MUST be ignored by the server.

Note Unlike the other grammars described in Request Types (section 2.2.7), all ABNF grammars defined in this document that add constraints to rules defined in MIME-related RFCs, [RFC2045], and [RFC2046], do not follow the ABNF rules defined in [RFC2616] section 2.1.

```
contentTypeMime       =  "Content-Type:multipart/mixed;"
                         ; see [RFC2045] section 5 & [RFC2046] section 5.1
                         *SP
                         "boundary="
                         boundary        ; see [RFC2046] section 5.1.1
```

Listing: Batch Request Headers

2.2.7.6.1 Change Set Syntax

Each change set MUST be represented within its MIME part in the Batch payload as a nested multipart/mixed message. This MUST be specified by including a Content-Type header, which conforms to the **contentTypeMime** rule in the Batch Request Headers listing in Batch Request (section 2.2.7.6). This Content-Type header MUST appear in the header section of the MIME part in the Batch Request (section 2.2.7.6) associated with the change set, MUST state that the media type is multipart/mixed, and MUST define the boundary for the nested multipart/mixed content, as specified in [RFC2046], of the change set. This boundary, which separates requests within a single change set, is different from that used by the batch to separate change sets from one another and from Query Operations.

Each MIME part that represents a request within a change set in the multipart/mixed message that defines a single change MUST include a Content-Type MIME part header, a Content-Transfer-Encoding MIME part header, and a Content-ID MIME part header. These MIME part headers are described in the grammar that is shown in the following Required Batch Request MIME Part Headers listing.

```
contentTypeMime-Part             =  "Content-Type:application/http;" [*SP "version=1.1"]
                                    ; see [RFC2045] section 5 & [RFC2616] section 19.1
contentTransferEncodingMime-Part =  "Content-Transfer-Encoding:binary"
                                    ; see [RFC2045] section 6
contentIdMime-Part               =  "Content-ID:" msgId
                                    ; see [RFC2045] section 7
```

Listing: Required Batch Request MIME Part Headers

As described by the "application/http" media type in the preceding grammar, the body of a MIME part that represents a request within a change set MUST be formatted as an HTTP request just as if it was a standalone HTTP request (not part of a Batch request) and follow the rules for Insert, Update, Delete, or Invoke request types, as defined in Insert Request Types (section 2.2.7.1), Update Request Types (section 2.2.7.3), Delete Request Types (section 2.2.7.4), and Invoke Request (section 2.2.7.5). The only exception is that the Content-Length request header is not mandatory in requests that include request payloads. For restrictions on the HTTP constructs that can be used in batched HTTP requests, see HTTP Request Restrictions (section 2.2.7.6.3).

2.2.7.6.1.1 Referencing Requests in a Change Set

To enable referencing a new entity created via an Insert request, as described in Insert Request Types (section 2.2.7.1), from a subsequent request within the same change set, this document defines a request referencing mechanism.

Each MIME part that represents a request in a change set MUST include a Content-ID MIME header as an HTTP header of the MIME part request. If a MIME part defines an InsertEntity request or invokes an action that returns an entity, then the entity, as defined by the InsertEntity request or action invocation, MAY be referenced by subsequent requests in the change set by using the "$<Content-ID value of previous request>" token as the root of the resource path that is used in a subsequent request within the change set. When used in this way, the token acts as an alias for the resource path that identifies the new entity. Requests in different change sets cannot reference one another, even if they are in the same batch.

2.2.7.6.2 Query Operation Syntax

Each query operation in a Batch Request is represented as a MIME part, separated from other MIME parts using the boundary defined in the Content-Type header of the Batch request.

A MIME part representing a query operation in a Batch request MUST include Content-Type and Content-Transfer-Encoding MIME headers, as described in the grammar in the Required Batch Request MIME Part Headers listing in Change Set Syntax (section 2.2.7.6.1).

As described by the "application/http" media type, specified in [RFC2616], in the grammar referenced, the body of a MIME part representing a query operation MUST be formatted as an HTTP request just as if it was a standalone HTTP request (not part of a Batch request) and follow the rules defined in Retrieve Request Types (section 2.2.7.2). For restrictions on the HTTP constructs that can be used in batched HTTP requests, see HTTP Request Restrictions (section 2.2.7.6.3).

2.2.7.6.3 HTTP Request Restrictions

Each MIME part body which represents a single request SHOULD NOT:

- Include authentication or authorization related HTTP headers because it is unlikely the infrastructure used for authentication will parse and utilize such headers.
- Include Expect, From, Max-Forwards, Range, or TE headers because their contents will be ignored.

Data services can choose to disallow additional HTTP constructs in HTTP requests serialized within MIME part bodies. For example, a data service might choose to disallow chunked encoding to be used by such HTTP requests.

2.2.7.6.4 Batch Request Syntax

The syntax of a Batch request is defined as follows.

```
batch-Req               =   batch-ReqLine
                            batch-ReqHeaders
                            CRLF
                            batch-ReqBody
batch-ReqLine            =   "POST"
                             SP batchUri
                             SP HTTP-Version
                             CRLF

batch-ReqHeaders         =   [DataServiceVersion]        ; see section 2.2.5.3
                             [MaxDataServiceVersion]     ; see section 2.2.5.7
                             contentTypeMime CRLF        ; see section 2.2.7.6
                             *(HTTP-Header-Types)

batchUri                 =   ; see URI9 in section 2.2.3.5

batch-ReqBody            =   [preamble CRLF]
                             dash-boundary transport-padding CRLF
                             body-part *encapsulation
                             close-delimiter transport-padding
                             [CRLF epilogue]

body-part                =   batchQueryOperation-ReqBodyPart
                             / batchChangeSet-ReqMultiPart
; this rule redefines and adds constraints to the 'body-part' grammar rule in [RFC2046]

batchQueryOperation-ReqBodyPart = mimePart-ReqHeaders
                                  <Any Retrieve request as described in section
                                   2.2.7.2>

mimePart-ReqHeaders       =  contentTypeMime-Part   ; see section 2.2.7.6.1
                             CRLF
                             contentTypeEncodingMime-Part ; see section 2.2.7.6.1
                             CRLF

batchChangeSet-ReqMultiPart = contentTypeMime CRLF  ; see section 2.2.7.6
                              batchChangeSet-ReqBody

batchChangeSet-ReqBody    =  [preamble CRLF]
                             dash-boundary transport-padding CRLF
                             batchChangeSet-ReqBodyPart
                             *batchChangeSet-ReqEncapsulation
```

```
                              close-delimiter transport-padding
                              [CRLF epilogue]

batchChangeSet-ReqBodyPart=   mimePart-ReqHeaders
                              contentIdMime-Part    ; see section 2.2.7.6.1
                              [Prefer]    ; see section 2.2.5.9
                              <Any request type valid within a Change Set as defined
                               in section 2.2.7.6>

batchChangeSet-ReqEncapsulation  =   delimiter transport-padding
                                     CRLF batchChangeSet-ReqBodyPart

preamble                = ; see [RFC2046] section 5.1.1
dash-boundary           = ; see [RFC2046] section 5.1.1
transport-padding       = ; see [RFC2046] section 5.1.1
encapsulation           = ; see [RFC2046] section 5.1.1
close-delimiter         = ; see [RFC2046] section 5.1.1
transport-padding       = ; see [RFC2046] section 5.1.1
epilogue                = ; see [RFC2046] section 5.1.1
id                      = ; see [RFC2045] section 7
```

2.2.7.6.5 Example Batch Request

This section shows an example Batch request that contains the following operations in the order described:

- A query operation
- A change set that contains the following requests:
  - InsertEntity Request (with Content-ID = 1)
  - Insert Entity (references request with Content-ID = 1)
  - UpdateEntity Request
- A second query operation

Legend:

- Request bodies are excluded in favor of English descriptions inside "<>" brackets to simplify the example

```
POST /service.svc/$batch HTTP/1.1
Host: host
Content-Type: multipart/mixed; boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding:binary

GET /service.svc/Customers('ALFKI')
Host: host


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd621
Content-Length: ###


--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding: binary
```

```
Content-ID: 1

POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Customer>

--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

POST $1/Orders HTTP/1.1
Host: host
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of a new Order>

--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-ID: 3

PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
If-Match: xxxxx
Content-Length: ###

<Verbose JSON representation of Customer ALFKI>

--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)--


--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding:binary

GET service.svc/AnEntitySetWhichDoesNotExist HTTP/1.1
Host: host


--batch(36522ad7-fc75-4b56-8c71-56071383e77b)--
```

2.2.7.6.6 Batch Responses

If a data service receives a Batch request (section 2.2.7.6) with a valid set of HTTP request headers, it MUST respond with a 202 Accepted HTTP response code to indicate that the request has been accepted for processing, but that the processing has not yet been completed. The requests within the Batch request body might be malformed or subsequently fail. This mechanism enables clients of a Batch implementation to stream the results of a Batch request without having to first wait for all requests to be processed.

Alternatively, if a data service receives a Batch request with an invalid set of headers, it MUST return a 4*xx* response code. For example, a Batch request cannot use post tunneling, as described in Tunneled Requests (section 2.2.7.7), and therefore, the presence of an X-HTTP-Method header in a Batch request results in a response containing a 4*xx* response code.

All responses to Batch request MUST use the multipart/mixed media type by including the Content-Type header as defined by the **contentTypeMime** rule in the ABNF grammar shown in Batch Request (section 2.2.7.6).

The server is free to re-order responses within a change set. In order for the client to correlate responses with requests, all responses within a change set MUST include the Content-ID MIME part header with the value of the Content-ID specified for the corresponding request. To support early client implementations that specify Content-ID as part of the individual request rather than as a MIME part header for that request and where use within the individual request does not conflict with some other use by the server, servers SHOULD recognize the Content-ID in either location and write it in the response in the same location in which it is read in the request.

Structurally, a Batch Response body MUST match one-to-one with the corresponding Batch Request (section 2.2.7.6) body, such that the same multipart MIME message structure defined for requests is used for responses. The difference is that each MIME part represents a response instead of request. The exception to this rule is that when a request within a change set fails, the change set response is not represented using the multipart/mixed media type. Instead, a single response, using the "application/http" media type, is returned that applies to all requests in the change set and MUST be formatted according to Error Response (section 2.2.8.1).

2.2.7.6.7 Batch Response Syntax

The syntax of a Batch response is defined as follows.

```
batch-Resp              =   Status-Line
                            batch-RespHeaders
                            CRLF
                            batch-RespBody

batch-RespHeaders       =   DataServiceVersion        ; see section 2.2.5.3
                            contentTypeMime CRLF        ; see section 2.2.7.6
                            *(HTTP-Header-Types)

batch-RespBody          =   [preamble CRLF]
                             dash-boundary transport-padding CRLF7
                             [body-part] *encapsulation
                             close-delimiter transport-padding
                             [CRLF epilogue]

body-part                =   batchQueryOperation-RespBodyPart
                            / batchChangeSet-RespMultiPart
                            / batchChangeSetErr-RespBodyPart
                              ;used only for Change Set error responses
; this rule (body-part) redefines and adds additional constraints to the 'body-part' grammar
rule in [RFC2046]

batchQueryOperation-RespBodyPart  =   mimePartResponseHeaders
                                      <A response to a Retrieve request as described
                                       in section 2.2.7.2>


mimePart-RespHeaders     =   contentTypeMime-Part  ; see section 2.2.7.6.1
                             CRLF
                             contentTypeEncodingMime-Part ; see section 2.2.7.6.1
                             CRLF
                             [DataServiceId]             ; see section 2.2.5.11

batchChangeSet-RespMultiPart =   contentTypeMime CRLF  ; see section 2.2.7.6
                                 batchChangeSet-RespBody

batchChangeSet-RespBody  =   [preamble CRLF]
                             dash-boundary transport-padding CRLF
                             batchChangeSet-RespBodyPart
                             *batchChangeSet-RespEncapsulation
                             close-delimiter transport-padding
                             [CRLF epilogue]

batchChangeSet-RespBodyPart       =   mimePart-RespHeaders
                                      contentIdMime-Part       ; see section 2.2.7.6.1
```

```
                                    [Preference-Applied]         ; see section 2.2.5.10
                                    <A (success) response to any request type
                                     valid within a Change Set as defined in
                                     section 2.2.7.6>

batchChangeSetErr-RespBodyPart    =   mimePartResp-Headers
                                      <A Change Set error response as defined in
                                       section 2.2.7.6.6>

batchChangeSet-RespEncapsulation  =   delimiter transport-padding
                                      CRLF batchChangeSet-RespBodyPart

preamble                = ; see [RFC2046] section 5.1.1
dash-boundary           = ; see [RFC2046] section 5.1.1
transport-padding       = ; see [RFC2046] section 5.1.1
encapsulation           = ; see [RFC2046] section 5.1.1
close-delimiter         = ; see [RFC2046] section 5.1.1
transport-padding       = ; see [RFC2046] section 5.1.1
epilogue                = ; see [RFC2046] section 5.1.1
Status-Line             = ; see [RFC2616] section 6.1
```

2.2.7.6.8 Example Batch Response

The examples below represent a few of the possible responses to the Batch Request shown in Example Batch Request (section 2.2.7.6.5).

Example 1: The following example response assumes all requests except the Retrieve Request in the final query operation succeeded. The Retrieve operation is assumed to have failed because the requested resource did not exist.

The formatting used is the same as that defined in Example Batch Request (section 2.2.7.6.5). Response bodies are excluded in favor of English descriptions inside "<>" brackets to simplify the example.

```
HTTP/1.1 202 Accepted
DataServiceVersion: 1.0
Content-Length: ####
Content-Type: multipart/mixed; boundary=batch(36522ad7-fc75-4b56-8c71-56071383e77b)


--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>

--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: multipart/mixed; boundary=changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Length: ###

--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Customer('POIUY')
Content-Length: ###

<AtomPub representation of a new Customer entity>
```

```
--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

HTTP/1.1 201 Created
Content-Type: application/atom+xml;type=entry
Location: http://host/service.svc/Orders(200)
Content-Length: ###

<AtomPub representation of a new Order entity>

--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 3

HTTP/1.1 204 No Content
Host: host


--changeset(77162fcd-b8da-41ac-a9f8-9357efbbd621)--


--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>

--batch(36522ad7-fc75-4b56-8c71-56071383e77b)--
```

Example 2: The example response below assumes that at least one of the requests in the change set failed due to a failed data service specific authorization check, and that the Retrieve Request in the final query operation failed because the requested resource did not exist. All other requests are assumed to have succeeded.

The formatting used is the same as that defined in Example Batch Request (section 2.2.7.6.5). Response bodies are excluded in favor of English descriptions inside "<>" brackets to simplify the example.

```
HTTP/1.1 202 Accepted
DataServiceVersion: 1.0
Content-Length: ####
Content-Type: multipart/mixed
              ; boundary=batch(36522ad7-fc75-4b56-8c71-56071383e77b)

--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 200 Ok
Content-Type: application/atom+xml;type=entry
Content-Length: ###

<AtomPub representation of the Customer entity with EntityKey ALFKI>
--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 403 Forbidden
```

```
Content-Type: application/xml
Content-Length: ###

<Error message>
--batch(36522ad7-fc75-4b56-8c71-56071383e77b)
Content-Type: application/http
Content-Transfer-Encoding: binary

HTTP/1.1 404 Not Found
Content-Type: application/xml
Content-Length: ###

<Error message>
--batch(36522ad7-fc75-4b56-8c71-56071383e77b)—
```

2.2.7.7 Tunneled Requests

This section does not define a new request type, but rather defines an alternate way of representing Update Request Types (section 2.2.7.3) and Delete Request Types (section 2.2.7.4) by using a POST request and the X-HTTP-Method (section 2.2.5.8) request header. This alternative representation is often referred to as "POST tunneling".

This Tunneled Request type is fully defined through X-HTTP-Method header usage. As such, see X-HTTP-Method (section 2.2.5.8) for details regarding how the X-HTTP-Method header is used to create a Tunneled Request.

2.2.8 Response Types

2.2.8.1 Error Response

This section defines the structure of a response payload if a request to a data service completed in error. Two types of error conditions are defined: "in-stream" and "top-level".

An in-stream error occurs when a data service that is processing a request detects an error after the status line and zero or more response headers and potentially part of the response body have been sent to the client. For an in-stream error, the HTTP response code, headers, and payload cannot be used to indicate that an error has occurred. This document does not prescribe an error response generation method for in-stream errors.

In the context of this document, "top-level" errors SHOULD be used when the data service that is processing a request detects that an error has occurred before the status line or any response headers have been sent to the client. This document defines three formats for top-level error messages: XML, JSON, and Verbose JSON, as specified in [RFC4627].

XML Error Response (section 2.2.8.1.1) defines the rules for a top-level error in a response body that is formatted by using XML. Servers MUST use this format when sending a top-level error response to a request that includes an Accept (section 2.2.5.1) request header with the values "application/xml" or "application/atom+xml", or to a request that does not include an Accept (section 2.2.5.1) request header. When formatting error responses by using XML, servers SHOULD include a Content-Type response header with the value "application/xml".

For details about formatting an error by using the preferred OData3.0 JSON format, see [MS-ODATAJSON] section 2.1.28.

Verbose JSON Error Response (section 2.2.8.1.2) defines the rules for a top-level error in a response body that is formatted by using Verbose JSON, as specified in [RFC4627]. Servers MUST use this format when a sending top-level error response to a request that includes an Accept (section 2.2.5.1) request header with the value "application/json;odata=verbose". When formatting error responses by

using Verbose JSON, servers SHOULD include a Content-Type response header with the value "application/json;odata=verbose".

The syntax of a top-level error response is defined as follows:

```
stdError-Resp         =   Status-Line              ; see [RFC2616] section 6.1.1
                          stdError-RespHeaders
                          CRLF
                          stdError-RespBody

stdError-RespHeaders  =   DataServiceVersion       ; see section 2.2.5.3
                          [Content-Type]           ; see section 2.2.5.2
                          *(HTTP-Header-Types)

stdError-RespBody     =   <Error response in JSON as per [MS-ODATAJSON]
                           section 2.1.28>
                          / <XML representation of a top-level error as defined in
                           section 2.2.8.1.1>
                          / stdErrorVJson-RespBody  ; see section 2.2.8.1.2
```

Listing: Top-level Error Response ABNF Grammar

2.2.8.1.1 XML Error Response

This section defines an XML format for error messages that MUST be used in response payloads representing top-level errors, as specified in Error Response (section 2.2.8.1), when the server uses the default XML format. For examples of errors that are defined in JSON, see Error Response in [MS-ODATAJSON] section 2.1.28. For examples of errors that are defined in Verbose JSON, see Verbose JSON Error Response (section 2.2.8.1.2).

The structure of a top-level error message formatted as XML (application/xml) is defined in the following XSD Schema for Top-level Error Payloads Formatted by Using XML listing.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://schemas.microsoft.com/ado/2007/08/
   dataservices/metadata">
<xs:element name="error">
 <xs:complexType>
  <xs:sequence>
    <xs:element name="code" type="xs:string" minOccurs="1"
    maxOccurs="1"/>
    <xs:element name="message" minOccurs="1" maxOccurs="1">
        <xs:complexType>
         <xs:simpleContent>
          <xs:extension base="xs:string">
           <xs:attribute ref="xml:lang" type="xs:string"/>
          </xs:extension>
         </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="innererror" type="xs:any"
    minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Listing: XSD Schema for Top-level Error Payloads Formatted by Using XML

code: A data-service-defined string that serves as a substatus to the HTTP response code.

message: A human-readable string that describes the error.

innererror: An optional element that contains data-service–specific debugging information to assist a service implementer in determining the cause of an error.

Note The **innererror** element is to be used only in development environments. If it is present in a response from a production data service, it is not to include system internal information in order to guard against information disclosure security concerns.

2.2.8.1.2 Verbose JSON Error Response

This section defines the structure of an error message represented in Verbose JSON, as specified in [RFC4627], that MUST be used in response payloads representing top-level errors, as specified in Error Response (section 2.2.8.1). For examples of errors that are represented in the preferred OData 3.0 JSON format, see [MS-ODATAJSON] section 2.1.28. For examples of errors that are defined in XML format, see XML Error Response (section 2.2.8.1.1).

The syntax of a top-level error using Verbose JSON (application/json;odata=verbose) is shown in the following ABNF Grammar for Top-level Error Payloads Formatted Using Verbose JSON listing.

```
stdErrorVJson-RespBody   =  begin-object
                            quotation-mark "error" quotation-mark
                            name-seperator
                            begin-object
                            codeNVP
                            value-seperator messageNVP
                            [value-seperator innererrorNVP]
                            end-object
                            end-object

codeNVP       =  quotation-mark "code" quotation-mark name-seperator string
   ; A data service defined string which serves as a
   ; sub status to the HTTP response code

messageNVP  =  quotation-mark "message" quotation-mark name-seperator
               begin-object
               langNVP
               value-seperator valueNVP
               end-object
  ; Human readable description of the error

langNVP       =  quotation-mark "lang" quotation-mark name-seperator string
  ;  A string as per [RFC4646]

valueNVP      =  quotation-mark "value" quotation-mark name-seperator string
  ; Human readable message describing the error

innererrorNVP = quotation-mark
                "innererror" quotation-mark name-seperator object
 ; Data service defined debugging information
 ; This name/value pair should only be used in development environments.
 ; If present in a response from a production data service, it should not
 ; include system internal information in order to guard against information
 ; disclosure security concerns.

begin-object  =  ; see [RFC4627] section 2
end-object    =  ; see [RFC4627] section 2
object        =  ; see [RFC4627] section 2.2
string        =  ; see [RFC4627] section 2.5
```

Listing: ABNF Grammar for Top-level Error Payloads Formatted by Using Verbose JSON

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

The abstract data model used by data service clients is described in Abstract Data Model (section 2.2.1).

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

Clients MUST adhere to the request generation and response processing rules (for the client role) defined in AtomPub [RFC5023] and HTTP [RFC2616]. This section defines the additional rules, specific to this document, to which clients MUST adhere.

3.1.4.1 Common Rules for All Requests

Requests sent by clients MUST NOT specify any of the HTTP headers or tokens defined in HTTP Header Fields (section 2.2.5) or [RFC2616] which are defined for use only in responses.

In order to make use of the versioning scheme defined in Versioning and Capability Negotiation (section 1.7), a client SHOULD specify the DataServiceVersion (section 2.2.5.3) and MaxDataServiceVersion (section 2.2.5.7) headers in all requests.

For all request types in which the response can include a response body, the client SHOULD specify the Accept (section 2.2.5.1) header or Format System Query Option ($format) (section 2.2.3.6.1.5) to control the content type of the response. If no Accept (section 2.2.5.1) header or $format query string option is specified, the data service SHOULD return an application/atom+xml (section 2.2.5.1.1)-based response. If an application/atom+xml (section 2.2.5.1.1) response, as specified in AtomPub Format (section 2.2.6.2), is not defined for the target resource, then the service SHOULD use application/xml as the response format (see XML Format (section 2.2.6.5)).

After sending a request, the client MUST wait for another higher-layer triggered event or for the response to the request to be received.

3.1.4.2 Request to Insert Resources

When a higher layer needs to insert entities into a data service, it MUST cause the client to send the appropriate Insert request type, as specified in Insert Request Types (section 2.2.7.1) and as detailed in the remainder of this section.

The higher layer MUST provide the URI, which the higher layer obtained from a prior data service response or other means that will be specified in all requests sent by the client. As defined by the Insert request types, Insert Request Types (section 2.2.7.1), the URI MUST identify the EntitySet or collection of entities that the new EntityType instance is to be inserted into. In addition, the higher layer MAY specify the preference/hint about whether to include an entity that represents the current

state of the resource in the response to a successful insert request by using the Prefer (section 2.2.5.9) header.

3.1.4.2.1 Sending an InsertEntity Request

The InsertEntity Request (section 2.2.7.1.1) MUST adhere to the syntax specified in InsertEntity Request (section 2.2.7.1.1).

The client SHOULD specify the Content-Type (section 2.2.5.2) header in the request, and the media type specified in the header value MUST be application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), or application/json;odata=verbose (section 2.2.5.1.3).

If the higher layer requests to bind the new entity specified in the request body to one or more existing entities, the client MUST indicate this by specifying values for the NavigationProperties, which represent the association between the new and existing entity, as specified in InsertEntity Request (section 2.2.7.1.1).

If the higher layer requests to insert the entity specified in the request body as well as additional related entities, the client MUST indicate this by also specifying the related entities in the request payload, as specified in InsertEntity Request (section 2.2.7.1.1).

If the new entity is to be inserted in an EntitySet whose base EntityType does not define any subtypes, the representation of the new entity MAY specify its **EntityType** in the request payload. However, if the new entity is to be inserted in an **EntitySet** with a base **EntityType** that does have subtypes, the representation of the new entity (in the request) MUST specify its **EntityType** if it is not the base **EntityType**. InsertEntity Request (section 2.2.7.1.1) describes the syntax rules for specifying an entity's type information in a request.

Rules for processing the response to an InsertEntity are specified in Responses from Insert Requests (section 3.1.5.2).

3.1.4.2.2 Sending an InsertLink Request

The InsertLink request MUST adhere to the syntax specified in InsertLink Request (section 2.2.7.1.2).

The client SHOULD specify the Content-Type (section 2.2.5.2) header in the request and the media type specified in the header value MUST be application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), or application/json;odata=verbose (section 2.2.5.1.3).

Rules for processing the response to an InsertLink Request (section 2.2.7.1.2) are specified in Common Rules for Receiving Responses from Data Service Requests (section 3.1.5.1).

3.1.4.3 Request to Retrieve Resources

When a higher layer needs to retrieve resources from a data service, it MUST cause the client to send the appropriate Retrieve Request Type (section 2.2.7.2), as specified in the remainder of this section.

The higher layer MUST provide the URI, which the higher layer obtained from a prior data service response or other means that will be specified in all requests sent by the client. As defined by the Retrieve Request Types (section 2.2.7.2), the URI MUST identify the data service resources to be retrieved. Such a URI MAY include Query Options (section 2.2.3.6) which further define the resources to be retrieved.

3.1.4.3.1 Common Rules for Sending Retrieve Requests

This section defines all the rules to which clients MUST adhere when sending Retrieve requests. Therefore, no additional rules for specific Retrieve request types are defined in this document.

Retrieve requests MUST adhere to the syntax rules that are specified in Retrieve Request Types (section 2.2.7.2).

As noted in Common Rules for all Requests (section 3.1.4.1), the client SHOULD specify the Accept (section 2.2.5.1) header or the Format system query option ($format) (section 2.2.3.6.1.5) to specify its preference regarding the content type used in the response.

If the higher layer causes the client to send a RetrieveEntitySet request or RetrieveEntity request, the media type(s) specified by the client in the Accept (section 2.2.5.1) header or the Format system query option (section 2.2.3.6.1.5) SHOULD include application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), and/or application/json;odata=verbose (section 2.2.5.1.3). If the request type is RetrieveComplexType, RetrievePrimitiveProperty, or RetrieveLink, the media type(s) specified in the Accept (section 2.2.5.1) header or the $format query string option SHOULD include application/xml, application/json (section 2.2.5.1.2), and/or application/json;odata=verbose (section 2.2.5.1.3). If the request is a RetrieveValue or RetrieveMediaResource, the required media type(s) are data dependent and are not defined by this document. If the request is a RetrieveServiceDocument request, the required media types SHOULD include application/xml and application/atomsvc+xml, application/json (section 2.2.5.1.2), and/or application/json;odata=verbose (section 2.2.5.1.3). Finally, if the request is a RetrieveCount request, the media type specified in the Accept (section 2.2.5.1) header or the $format query string option MUST be text/plain.

If the retrieve request URI (provided by the higher layer) identifies an entity, ComplexType, primitive property, or property value, and the associated EntityType defines a concurrency token, the client MAY include an If-None-Match in the request. Even if the request URI includes a $expand query option, the concurrency token provided as the value of the If-None-Match header MUST be a token associated with the **EntityType** of the entity identified by the request URI. RetrieveLink requests MUST NOT include an If-None-Match or If-Match header.

If the retrieve request URI identifies a Media Resource, the request MAY include an If-None-Match (section 2.2.5.6) header.

For rules for processing responses, see Retrieve Request Types (section 2.2.7.2).

3.1.4.4 Request to Update Resources

When a higher layer needs to update an existing resource in a data service, it MUST cause the client to send the appropriate Update Request Type (section 2.2.7.3), as specified in the remainder of this section.

The higher layer MUST provide the request URI, which it obtained from a prior data service response or other means, as specified in Update Request Types (section 2.2.7.3). The URI MUST identify the data service resource to be updated.

3.1.4.4.1 Common Rules for Sending Update Requests

This section defines all the rules to which clients MUST adhere when sending Update requests. Therefore, no additional rules for specific Update request types are defined in this document.

Update requests MUST adhere to the syntax rules specified in Update Request Types (section 2.2.7.3).

The client SHOULD specify the Content-Type (section 2.2.5.2) header in the request. If the higher layer issues an UpdateEntity request, the media type specified by the client in the Content-Type header MUST be application/atom+xml (section 2.2.5.1.1), application/json (section 2.2.5.1.2), or application/json;odata=verbose (section 2.2.5.1.3). If the request type is UpdateComplexType or UpdatePrimitiveProperty, the media type specified MUST be application/xml or application/json;odata=verbose (section 2.2.5.1.3). Finally, if the request is an UpdateValue or UpdateMediaResource request, a required media type is data dependent and not defined by this document.

If the Update request URI (provided by the higher layer) identifies an entity, ComplexType instance, primitive property, or property value, and the associated EntityType defines a concurrency token, the client SHOULD include an If-Match (section 2.2.5.5) header in the request. The concurrency token provided as the value of the header MUST be a token that is associated with the EntityType of the entity identified by the request URI. UpdateLink requests MUST NOT include an If-Match (section 2.2.5.5) header.

If the update request URI identifies a Media Resource, the request MAY include an If-Match (section 2.2.5.5) header. If an If-Match header is included, the value of the header MUST be the concurrency token associated with the Media Resource.

Use of the If-None-Match request header with Update request types is undefined by this document.

If the data service identified by the request URI supports tunneled requests, any of the Update Request Types (section 2.2.7.3) MAY be sent as a tunneled request (section 2.2.7.7).

Rules for processing responses to Update requests are specified in Update Request Types (section 2.2.7.3).

The client MAY specify a preference/hint about whether to include an entity that represents the current state of the resource in the response to a successful update request by using the Prefer (section 2.2.5.9) header.

3.1.4.5 Request to Delete Resources

When the higher layer needs to delete an existing resource in a data service it MUST cause the client to send the appropriate Delete Request Type (section 2.2.7.4), as specified in the remainder of this section.

The higher layer MUST provide the request URI, which the higher layer obtained from a prior data service response or other means. The URI MUST identify the data service resource to be deleted.

3.1.4.5.1 Common Rules for Sending Delete Requests

This section defines all the rules to which clients MUST adhere when sending Delete requests. Therefore, no additional rules for specific Delete request types are defined in this document.

Delete requests MUST adhere to the syntax rules specified in Delete Request Types (section 2.2.7.4).

If the delete request URI (provided by the higher layer) identifies an EntityType instance for a type that has a concurrency token defined, then the client SHOULD include an If-Match (section 2.2.5.5) header in the request. The concurrency token provided as the value of the If-Match header MUST be a token associated with the EntityType of the entity identified by the request URI. DeleteLink requests MUST NOT include an If-Match (section 2.2.5.5).

Use of the If-None-Match request header with Delete request types is not defined in this document.

If the data service identified by the request URI supports tunneled requests, then any of the Delete Request Types (section 2.2.7.4) MAY be sent as a tunneled request (section 2.2.7.7).

Rules for processing the response to Delete Requests is specified in Common Rules for Receiving Responses from Data Service Requests (section 3.1.5.1).

3.1.4.6 Request to Invoke a Service Operation

When the higher layer needs to invoke a Service Operation in a data service, it MUST cause the client to send the appropriate Invoke Request (section 2.2.7.5) type.

The higher layer MUST provide the request URI, which the higher layer obtained from a prior data service response or other means. As defined by the Invoke request types, Invoke Request (section 2.2.7.5), the URI MUST identify the service operation to be invoked.

Invoke requests MUST adhere to the syntax rules specified in Invoke Request (section 2.2.7.5).

If the Invoke request URI provided by the higher layer identifies a Service Operation that returns a collection of EntityType instances, system query options and additional resource path segments might be included in the request URI, as described in Query Options (section 2.2.3.6) and Resource Path: Semantics (section 2.2.3.5).

Request headers MAY be provided with Invoke request types. However, this document defines no additional meaning or semantics to such request headers.

If the data service identified by the request URI supports tunneled requests, then an Invoke Request (section 2.2.7.5) MAY be sent as a tunneled request (section 2.2.7.7).

Rules for processing the response to Service Operation requests are specified in Common Rules for Receiving Responses from Data Service Requests (section 3.1.5.1).

3.1.4.7 Request to Send a Batch of Operations

When the higher layer needs to send a batch of operations to a data service, it MUST cause the client to send a Batch Request (section 2.2.7.6), as specified in the remainder of this section.

The higher layer MUST provide the request URI, which the higher layer obtained from a prior data service response or other means. The request URI MUST identify the data service's batch endpoint, as specified in **Batch Request**.

Batch requests MUST adhere to the syntax rules specified in **Batch Request**.

Each operation sent within a **Batch Request** MUST follow the guidelines for the associated request types specified in section 3.1.4.

3.1.4.8 Request to Invoke an Action

Applies to the OData 3.0 protocol

When the higher layer needs to invoke an action in a data service, it MUST cause the client to send the appropriate Invoke Action request (section 2.2.7.5.1) type, as specified in the remainder of this section.

The higher layer MUST provide the request URI, which the higher layer obtained from a prior data service response or other means. As defined in Invoke Action Request (section 2.2.7.5.1), the URI MUST identify the Action to be invoked.

Requests to invoke an action MUST adhere to the syntax rules specified in Invoke Action Request (section 2.2.7.5.1).

Request headers MAY be provided with Invoke Action request types. However, this document defines no additional meaning or semantics to such request headers.

If the data service identified by the request URI supports tunneled requests and the action requires no parameter to be passed in the POST body, then an Invoke Action Request (section 2.2.7.5.1) MAY be sent as a tunneled request (section 2.2.7.7).

Rules for processing the response to an Invoke Action request are specified in Common Rules for Receiving Responses from Data Service Requests (section 3.1.5.1) and Executing a Received Invoke Action Request (section 3.2.5.9).

3.1.4.9 Request to Invoke a Function

Applies to the OData 3.0 protocol

When the higher layer needs to invoke a function in a data service, it MUST cause the client to send the appropriate Invoke Function request (section 2.2.7.5.2) as specified in the remainder of this section.

The higher layer MUST provide the request URI, which the higher layer obtained from a prior data service response or other means. As defined by the Invoke Function Request (section 2.2.7.5.2), the URI MUST identify the function to be invoked.

Requests to invoke a function MUST adhere to the syntax rules specified in Invoke Function Request (section 2.2.7.5.2).

Request headers MAY be provided with Invoke Function request types. However, this document defines no additional meaning or semantics to such request headers.

Rules for processing the response to Invoke function requests are specified in Common Rules for Receiving Responses from Data Service Requests (section 3.1.5.1) and Executing a Received Invoke Function Request (section 3.2.5.10).

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Common Rules for Receiving Responses from Data Service Requests

This section defines a common set of rules to which a client MUST adhere when receiving a response to any data service request that is sent.

The client MUST verify that the response adheres to the syntax specified in the subsection of Request Types (section 2.2.7) that defines the response associated with the request type.

If the HTTP status code indicates that the request completed in error, the client MUST validate and interpret the response body (if one is present) by using the syntax rules in Error Response (section 2.2.8.1).

If the HTTP status code indicates that the request succeeded, the client MUST validate and interpret the response body (if one is present) by using the syntax rules defined in the appropriate subsection(s) of Common Payload Syntax (section 2.2.6).

If a Content-Type (section 2.2.5.2) header is present in the response and its value is not one of the valid values for an Accept (section 2.2.5.1) request header on the associated request type, then the client SHOULD report an error to the higher layer.

If a DataServiceVersion (section 2.2.5.3) header is included in the response, the client MUST validate that the version number in the header is lower than or equal to the highest version of this document that it understands. If it is lower than or equal to the highest version of this document that the client understands, the client continues to process the request. If the version number in the header is higher than the highest version of this document that the client can interpret, the client MUST report an error to the higher layer.

If no DataServiceVersion (section 2.2.5.3) header is present in the response, then the client SHOULD interpret the request by using the highest version number of this document that it understands. For additional details on the versioning scheme defined by this document, see Versioning and Capability Negotiation (section 1.7).

If the ETag (section 2.2.5.4) header is present in the response, the client SHOULD report the value of the header (which represents the concurrency token associated with the resource on which the

request acted) to the higher layer such that the value can be used in a future request, or in an If-Match (section 2.2.5.5) or If-None-Match (section 2.2.5.6) request header.

3.1.5.2 Responses from Insert Requests

In addition to the common response processing rules in Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1), the client MUST interpret the response body of a response to an Insert request in Insert Request Types (section 2.2.7.1) as containing the most recent values (as known to the client) for the inserted entity. Such behavior allows a data service to alter the data (by using, for example, data service specific validation rules) it received in an Insert Request and reflect the alterations back to the client.

As specified in [RFC5023], the client MUST use the URI returned in the Location response header as the request URI in subsequent requests to the entity.

In OData 3.0, it is possible for the client to express the preference of receiving no response to an insert request (see Prefer (section 2.2.5.9)). In this case, the data service alters the data it received in an Insert request and the alterations are not reflected back to the client.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

If the TCP connection to the server is disconnected after the client has sent a request but before it has completely received the response, and the client did not initiate the disconnection, the client SHOULD report this event as an error to the higher layer.

3.2 Server Details

3.2.1 Abstract Data Model

The abstract data model used by data services is described in Abstract Data Model (section 2.2.1).

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Common Rules for Receiving All Data Service Requests

This section outlines a set of directives that a server MUST follow when processing requests. In addition, further directives specific to each request type are defined in the following sections.

If a request includes a DataServiceVersion (section 2.2.5.3) header, the server MUST validate that the header value is correctly formatted according to the rules in DataServiceVersion (section 2.2.5.3) and that the version number provided in the header is less than or equal to the maximum version of this document the data service implements. If so, then the server MUST interpret the request as defined by the protocol version specified in the header. If the version number in the DataServiceVersion (section 2.2.5.3) header of the request is larger than the maximum version number the server implements or is malformed, the server MUST return a 4xx error response code. If a request does not specify a DataServiceVersion (section 2.2.5.3) header, the server MUST interpret the request as defined by the highest protocol version number the server understands.

If the request includes a MaxDataServiceVersion (section 2.2.5.7) header, the server MUST parse and validate the header value to ensure it adheres to the syntax specified in MaxDataServiceVersion (section 2.2.5.7). If the header value is malformed, the server MUST return a 4xx error response code. If a valid version number is provided, the server MUST use the version number as specified in Versioning and Capability Negotiation (section 1.7), which outlines this document's versioning scheme. If no MaxDataServiceVersion (section 2.2.5.7) header is present, the server SHOULD assume the client understands all the protocol versions that the server implements.

If the request includes an If-Match (section 2.2.5.5) or an If-None-Match (section 2.2.5.6) header but the EntityType associated with the resource identified by the request URI, referred to as the "target EntityType", does not define a concurrency token, then the server MUST return a 4xx error response. The one exception to this rule is that if an If-Match header with a value of "*" is present on the request, then a data service MAY accept the request.

If the request includes an If-Match (section 2.2.5.5) header, it MUST be parsed and processed according to If-Match (section 2.2.5.5). If the value of the header is "*" and the target **EntityType** defines a concurrency token, the request is processed as if the concurrency check succeeded.

Servers MAY<69> choose to implement access control policies where certain requests are rejected based on the requesting identity, the target resource, and potentially other environmental information.

3.2.5.2 Common Rules for Executing Received Insert, Update, or Delete Data Service Requests

This section outlines a set of directives that a server MUST follow when processing requests of the Insert, Update, or Delete requests types (whether those requests are sent standalone, tunneled, or are included in Batch Requests). In addition, further directives specific to each request type are defined in the following sections.

Servers SHOULD respond to any HTTP PUT, HTTP MERGE, HTTP PATCH, HTTP POST, or HTTP DELETE requests sent to a URI that does not support the method with a 405 (Method Not Allowed) response.

When executing an HTTP request that is intended to change the state of a resource, if the server determines that executing the operation would require violating the rules defined by the abstract data model which describes the data service, as defined in Request Types (section 2.2.7), it MUST respond with a 4xx response code.

When executing an HTTP request that is intended to change the state of a resource, the server SHOULD execute the requested changes in an order such that any interim changes that are externally visible while the request is being executed maintain the consistency of the data model of the data service. If no such ordering can be determined, the request SHOULD be rejected with a 5xx response code.

If the resource identified by the request URI defines a concurrency token, but the request does not include an If-Match (section 2.2.5.5) request header, the server returns a 4*xx* response code.

The execution of an HTTP request that is intended to change the state of a resource MAY<70> have additional side effects on that resource, on other resources accessible through the data service interface, or on data or processes outside of the data service itself.

3.2.5.2.1 Common Rules for Executing Requests Containing a Customizable Feeds Mapped Property

Applies to the OData 2.0 and OData 3.0 protocols

In OData 2.0 and OData 3.0, it is possible to map the value of a property on an **EntityType** to another location in the feed.

If a property mapping that has been defined in the CSDL contains an **FC_KeepInContent** attribute with value "false", the value of the property MUST NOT be included inside the **m:properties** element in the response.

If all property mappings that have been defined in the CSDL on **EntityTypes** that are addressed by the request contain an **FC_KeepInContent** attribute with value "true", the data service SHOULD respond with an OData 1.0 response, provided that no other aspect of the response would cause the version of the response to be higher. If any **EntityType** addressed by the request has a property mapping defined on it in the CSDL with an **FC_KeepInContent** attribute with a value of "false", the data service MUST respond with an OData 2.0 or an OData 3.0 response.

For OData 2.0, if an **EntityType** instance included in a data service response contains a mapped property that has a value of null, the property element for that property MUST be included in the **m:properties** element and have an attribute of **m:null** with a value of "true". If an **EntityType** instance included in a data service response contains a mapped property that has a value of null, the element being mapped to MAY still be present and MUST have empty content. If the property value is included in the **m:properties** element (as described in section 2.2.6.2.2), the data service MUST use that value when updating or inserting the **EntityType** instance.

The format of the URI in a request does not change when the request is being made to an **EntityType** instance or collection of **EntityType** instances with a customizable feed property mapping. System query options and service operations that accept a property name as a parameter MUST be addressed by using the name of a property identified on an **EntityType** and not the mapped location of that property.

3.2.5.3 Executing a Received Insert Request

The directives defined in this section apply when executing a received request of any of the Insert request types defined in Insert Request Types (section 2.2.7.1).

The server MUST validate the HTTP request URI identified as an EntitySet, collection of entities, or a collection of Links, as defined by the data service's data model in Abstract Data Model (section 2.2.1). If this validation fails, a 4xx error response code MUST be returned, as specified in Common Response Codes (section 3.2.8). If the validation succeeds, the server MUST insert the new entity, Media Resource, or Link as appropriate, based on the description in Insert Request Types (section 2.2.7.1).

If an Insert request is received with a null value for a data service resource and the type of that resource is not nullable (as defined by the Entity Data Model associated with the data service) then the server MUST return a 4*xx* response code, as specified in Common Response Codes (section 3.2.8).

If an Insert request is received with an empty value for a data service resource and the type of that resource does not define an empty state, the server MUST return a 4xx response code, as specified in Common Response Codes (section 3.2.8).<71>

Any inlined content (section 2.2.6.2.6.1) in a request payload MUST be treated as a "deep insert" as specified in InsertEntity Request (section 2.2.7.1.1), inlined content (section 2.2.6.2.6.1), and Deferred Content (section 2.2.6.3.9).

If the request URI does not match associated URIs anywhere in the request payload where URIs are expected, then the request URI takes precedence and the payload SHOULD be treated as if the URIs in it matched the value of the request URI.

A data service MAY alter or ignore any of the values provided in the request payload before performing the insert operation.

3.2.5.3.1 Executing a Received InsertEntity Request

In addition to the directives specified in sections Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1), Common Rules for Executing Received Insert, Update, or Delete Data Service Requests (section 3.2.5.2), and Executing a Received Insert Request (section 3.2.5.3), the directives defined in this section apply when executing an InsertEntity Request (section 2.2.7.1.1).

Any data in the request payload not required to complete the insert operation SHOULD be ignored by a data service. If a data service ignores a particular payload construct, the client MAY<72> omit the construct from the request payload.

If the entity represented in the request payload is an instance of an OpenEntityType, then the property values in the request payload, in addition to those that represent the values of declared properties on the **OpenEntityType**, MUST be treated as values of dynamic properties associated with the **OpenEntityType** instance being inserted.

In OData 1.0 and OData 2.0, if the insert succeeds in full, the server MUST return a response with a 201 (Created) status code and a response body that conforms to the syntax specified in InsertEntity Request (section 2.2.7.1.1). The response body MUST contain the values of the inserted resource after the server has executed all its server-specific data processing rules (validation, and so on). The server MAY alter the values of the resource received from the client before the resource is inserted on the server.

In OData 3.0, the response MAY have a 204 status code, as specified in [RFC2616], based on the client preference (see Prefer (section 2.2.5.9)) on the InsertEntity Request.

3.2.5.3.2 Executing a Received InsertLink Request

In addition to the directives specified in Common Rules for Receiving All Data Service Requests (section 3.2.5.1), Common Rules for Executing Received Insert, Update, or Delete Data Service Requests (section 3.2.5.2), and Executing a Received Insert Request (section 3.2.5.3), the directives defined in this section apply when executing an InsertLink request (section 2.2.7.1.2).

In OData 1.0 and OData 2.0, if the insert succeeds in full, the server MUST return a 204 (No Content) response code. In OData 3.0, the response MAY have a 201 response code, as specified in [RFC2616], if the UpdateLink response body includes the updated link. Then, the response will have a response body that is the same as the response body to a RetrieveLink request (section 2.2.7.2.9).

3.2.5.3.3 Executing a Received InsertMediaResource Request

In addition to the directives specified in sections 3.2.5.2 and 3.2.5.3, the directives defined in this section apply when executing an InsertMediaResource Request (section 2.2.7.1.3).

In addition to inserting the provided Media Resource, the data service MUST also create an **EntityType** instance which represents the Media Link Entry for the newly inserted Media Resource.

In OData 1.0 and OData 2.0, if the insert succeeds in full, the server MUST return a response with a 201 (Created) status code and a request body that conforms to the syntax specified in InsertEntity Request (section 2.2.7.1.1). The response body MUST contain a representation of the entity created as the Media Link Entry that holds the metadata about the created Media Resource. Processing of the SLUG header [RFC5023] section 9.7 is data service-specific and is undefined by this specification.

In OData 3.0, the response MAY have a 204 response code, as specified in [RFC2616], based on the client preference (see Prefer (section 2.2.5.9)) on the InsertMediaResource Request.

3.2.5.4 Executing a Received Retrieve Request

The directives defined in this section apply when executing a received request of any of the Retrieve request types defined in Retrieve Request Types (section 2.2.7.2).

If the request includes an If-None-Match (section 2.2.5.6), the header MUST be parsed and processed, as described in If-None-Match (section 2.2.5.6). If the value of the header is "*" and the target EntityType defines a concurrency token, then a 304 (Not Modified) response MUST be returned. When retrieving a Media Resource, the target **EntityType** in the prior sentence is the **EntityType** of the entity, which is the Media Link Entry for the Media Resource being retrieved.

If the request is a RetrievePrimitiveProperty (section 2.2.7.2.4), RetrieveComplexType (section 2.2.7.2.3), or RetrieveValue (section 2.2.7.2.5) request and the request URI identifies a dynamic property that does not currently exist (on the associated OpenEntityType), then the server MUST respond as if the property existed and its value is null.

If the request includes an InlineCount system query option (section 2.2.3.6.1.10), the response SHOULD include a DataServiceVersion (section 2.2.5.3) response header that indicates that the response is using either the OData 2.0 or OData 3.0 protocol.

In addition to validating the request headers, the server MUST validate that the HTTP request URI and payload adhere to the syntax for Retrieve Request Types (section 2.2.7.2). The server validates the request based on its knowledge of the Entity Data Model associated with the service. For example, the server validates each request URI path segment against the **EntityTypes**, EntitySets, and so on, in the associated data model to ensure that each identifies a valid construct defined in the data service's Abstract Data Model (section 2.2.1) and that the request URI as a whole identifies a valid resource.

If this validation fails, the server MUST respond with a valid HTTP [RFC2616] error status code, as specified in Common Response Codes (section 3.2.8), and an error message that is formatted according to Error Response (section 2.2.8.1).

If the validation succeeds, the server MUST obtain the requested resources and return a response (with a 2*xx* response code) following the rules for the retrieve request type, as specified in Retrieve Request Types (section 2.2.7.2).

3.2.5.4.1 Executing a Received RetrieveEntitySet Request

In addition to the directives specified in sections Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1) and Executing a Received Retrieve Request (section 3.2.5.4), the directives defined in this section apply when receiving a RetrieveEntitySet request, see RetrieveEntitySet Request (section 2.2.7.2.1).

As described in AtomPub [RFC5023] section 10.1, when processing a request of this type, the server MUST determine if the response returned to the client will include all entities or a partial set of the entities identified by the request URI. Whether or not a partial collection is returned is data service dependent.

Partial collections of entities are supported only in OData 2.0 and OData 3.0. Therefore, if the response from server to client includes a representation of a partial set of the entities identified by the request URI, then the response MUST include a "next link" as defined in Entity Set (as an Atom Feed Element) (section 2.2.6.2.1), Annotation odata.nextLink ([ODataJSON4.0] section 4.5.5), or Entity Set (as a Verbose JSON array) (section 2.2.6.3.2), as appropriate. Such a response MUST also include a DataServiceVersion (section 2.2.5.3) response header that indicates which version of the OData protocol that the response is. The header indicates the OData 2.0 protocol if there are no OData 3.0 features (as defined in section 1.7.2). Otherwise, the response header indicates that the response is using OData 3.0. See sections 4.2.1.3 and 4.2.1.4 for examples of data service responses that include partial sets of entities.

3.2.5.4.2 Executing a Received RetrieveValue Request

In addition to the directives specified in sections Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1), and Executing a Received Retrieve Request (section 3.2.5.4), the directives defined in this section apply when receiving a RetrieveValue request, see RetrieveValue Request (section 2.2.7.2.5).

If the value to be returned by a response to a RetrieveValue request is null, then the server MUST respond with a 404 (Not Found) and an error message formatted according to Error Response (section 2.2.8.1).

3.2.5.4.3 Executing a Received RetrieveCount Request

Applies to the OData 2.0 and OData 3.0 protocols

In addition to the directives specified in the sections Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1) and Executing a Received Retrieve Request (section 3.2.5.4), the directives defined in this section apply when receiving a RetrieveCount request; see RetrieveCount Request (section 2.2.7.2.10).

When the InlineCount system query option is used, the resource path section of the URI MUST identify a collection of entities. If the Resource Path section of the URI does not identify a collection of entities or a single **EntityType** instance, then the data service MUST return a 4xx error response code. As well, the data service MAY support the InlineCount system query option on a URI that identifies a single **EntityType** instance.

The presence of the **$top**, **$skip**, **$orderby**, **$format**, or **$expand** query option in the data service URI MUST NOT change the count value N.

The InlineCount system query option MUST be used with an HTTP GET request type. If an InlineCount request is made by using an HTTP verb other than GET, the data service MUST return a 4*xx* error response code.

RetrieveCount requests are supported in the OData 2.0 and OData 3.0 protocols. A response to a RetrieveCount request MUST also include a DataServiceVersion (section 2.2.5.3) response header to indicate which version of the OData protocol that the response is using.

3.2.5.5 Executing a Received Update Request

The directives defined in this section apply when executing a received request of any of the Update request types that are defined in Update Request Types (section 2.2.7.3).

The server MUST validate the HTTP request URI identified as an existing EntityType, ComplexType instance, primitive property, Media Resource, or Link, as defined by the data service's data model in Abstract Data Model (section 2.2.1). If this validation fails, a 4*xx* error response code MUST be returned, as specified in Common Response Codes (section 3.2.8). If the validation succeeds, the server MUST update the value of the resource identified by the request URI with the values specified in the request's payload.

If the Update request used the **HTTP PUT** method, the request MUST be processed by first setting the resource identified in the request URI to its default value(s) and then updating the default value(s) with those provided in the request payload.

If the Update request used the HTTP PATCH or HTTP MERGE method, the request MUST be processed using a merge-based update, as specified in PATCH/MERGE (section 2.2.4.1).

The HTTP MERGE, HTTP PATCH, and HTTP PUT methods are defined by this document to have identical semantics for UpdateProperty (section 2.2.7.3.3), UpdateLink (section 2.2.7.3.6), and UpdateValue (section 2.2.7.3.5) request types. Thus, when servers execute such requests, there MUST NOT be any observable difference (from the client's perspective) between a successful response

to a request made by using the HTTP MERGE method and the same request made by using the HTTP PUT method.

If an Update request is received that would set the value of a data service resource to null when the type of that resource is not nullable, as defined by the Entity Data Model associated with the data service, then the server MUST return a 4*xx* response code, as specified in Common Response Codes (section 3.2.8).

If an Update request is received that would set the value of a resource to empty when the type of that resource does not define an empty state, the server MUST return a 4*xx* response code, as specified in Common Response Codes (section 3.2.8).<73>

Any inlined content in an Update request payload SHOULD be ignored, as specified in Inline Representation (section 2.2.6.2.6.1), Deferred Content (section 2.2.6.3.9), and Inline Representation (section 2.2.6.3.9.1).

If the request URI does not match associated URIs that are anywhere in the request payload where URIs are expected, then the request URI takes precedence and the payload MAY be treated as if the URIs in it match the value of the request URI.

If the payload of an Update request contains, as part of the serialization of a resource, one or more of the key properties for the associated EntityType, those key values MUST be ignored by the server because EntityKeys are immutable.

If the request URI identifies a property P of an OpenEntityType instance and P does not represent a declared property or NavigationProperty of the entity, then P MUST be considered to represent a dynamic property, and the request represents a request to update the value of P.

In OData 1.0 and OData 2.0, if the update succeeds in full, the server MUST return a 204 (No Content) response code.

In OData 3.0, the response MAY have a 200 response code, as specified in [RFC2616], if the response body includes the updated values as they exist on the server.

3.2.5.5.1 Executing a Received UpdateEntity Request

In addition to the directives specified in Common Rules: Receiving Responses from Data Service Requests (section 3.1.5.1), Common Rules for Executing Received Insert, Update, or Delete Data Service Requests (section 3.2.5.2), and Executing a Received Update Request (section 3.2.5.5), the directives defined in this section apply when executing any received request of the UpdateEntity request type UpdateEntity Request (section 2.2.7.3.1).

If the request payload includes (re)binding information, the server MUST rebind the **EntityType** instance being updated to the existing entities specified in the request payload. A rebind request can be included in the request payload for each **NavigationProperty** defined on the associated **EntityType**.

If the entity represented in the request payload is an instance of an **OpenEntityType**, then the property values in the request payload, in addition to those that represent the values of declared properties on the **OpenEntityType**, MUST be treated as values of dynamic properties associated with the **OpenEntityType** instance being inserted.

An update MUST only be considered successful if the target **EntityType** instance's **EDMSimpleType** and **ComplexType** property values are updated (with the values specified in the request payload) and all rebind operations are completed successfully. If all requested updates are not completed, an error response code MUST be returned.

3.2.5.6 Executing a Received Delete Request

The directives defined in this section apply when executing a received request of any of the Delete request types defined in Delete Request Types (section 2.2.7.4).

The server MUST validate the HTTP request URI identifies an existing EntityType instance, an **EntityType** instance property value, or a link. If this validation fails, a 4*xx* error response code MUST be returned, as specified in Common Response Codes (section 3.2.8). If the validation succeeds, the server MUST delete the resource (entityor link) identified by the request URI and return a response following the rules for the request type as specified in Delete Request Types (section 2.2.7.4).

This document defines the semantics of Delete request types (section 2.2.7.4) such that the resource identified by the request URI MUST no longer be available at that URI after the request successfully completes. This does not imply any mandatory action by the server in regard to the underlying resource; however, it SHOULD be deleted or moved to an alternate location (such as the recycle bin).

[RFC2616] stipulates that "A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity". A data service MUST complete the requested action prior to responding to a Delete request, and thus, successful delete operations MUST always return a 204 (No Content) response code.

If a Delete request type (section 2.2.7.4) includes a request payload, it MUST be ignored and the request MUST be treated as if no entity body was provided.

3.2.5.7 Executing a Received Invoke Request

The directives defined in this section apply when executing a received Invoke request, as specified in Invoke Request (section 2.2.7.5).

The server MUST validate that the HTTP request URI identifies a service operation exposed by the data service. If this validation fails, a 4*xx* response code MUST be returned. If the validation succeeds, the server MUST read all required input parameter values from the request URI query string. If any of the parameter values are malformed, a 4*xx* response code MUST be returned. If the request URI is valid, the server MUST invoke the **FunctionImport**, as specified in [MC-CSDL], associated with the service operation by using the parameter values specified in the request URI.

If a **FunctionImport** requires an input parameter not present in the request URI, the server SHOULD pass null for the parameter value to the function. If the parameter type is not nullable, as specified in [MC-CSDL], a 4*xx* response code MUST be returned.

If the **FunctionImport**, as specified in [MC-CSDL], is invoked successfully, the return value MUST be serialized according to the syntax rules defined in Invoke Request (section 2.2.7.5).

3.2.5.8 Executing a Received Batch Request

The directives defined in this section apply when executing a received Batch request as specified in Batch Request (section 2.2.7.6).

The server MUST validate the HTTP request using the "multipart/mixed" content type, and the request MUST conform to the syntax for Batch requests defined in Batch Request (section 2.2.7.6). If this validation fails, a 4*xx* error response code MUST be returned, as specified in Common Response Codes (section 3.2.8). If the validation succeeds, the server MUST parse the batch request into a set of individual operations and perform each operation specified in the batch according to the semantics outlined in Batch Request (section 2.2.7.6).

Each MIME part within the Batch request using the application/http content type has to be processed according to the "executing rules" for the request type defined in Message Processing Events and Sequencing Rules (section 3.2.5).

If the headers of a Batch request are received successfully, a data service MUST respond with a 202 (Accepted) response code allowing the service to signal that part of the request is received, but that processing of the request has not yet occurred. The response to a Batch request MUST adhere to the syntax and rules defined in Batch Responses (section 2.2.7.6.6).

3.2.5.9 Executing a Received Invoke Action Request

The directives defined in this section apply when executing a received Invoke Action request, as specified in Invoke Action Request (section 2.2.7.5.1).

The server MUST validate that the HTTP request URI identifies an action exposed by the data service. If this validation fails, a 4*xx* response code MUST be returned. If the validation succeeds, the server MUST read all required input parameter values from the request body (section 2.2.7.5.1). If any of the parameter values are malformed, a 4*xx* response code MUST be returned. If the request URI is valid, the server MUST invoke the **FunctionImport**, as specified in [MC-CSDL], associated with the action by using the parameter values specified in the request URI.

If a **FunctionImport** requires an input parameter not present either in the request body or as a binding parameter prefix, the server SHOULD pass null for the parameter value to the function. If the parameter type is not nullable, as specified in [MC-CSDL], a 4*xx* response code MUST be returned.

If the **FunctionImport**, as specified in [MC-CSDL], is invoked successfully, the return value MUST be serialized according to the syntax rules defined in Invoke Request Types (section 2.2.7.5).

If the Invoke Action request is bound to a resource that does not exist, the server MUST return a 404 response code.

If an Invoke Action request is received in which the action name is not the last segment in the URI path or the action identified is not available on the current resource, the server MUST return a 405 (Method Not Allowed) response code.

If the Invoke Action request includes an If-Match (section 2.2.5.5) header and the request URI binds the action to a resource, the server MUST verify that the bound resource has not changed by comparing its current ETag with the ETag included in the If-Match header. If the resource has changed, the server MUST return a 412 (Precondition Failed) response code, in accordance with the rules specified in [RFC2616].

3.2.5.10 Executing a Received Invoke Function Request

The directives defined in this section apply when executing a received Invoke Function request, as specified in Invoke Function Request (section 2.2.7.5.2).

The server MUST validate that the HTTP request URI identifies a function exposed by the data service. If this validation fails, a 4*xx* response code MUST be returned. If the validation succeeds, the server MUST read all required input parameter values from the request URI (section 2.2.7.5.2).

If the server has multiple **FunctionImport** elements with the same Name (that is, a function has overloads), the server SHOULD use the parameter types and parameter names that are provided to resolve which **FunctionImport** is to be invoked. If the server fails to resolve the request to a specific **FunctionImport**, the server MUST not execute any **FunctionImport** and MUST instead return a 4*xx* response code.

If any of the parameter values are malformed, a 4*xx* response code MUST be returned. If the request URI is valid, the server MUST invoke the **FunctionImport**, as specified in [MC-CSDL], associated with the function by using the parameter values specified in the request URI.

If a **FunctionImport** is invoked with no parameter names mentioned in the URI path and a required parameter is not present in the query string, the server SHOULD pass null for the parameter value to the function.

If the parameter type is not nullable, as specified in [MC-CSDL], a 4*xx* response code MUST be returned.

If the **FunctionImport**, as specified in [MC-CSDL], is invoked successfully, the return value MUST be serialized according to the syntax rules defined in Invoke Request Types (section 2.2.7.5).

If the Invoke Function request is bound to a resource that does not exist on the server, the server MUST return a 404 response code.

If the Invoke Function Request is bound to a resource that does not support the function, the server MUST return a 405 (Method Not Allowed) response code.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

If the TCP connection to the client is disconnected after the server started processing a received request but before it has completely finished processing the request, and the server did not initiate the disconnection, the server SHOULD complete the requested action even though all of the results cannot be returned to the client.

3.2.8 Common Response Codes

This section summarizes the response codes a data service uses to indicate various conditions. A data service can use alternate or additional response codes. If multiple conditions apply, the HTTP [RFC2616] and AtomPub specification [RFC5023] are consulted to determine the most appropriate response code. Additional information about the following response codes is provided in [RFC2616].

200 (OK): Indicates that a request has been received and processed successfully by a data service and that the response includes a non-empty response body.

202 (Accepted): Indicates that a batch request has been accepted for processing, but that the processing has not been completed.

204 (No Content): Indicates that a request has been received and processed successfully by a data service and that the response does not include a response body.

400 (Bad Request): Indicates that the payload, request headers, or request URI provided in a request are not correctly formatted according to the syntax rules defined in this document.

404 (Not Found): Indicates that a segment in the request URI's Resource Path does not map to an existing resource in the data service. A data service MAY<74> respond with a representation of an empty collection of entities if the request URI addressed a collection of entities.

405 (Method Not Allowed): Indicates that a request used an **HTTP** method not supported by the resource identified by the request URI, see Request Types (section 2.2.7).

412 (Precondition Failed): Indicates that one or more of the conditions specified in the request headers evaluated to false. This response code is used to indicate an optimistic concurrency check failure, see If-Match (section 2.2.5.5) and If-None-Match (section 2.2.5.6).

500 (Internal Server Error): Indicates that a request being processed by a data service encountered an unexpected error during processing.

4 Protocol Examples

All examples in this section use the sample data model and instance data shown in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

4.1 Insert a New Entity

Detailed examples for inserting a new entity are provided in Examples (section 2.2.7.1.1.1).

4.2 Retrieve Resources

4.2.1 Retrieve a Collection of Entities

4.2.1.1 Retrieve a Collection of Entities by Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to retrieve a collection of Order entities related to a specific Customer entity from a data service.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=feed
Content-Length: nnn
DataServiceVersion:3.0

<feed  xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
        <title type="text">Orders</title>
        <id>http://host/service.svc/Customers('ALFKI')/Orders</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Orders" href="Customers('ALFKI')/SampleModel.Customer
/Orders" />
        <entry>
          <category term="SampleModel.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Orders(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Orders" href="Orders(1)/SampleModel.Order " />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(1)/SampleModel.Order/Customer" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                type="application/xml" title="Customer"
```

```
                href="Orders(1)/ SampleModel.Order/$links/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/SampleModel.Order/OrderLines" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml" title="OrderLines"
                href="Orders(1)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
               <m:properties>
                 <d:OrderID m:type="Edm.Int32">1</d:OrderID>
                 <d:ShippedDate m:type="Edm.DateTime">1997-08-25T00:00:00</d:ShippedDate>
               </m:properties>
            </content>
          </entry>
          <entry>
            <category term="SampleModel.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
            <id>http://host/service.svc/Orders(2)</id>
            <title type="text" />
            <updated>2008-03-30T21:52:45Z</updated>
            <author>
              <name />
            </author>
            <link rel="edit" title="Orders" href="Orders(2)/SampleModel.Order" />
            <link
         rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(2)/SampleModel.Order/Customer" />
            <link
         rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                type="application/xml" title="Customer"
                href="Orders(2)/SampleModel.Order/$links/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/SampleModel.Order/OrderLines" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml" title="OrderLines"
                href="Orders(2)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
              <m:properties>
                 <d:OrderID m:type="Edm.Int32">2</d:OrderID>
                 <d:ShippedDate m:type="Edm.DateTime">1997-10-03T00:00:00</d:ShippedDate>
              </m:properties>
            </content>
          </entry>
</feed>
```

4.2.1.2 Retrieve a Collection of Entities by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a collection of Order entities that are related to a specific Customer entity from a data service.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 3.0

{"d": [
         {
            "__metadata": { "uri": "Orders(1)",
                            "type": "SampleModel.Order",
                            "properties" : {
                              "Customer" : {
                                "associationuri" :
"Orders(1)/SampleModel.Order/$links/Customer",
                              },
                              "OrderLines" : {
                                "associationuri" :
"Orders(1)/SampleModel.Order/$links/OrderLines",
                              }
                            }

                          },
            "OrderID": 1,
            "ShippedDate": "\/Date(872467200000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(1)/SampleModel.Order/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(1)/SampleModel.Order/OrderLines"}
}
         },
         {
            "__metadata": { "uri": "Orders(2)",
                            "type": "SampleModel.Order",
                            "properties" : {
                              "Customer" : {
                                "associationuri" :
"Orders(2)/SampleModel.Order/$links/Customer",
                              },
                              "OrderLines" : {
                                "associationuri" :
"Orders(2)/SampleModel.Order/$links/OrderLines",
                              }
                            }

                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/SampleModel.Order/Customer"} }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/SampleModel.Order/OrderLines"}
}

         }
]}
```

4.2.1.3 Retrieve a Partial Collection of Entities by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a collection of Order entities that are related to a specific Customer entity from a data service. This example assumes that the server has a limit in place to return, at the most, two Order entities at a time and that there exist more than two Order entities associated with the Customer entity that is identified by the key value "ALFKI".

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/atom+xml
```

```
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=feed
Content-Length: nnn
DataServiceVersion: 3.0

<feed xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
        <title type="text">Orders</title>
        <id>http://host/service.svc/Customers('ALFKI')/Orders</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Orders" href="Customers('ALFKI')/SampleModel.Customer/Orders"
/>
        <entry>
          <category term="SampleModel.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Orders(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
          <name />
          </author>
          <link rel="edit" title="Orders" href="Orders(1)/SampleModel.Order " />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(1)/SampleModel.Order /Customer" />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                type="application/ xml " title="Customer"
                href="Orders(1)/SampleModel.Order /$links/Customer" />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/SampleModel.Order /OrderLines" />
          <link
      rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml " title="OrderLines"
                href="Orders(1)/SampleModel.Order /$links/OrderLines" />

          <content type="application/xml">
            <d:OrderID m:type="Edm.Int32">1</d:OrderID>
            <d:ShippedDate m:type="Edm.DateTime">1997-08-25T00:00:00</d:ShippedDate>
          </content>
        </entry>
        <entry>
          <category term="SampleModel.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Orders(2)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Orders" href="Orders(2)/SampleModel.Order " />
          <link
       rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(2)/SampleModel.Order /Customer" />
          <link
       rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
```

```
                type="application/ xml" title="Customer"
                href="Orders(2)/SampleModel.Order /$links/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/SampleModel.Order /OrderLines" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml " title="OrderLines"
                href="Orders(2)/SampleModel.Order /$links/OrderLines" />
            <content type="application/xml">
              <d:OrderID m:type="Edm.Int32">2</d:OrderID>
              <d:ShippedDate m:type="Edm.DateTime">1997-10-03T00:00:00</d:ShippedDate>
            </content>
          </entry>
          <link rel="next"
                href="http://host/service.svc/Customers('ALFKI')/Orders?$skiptoken=2" />
</feed>
```

4.2.1.4 Retrieve a Partial Collection of Entities by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve, from a data service, a collection of Order entities that are related to a specific Customer entity.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 3.0

{"d": { "results":[
          {
             "__metadata": { "uri": "Orders(1)",
                             "type": "SampleModel.Order"
                             "properties" : {
                               "Customer" : {
                                 "associationuri" :
"Orders(1)/SampleModel.Order/$links/Customer"
                               },
                               "OrderLines" : {
                                 "associationuri" :
"Orders(1)/SampleModel.Order/$links/OrderLines"
                               }}
                       },
             "OrderID": 1,
             "ShippedDate": "\/Date(872467200000)\/",
             "Customer":   { "__deferred": { "uri": "Orders(1)/SampleModel.Order/Customer" } }
             "OrderLines": { "__deferred": { "uri": "Orders(1)/SampleModel.Order/OrderLines" }
}
          },
          {
             "__metadata": { "uri": "Orders(2)",
                             "type": "SampleModel.Order"
```

```
                              "properties" : {
                               "Customer" : {
                                 "associationuri" :
"Orders(2)/SampleModel.Order/$links/Customer"
                               },
                               "OrderLines" : {
                                 "associationuri" :
"Orders(2)/SampleModel.Order/$links/OrderLines"
                               }}
                         },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/SampleModel.Order/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/SampleModel.Order /OrderLines"
} }
         }],
         "__next":
                "http://host/service.svc/Customers('ALFKI')/Orders?$skiptoken=3"
      }
   }
```

4.2.1.5 Retrieve a Collection of Entities with an Inline Count by Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to retrieve a collection of Order entities related to a specific Customer entity and include the count of all orders associated with the Customer entity in the response from the data service. This example is supported in the OData 2.0 and OData 3.0 protocols.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders?$inlinecount=allpages HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 2.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=feed
Content-Length: nnn
DataServiceVersion: 3.0

<feed  xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
        <title type="text">Orders</title>
        <id>http://host/service.svc/Customers('ALFKI')/Orders</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Orders" href="Customers('ALFKI')/SampleModel.Customer/Orders"
/>
        <m:count>2<m:count>
        <entry>
          <category term="SampleModel.Order"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Orders(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
```

```
            <link rel="edit" title="Orders" href="Orders(1)/SampleModel.Order" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                  type="application/atom+xml;type=entry" title="Customer"
                  href="Orders(1)/SampleModel.Order/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                  type="application/xml " title="Customer"
                  href="Orders(1)/SampleModel.Order/$links/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                  type="application/atom+xml;type=feed" title="OrderLines"
                  href="Orders(1)/SampleModel.Order/OrderLines" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                  type="application/ xml " title="OrderLines"
                  href="Orders(1)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
              <d:OrderID m:type="Edm.Int32">1</d:OrderID>
              <d:ShippedDate m:type="Edm.DateTime">1997-08-25T00:00:00</d:ShippedDate>
            </content>
          </entry>
          <entry>
            <category term="SampleModel.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
            <id>http://host/service.svc/Orders(2)</id>
            <title type="text" />
            <updated>2008-03-30T21:52:45Z</updated>
            <author>
              <name />
            </author>
            <link rel="edit" title="Orders" href="Orders(2)/SampleModel.Order " />
            <link
         rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                  type="application/atom+xml;type=entry" title="Customer"
                  href="Orders(2)/SampleModel.Order/Customer" />
            <link
         rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                  type="application/ xml " title="Customer"
                  href="Orders(2)/SampleModel.Order/$links/Customer" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                  type="application/atom+xml;type=feed" title="OrderLines"
                  href="Orders(2)/SampleModel.Order/OrderLines" />
            <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                  type="application/ xml " title="OrderLines"
                  href="Orders(2)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
              <d:OrderID m:type="Edm.Int32">2</d:OrderID>
              <d:ShippedDate m:type="Edm.DateTime">1997-10-03T00:00:00</d:ShippedDate>
            </content>
          </entry>
</feed>
```

4.2.1.6 Retrieve a Collection of Entities with an Inline Count by Using the Verbose JSON Format

The following example illustrates the exchange of messages required for a client to retrieve a collection of Order entities related to a specific Customer entity and a count of all orders associated with the Customer entity from a data service. This example is supported only in the OData 2.0 and OData 3.0 protocols.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders?$inlinecount=allpages HTTP/1.1
```

```
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 2.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 3.0

{"d": [
         {
            "__count": "2"
         },
         {
            "__metadata": { "uri": "Orders(1)",
                            "type": "SampleModel.Order",
                            "properties" : {
                              "Customer" : {
                                "associationuri" :
"Orders(1)/SampleModel.Order/$links/Customer"
                              },
                              "OrderLines" : {
                                "associationuri" :
"Orders(1)/SampleModel.Order/$links/OrderLines"
                              }}
                          },
            "OrderID": 1,
            "ShippedDate": "\/Date(872467200000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(1)/SampleModel.Order/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(1)/SampleModel.Order/OrderLines" }
}
         },
         {
            "__metadata": { "uri": "Orders(2)",
                            "type": "SampleModel.Order",
                            "properties" : {
                              "Customer" : {
                                "associationuri" :
"Orders(2)/SampleModel.Order/$links/Customer"
                              },
                              "OrderLines" : {
                                "associationuri" :
"Orders(2)/SampleModel.Order/$links/OrderLines"
                              }}
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/SampleModel.Order/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/SampleModel.Order/OrderLines" }
}

         }
]}
```

4.2.1.7 Retrieve a Collection of Entities with Named Resource Streams by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a collection of Photo entities, each with two named resource streams (Thumbnail and PrintReady), by using the AtomPub format. This example is supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Photos HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 3.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=feed
Content-Length: nnn
DataServiceVersion: 3.0

<feed  xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
        <title type="text">Photos</title>
        <id>http://host/service.svc/Photos</id>
        <updated>2008-03-30T21:52:46Z</updated>
        <link rel="self" title="Photos" href="Photos/SampleModel.Photo" />
        <entry>
          <category term="SampleModel.Photo"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
          <id>http://host/service.svc/Photos(1)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Photos" href="Photos(1)/SampleModel.Photo " />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/Thumbnail"

            type="image/png"
            title="Thumbnail"
            href="Photos(1)/SampleModel.Photo/Thumbnail " />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/Thumbnail"
            type="image/png"
            title="Thumbnail"
            href="Photos(1)/SampleModel.Photo/Thumbnail " />

          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/PrintReady"
            type="image/png"
            title="PrintReady"
            href="Photos(1)/SampleModel.Photo/PrintReady " />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/PrintReady"
            type="image/png"
            title="PrintReady"
            href="Photos(1)/SampleModel.Photo/PrintReady " />

          <content type="application/xml">
           <m:properties>
            <d:ID m:type="Edm.Int32">1</d:ID>
            <d:Name m:type="Edm.String">Mount Fuji</d:Name>
           </m:properties>
          </content>
        </entry>
        <entry>
          <category term="SampleModel.Photo"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
```

```
          <id>http://host/service.svc/Photos(2)</id>
          <title type="text" />
          <updated>2008-03-30T21:52:45Z</updated>
          <author>
            <name />
          </author>
          <link rel="edit" title="Photos" href="Photos(2)/SampleModel.Photo" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/Thumbnail"
            type="image/png"
            title="Thumbnail"
            href="Photos(2)/SampleModel.Photo/Thumbnail " />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/Thumbnail"
            type="image/png"
            title="Thumbnail"
            href="Photos(2)/SampleModel.Photo/Thumbnail "
            m:etag="######" />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/mediaresource/PrintReady"
            type="image/png"
            title="PrintReady"
            href="Photos(2)/SampleModel.Photo/PrintReady " />
          <link
rel="http://schemas.microsoft.com/ado/2007/08/dataservices/edit-media/PrintReady"
            type="image/png"
            title="PrintReady"
            href="Photos(2)/SampleModel.Photo/PrintReady "
            m:etag="######" />

          <content type="application/xml">
            <m:properties>
             <d:ID m:type="Edm.Int32">2</d:ID>
             <d:Name m:type="Edm.String">Mount Rainier</d:Name>
           </m:properties>
          </content>
        </entry>
</feed>
```

4.2.1.8 Retrieve a Collection of Entities with Named Resource Streams by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a collection of Photo entities that each contain two named resource streams (Thumbnail and PrintReady) by using the Verbose JSON Format. This example is supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Photos HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 3.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose; type=feed
Content-Length: nnn
DataServiceVersion: 3.0

{"d": [
        {
```

```
            "__metadata": { "uri": "Photos(1)",
                            "type": "SampleModel.Photo",
                          },
            "ID": 1,
            "Name": "Mount Fuji",
            "Thumbnail": {
                "__mediaresource": {
                    "edit_media": "Photos(1)/SampleModel.Photo/Thumbnail ",
                    "media_src": "Photos(1)/SampleModel.Photo/Thumbnail ",
                    "content-type": "img/jpeg",
                    "media_etag": "####"
                }
            },
            "PrintReady": {
                "__mediaresource": {
                    "edit_media": "Photos(1)/SampleModel.Photo/PrintReady ",
                    "media_src": "Photos(1)/SampleModel.Photo/PrintReady ",
                    "content-type": "img/png",
                    "media_etag": "####"
                }
            },
        },
        {
            "__metadata": { "uri": "Photos(2)",
                            "type": "SampleModel.Photo",
                          },
            "ID": 2,
            "Name": "Mount Rainier",
            "Thumbnail": {
                "__mediaresource": {
                    "edit_media": "Photos(2)/SampleModel.Photo/Thumbnail ",
                    "media_src": "Photos(2)/SampleModel.Photo/Thumbnail ",
                    "content-type": "img/jpeg",
                    "media_etag": "####"
                }
            },
            "PrintReady": {
                "__mediaresource": {
                    "edit_media": "Photos(2)/SampleModel.Photo/PrintReady ",
                    "media_src": "Photos(2)/SampleModel.Photo/PrintReady ",
                    "content-type": "img/png",
                    "media_etag": "####"
                }
            }
        }
]}
```

4.2.2 Retrieve a Single Entity by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity with an EntityKey value equal to "ALFKI" by using the AtomPub format.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
```

```
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 3.0

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom" m:etag="W/"X'000000000000FA01'""">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customer" href="Customers('ALFKI')/SampleModel.Customer " />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/Orders" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Orders"
        type="application/ xml "
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/$links/Orders" />
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ALFKI</d:CustomerID>
    <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
    <d:Address>
      <d:Street>57 Contoso St</d:Street>
      <d:City>Seattle</d:City>
      <d:Location m:type="Edm.GeographyPoint"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
    </d:Address>
    <d:Version>AAAAAAAA+gE=</d:Version>
   </m:properties>
  </content>
</entry>
```

4.2.2.1 Retrieve a Single Entity with a Mapped Property by Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to retrieve an Employee entity with an EntityKey value equal to 1 by using the AtomPub format. The EmployeeName property and the City property of the Address complex-type property on the Employee EntityType have been mapped to another element by using the customizable feeds attributes that are included in the CSDL file. One of the property mappings on the Employee **EntityType** has the **FC_KeepInContent** attribute set to "false" and, as a result, the following exchange of messages is supported only in OData 2.0 and OData 3.0.

Request:

```
GET /service.svc/Employees(1) HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 2.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 2.0

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns="http://www.w3.org/2005/Atom" m:etag="W/"X'000000000000FA01'""">
  <category term="SampleModel.Employee"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Employees(1)</id>
  <title type="text"> Nancy Davolio</title>
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Employees" href="Employees(1)/SampleModel.Employee" />
  <content type="application/xml">
   <m:properties>
    <d:EmployeeID>ALFKI</d:EmployeeID>
    <d:Address>
      <d:Street>507 - 20th Ave. E. Apt. 2A<d:Street>
      <d:City>Seattle</d:City>
      <d:Location m:type="Edm.GeographyPoint"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
    </d:Address>
    <d:Version>BBBBBBBB+gE=</d:Version>
   </m:properties>
  </content>
  <emp:Location xmlns:emp="http://www.microsoft.com">Seattle</emp:Location>
</entry>
```

4.2.3 Retrieve a Single Entity by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity with an EntityKey value that is equal to "ALFKI" by using the Verbose JSON format.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 3.0

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
```

```
                  "etag": "W/\"X\'000000000000FA01\'\""
                   "properties" : {
                             "Orders" : {
                               "associationuri" :
"Customers(\'ALFKI\')/SampleModel.Customer/$links/Orders ",
                             },
                 },
   "CustomerID": "ALFKI",
   "CompanyName": " Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle",
     "Location": {
       "crs": {
         "type": "name",
         "properties": { "name": "EPSG:4326" }
       },
       "type": "Point", "coordinates": [-127.9324, 49.2345]
     }
   },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/SampleModel.Customer/Orders" } }
 }
}
```

4.2.4 Retrieve a Single Entity and Its Directly Related Entities by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity (with EntityKey value equal to "ALFKI") and its associated Order **EntityType** instances. This example uses the AtomPub format for all messages.

Request:

```
GET /service.svc/Customers('ALFKI')?$expand=Orders HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
DataServiceVersion: 3.0

<?xml version="1.0" encoding="utf-8"?>
<entry  xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-03-30T21:32:23Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')/SampleModel.Customer" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
```

```
        href="Customers('ALFKI')/SampleModel.Customer/Orders">
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Orders"
        type="application/xml"
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/$links/Orders">
      <m:inline>
        <feed>
          <title type="text">Orders</title>
          <id>http://host/service.svc/Customers('ALFKI')/Orders</id>
          <updated>2008-03-30T21:52:46Z</updated>
          <link rel="self" title="Orders" href="Customers('ALFKI')/SampleModel.Customer/Orders"
/>
          <entry>
            <category term="SampleModel.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
            <id>http://host/service.svc/Orders(1)</id>
            <title type="text" />
            <updated>2008-03-30T21:52:45Z</updated>
            <author>
              <name />
            </author>
            <link rel="edit" title="Orders" href="Orders(1)/SampleModel.Order" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(1)/SampleModel.Order/Customer" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                type="application/xml" title="Customer"
                href="Orders(1)/SampleModel.Order/$links/Customer" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/SampleModel.Order/OrderLines" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml" title="OrderLines"
                href="Orders(1)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
              <m:properties>
                <d:OrderID m:type="Edm.Int32">1</d:OrderID>
                <d:ShippedDate m:type="Edm.DateTime">1997-08-25T00:00:00</d:ShippedDate>
              </m:properties>
            </content>
          </entry>
          <entry>
            <category term="SampleModel.Order"
              scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
            <id>http://host/service.svc/Orders(2)</id>
            <title type="text" />
            <updated>2008-03-30T21:52:45Z</updated>
            <author>
              <name />
            </author>
            <link rel="edit" title="Orders" href="Orders(2)/SampleModel.Order" />
            <link
           rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer"
                type="application/atom+xml;type=entry" title="Customer"
                href="Orders(2)/SampleModel.Order/Customer" />
            <link
           rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Customer"
                type="application/xml" title="Customer"
                href="Orders(2)/SampleModel.Order/$links/Customer" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/SampleModel.Order/OrderLines" />
            <link
          rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/OrderLines"
                type="application/xml" title="OrderLines"
```

```
                href="Orders(2)/SampleModel.Order/$links/OrderLines" />
            <content type="application/xml">
              <m:properties>
                <d:OrderID m:type="Edm.Int32">2</d:OrderID>
                <d:ShippedDate m:type="Edm.DateTime">1997-10-03T00:00:00</d:ShippedDate>
              </m:properties>
            </content>
          </entry>
        </feed>
      </m:inline>
    </link>
  <content type="application/xml">
    <d:CustomerID>ALFKI</d:CustomerID>
    <d:CompanyName>Alfreds Futterkiste</d:CompanyName>
    <d:Address>
      <d:Street>57 Contoso St</d:Street>
      <d:City>Seattle</d:City>
      <d:Location m:type="Edm.Point"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
    </d:Address>
    <d:Version>AAAAAAAA+gE=</d:Version>
  </content>
</entry>
```

4.2.5 Retrieve a Single Entity and Its Directly Related Entities by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity (with EntityKey value equal to "ALFKI") and its associated Order **EntityType** instances. This example uses the Verbose JSON Format for all messages.

Request:

```
GET /service.svc/Customers('ALFKI')?$expand=Orders HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 3.0

{"d": {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                   "properties" : {
                              "Orders" : {
                                "associationuri" :
"Customer(\'ALFKI\')/SampleModel.Customer/$links/Orders"
}}
                  },
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle",
     "Location": {
       "crs": {
         "type": "name",
         "properties": { "name": "EPSG:4326" }
```

```
        },
        "type": "Point", "coordinates": [-127.9324, 49.2345]
      }
    },
    "Version": "AAAAAAAA+gE=",
    "Orders": [
          {
             "__metadata": { "uri": "Orders(1)",
                             "type": "SampleModel.Order"
                             "properties" : {
                               "Customer" : {
                                 "associationuri" :
"Orders(1)/SampleModel.Order/$links/Customer",
                               },
                               "OrderLines" : {
                                 "associationuri" :
"Orders(1)/SampleModel.Order/$links/OrderLines",
                               }}
                           },
             "OrderID": 1,
             "ShippedDate": "\/Date(872467200000)\/",
             "Customer":   { "__deferred": { "uri": "Orders(1)/SampleModel.Order/Customer" } }
             "OrderLines": { "__deferred": { "uri": "Orders(1)/SampleModel.Order/OrderLines" }
}
          },
          {
             "__metadata": { "uri": "Orders(2)",
                             "type": "SampleModel.Order"
                              "properties" : {
                               "Customer" : {
                                 "associationuri" :
"Orders(2)/SampleModel.Order/$links/Customer",
                               },
                               "OrderLines" : {
                                 "associationuri" :
"Orders(2)/SampleModel.Order/$links/OrderLines",
                               }}
                         },
             "OrderID": 2,
             "ShippedDate": "\/Date(875836800000)\/",
             "Customer":   { "__deferred": { "uri": "Orders(2)/SampleModel.Order/Customer" } }
             "OrderLines": { "__deferred": { "uri": "Orders(2)/SampleModel.Order/OrderLines" }
}

          }
    ]
} }
```

4.2.6 Retrieve a Data Service's Metadata Document (CSDL)

The following example illustrates the exchange of messages that is required for a client to obtain a description document of a data service, as specified in RetrieveServiceMetadata Request (section 2.2.7.2.7).

The following data service metadata document contains customizable feed property mappings that are supported only in OData 2.0 and OData 3.0.

The data service metadata document also contains named resource streams that are supported only in OData 3.0.

Request:

```
GET /service.svc/$metadata HTTP/1.1
Host: host
Accept: application/xml
```

```
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/xml
Content-Length: nnn
DataServiceVersion: 3.0

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<edmx:Edmx Version="1.0"
       xmlns:edmx="http://schemas.microsoft.com/ado/2010/02/edmx"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<edmx:DataServices m:DataServiceVersion="3.0">
   <Schema Namespace="SampleModel"
     xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityContainer Name="SampleEntities"
                       m:IsDefaultEntityContainer="true">
        <EntitySet Name="Customers" EntityType="SampleModel.Customer" />
        <EntitySet Name="Employees" EntityType="SampleModel.Employee" />
        <EntitySet Name="Orders" EntityType="SampleModel.Order" />
        <EntitySet Name="OrderLines" EntityType="SampleModel.OrderLine" />
        <EntitySet Name="Photos" EntityType="SampleModel.Photo" />
        <AssociationSet Name="Orders_Customers"
             Association="SampleModel.Orders_Customers">
            <End Role="Customers" EntitySet="Customers" />
            <End Role="Orders" EntitySet="Orders" />
        </AssociationSet>
        <AssociationSet Name="OrderLines_Orders"
             Association="SampleModel.OrderLines_Orders">
            <End Role="OrderLine" EntitySet="OrderLines" />
            <End Role="Order" EntitySet="Orders" />
        </AssociationSet>
        <FunctionImport Name="CustomersByCity"
                        EntitySet="Customers"
                        ReturnType="Collection(SampleModel.Customer)"
                        m:HttpMethod="GET">
           <Parameter Name="city" Type="Edm.String" Mode="In" />
        </FunctionImport>
        <FunctionImport Name="GetRelatedCustomers"
                        EntitySet="Customers"
                        IsBindable="true"
                        ReturnType="Collection(SampleModel.Customer)"
                        IsSideEffecting="false">
           <Parameter Name="company" Type="SampleModel.Company" Mode="In" />
        </FunctionImport>
        <FunctionImport Name="CreateOrder"
                        EntitySet="Orders"
                        IsBindable="true"
                        IsSideEffecting="true"
                        m:IsAlwaysBindable="true">
           <Parameter Name="customer" Type="SampleModel.Customer" Mode="In" />
           <Parameter Name="quantity" Type="Edm.Int32" Mode="In" />
           <Parameter Name="discountCode" Type="Edm.String" Mode="In" />
        </FunctionImport>
      </EntityContainer>
      <EntityType Name="Order">
        <Key>
          <PropertyRef Name="OrderID" />
        </Key>
        <Property Name="OrderID" Type="Edm.Int32" Nullable="false" />
        <Property Name="ShippedDate" Type="Edm.DateTime" Nullable="true"
           DateTimeKind="Unspecified" PreserveSeconds="true" />
        <NavigationProperty Name="Customer"
           Relationship="SampleModel.Orders_Customers"
           FromRole="Order" ToRole="Customer" />
```

```
  <NavigationProperty Name="OrderLines"
     Relationship="SampleModel.OrderLines_Orders"
     FromRole="Order" ToRole="OrderLine" />
</EntityType>
<EntityType Name="OrderLine">
  <Key>
    <PropertyRef Name="OrderLineID" />
  </Key>
  <Property Name="OrderLineID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Quantity" Type="Edm.Int32" Nullable="false" />
  <Property Name="UnitPrice" Type="Edm.Decimal" Nullable="false" />
  <NavigationProperty Name="Order"
     Relationship="SampleModel.OrderLines_Orders"
     FromRole="OrderLine" ToRole="Order" />
</EntityType>
<EntityType Name="Customer">
  <Key>
    <PropertyRef Name="CustomerID" />
  </Key>
  <Property Name="CustomerID" Type="Edm.String" Nullable="false"
     MaxLength="5" Unicode="true" FixedLength="true" />
  <Property Name="CompanyName" Type="Edm.String" Nullable="false"
     MaxLength="40" Unicode="true" FixedLength="false" />
  <Property Name="Address" Type="Sample.CAddress" Nullable="true" />
  <Property Name="Version" Type="Edm.Binary" Nullable="true" MaxLength="8"
     FixedLength="true" ConcurrencyMode="Fixed" />
  <NavigationProperty Name="Orders"
     Relationship="SampleModel.Orders_Customers"
     FromRole="Customer" ToRole="Order" />
</EntityType>
<EntityType Name="Employee" m:FC_KeepInContent="true"
     m:FC_TargetPath="Location" m:FC_SourcePath="Address/City"
     m:FC_NsUri="http://www.microsoft.com" m:FC_NsPrefix="emp">
  <Key>
    <PropertyRef Name="EmployeeID" />
  </Key>
  <Property Name="EmployeeID" Type="Edm.String" Nullable="false"
     MaxLength="5" Unicode="true" FixedLength="true" />
  <Property Name="EmployeeName" Type="Edm.String" Nullable="false"
     MaxLength="40" Unicode="true" FixedLength="false"
     m:FC_KeepInContent="false"
     m:FC_TargetPath="SyndicationTitle"/>
  <Property Name="Address" Type="Sample.EAddress" Nullable="true" />
  <Property Name="Version" Type="Edm.Binary" Nullable="true" MaxLength="8"
     FixedLength="true" ConcurrencyMode="Fixed" />
</EntityType>
<EntityType Name="Company">
  <Key>
    <PropertyRef Name="CompanyID" />
  </Key>
  <Property Name="CompanyID" Type="Edm.String" Nullable="false"
     MaxLength="5" Unicode="true" FixedLength="true" />
  <Property Name="CompanySize" Type="Edm.String" Nullable="true"/>
</EntityType>

<EntityType Name="Photo" m:HasStream="true">
  <Key>
    <PropertyRef Name="ID" />
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Name" Type="Edm.String" Nullable="true" />
  <Property Name="Thumbnail" Type="Edm.Stream" />
  <Property Name="PrintReady" Type="Edm.Stream" />
</EntityType>

<ComplexType Name="CAddress">
  <Property Name="Street" Type="Edm.String" Unicode="true" />
  <Property Name="City" Type="Edm.String" Unicode="true"/>
</ComplexType>
<ComplexType Name="EAddress">
```

```
        <Property Name="Street" Type="Edm.String" Unicode="true" />
        <Property Name="City" Type="Edm.String" Unicode="true"/>
      </ComplexType>
      <Association Name="Orders_Customers">
        <End Role="Customer" Type="SampleModel.Customer"
             Multiplicity="0..1" />
        <End Role="Order" Type="SampleModel.Order" Multiplicity="*" />
      </Association>
      <Association Name="OrderLines_Orders">
        <End Role="Order" Type="SampleModel.OrderLine"
             Multiplicity="*" />
        <End Role="OrderLine" Type="SampleModel.Order" Multiplicity="0..1" />
      </Association>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

4.2.7 Retrieve the Count of a Collection of Entities

The following example illustrates the exchange of messages required for a client to retrieve a count of all Customer entities. This example is supported only in the OData 2.0 and OData 3.0 protocols.

Request:

```
GET /service.svc/Customers/$count HTTP/1.1
Host: host
Accept: text/plain
DataServiceVersion: 2.0
MaxDataServiceVersion: 2.0
Content-Type: text/plain
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/plain
DataServiceVersion: 2.0;
Date: Fri, 01 May 2009 21:41:31 GMT
Content-Length: 2

91
```

4.2.8 Retrieve a Single Entity Exposing an Action by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Company entity with an **EntityKey** value equal to 1 using the AtomPub format. The server indicates that the returned Company allows an action (Audit) to be invoked against it. Actions are supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Company(1) HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Thu, 02 Sep 2010 03:40:29 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
DataServiceVersion: 3.0
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://host/service.svc/Company(1)</id>
  <title type="text"/>
  <updated> 2010-08-28T01:29:11Z </updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Company" href="Company(1)/SampleModel.Company" />
  <m:action rel="SampleEntities.Audit"
            title="Audit"
            target="Company(1)/SampleEntities.Audit" />
  <category term="SampleModel.Company"
           scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <category term="Large" scheme="http://company.com/CompanySize" />
  <content type="application/xml">
   <m:properties>
    <d:CompanyID>1</d:CompanyID>
   </m:properties>
  </content>
</entry>
```

4.2.9 Retrieve a Single Entity Exposing an Action by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity with **EntityKey** value equal to "ALFKI" by using the Verbose JSON format. The server indicates that the returned Customer allows an action (CreateOrder) to be invoked against it. Actions are supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 3.0

{"d":
 {
   "__metadata": {
       "uri": "Customers(\'ALFKI\')",
       "type": "SampleModel.Customer",
       "etag": "W/\"X\'000000000000FA01\'\""
       "properties" : {
           "Orders" : {
              "associationuri" : "Customers(\'ALFKI\')/SampleModel.Customer/$links/Orders",
```

```
            }
        },
        "actions" : {
            "SampleEntities.CreateOrder" : [{
                "title" : "Create Order",
                "target" : "Customers(\'ALFKI\')/SampleEntities.CreateOrder"
            }]
        }
    },
    "CustomerID": "ALFKI",
    "CompanyName": " Alfreds Futterkiste",
    "Address": { "Street": "57 Contoso St", "City": "Seattle",
      "Location": {
        "crs": {
          "type": "name",
          "properties": { "name": "EPSG:4326" }
        },
        "type": "Point", "coordinates": [-127.9324, 49.2345]
      }
    },
    "Version": "AAAAAAAA+gE=",
    "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/SampleModel.Customer/Orders" } }
  }
}
```

4.2.10 Retrieve a Single Entity Exposing a Function by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Company entity with an **EntityKey** value equal to 2 by using the AtomPub format. The server indicates that the returned Company supports a GetRelatedCustomers function. Functions are supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Company(2) HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Thu, 02 Sep 2010 03:40:29 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
DataServiceVersion: 3.0
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <id>http://host/service.svc/Company(2)</id>
  <title type="text"/>
  <updated> 2010-08-28T01:29:11Z </updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Company" href="Company(2)/SampleModel.Company" />
  <m:function rel="SampleEntities.GetRelatedCustomers"
            title="Get Related Customers"
            target="Company(2)/SampleEntities.GetRelatedCustomers" />
  <category term="SampleModel.Company"
           scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
```

```
  <category term="Large" scheme="http://company.com/CompanySize" />
  <content type="application/xml">
   <m:properties>
    <d:CompanyID>2</d:CompanyID>
   </m:properties>
  </content>
</entry>
```

4.2.11 Retrieve a Single Entity Exposing a Function by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to retrieve a Customer entity with **EntityKey** value equal to "ALFKI" by using the Verbose JSON format. The server indicates that the returned Customer allows a function (TopTenOrders) to be invoked against it. Functions are supported only in the OData 3.0 protocol.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 3.0

{"d":
 {
   "__metadata": {
       "uri": "Customers(\'ALFKI\')",
        "type": "SampleModel.Customer",
       "etag": "W/\"X\'000000000000FA01\'\""
        "properties" : {
           "Orders" : {
              "associationuri" : "Customers(\'ALFKI\')/SampleModel.Customer/$links/Orders",
           }
       },
       "functions" : {
           "SampleEntities.TopTenOrders" : [{
               "title" : "Get Top Ten Orders for this Customer",
               "target" : "Customers(\'ALFKI\')/SampleEntities.TopTenOrders"
           }]
       }
   },
   "CustomerID": "ALFKI",
   "CompanyName": " Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle",
     "Location": {
       "crs": {
         "type": "name",
         "properties": { "name": "EPSG:4326" }
       },
       "type": "Point", "coordinates": [-127.9324, 49.2345]
     }
   },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/SampleModel.Customer/Orders" } }
 }
```

```
}
```

4.3 Update an Existing Entity

4.3.1 Replace-Based Update by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to update an existing entity in a data service by using the AtomPub format and replace-based update semantics.

HTTP Request:

```
PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/atom+xml
If-Match: W/"X'000000000000FA01'"
Accept: application/atom+xml
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
Prefer: return-content
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ALFKI</d:CustomerID>
    <d:CompanyName>Updated Company Name</d:CompanyName>
    <d:Address>
      <d:Street>Updated Street</d:Street>
    </d:Address>
   </m:properties>
  </content>
</entry>
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 3.0
Preference-Applied: return-content

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-04-30T17:17:11Z</updated>
  <author>
    <name />
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')/SampleModel.Customer" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
```

```
        href="Customers('ALFKI')/SampleModel.Customer/Orders" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Orders"
        type="application/xml "
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/$links/Orders" />
  <content type="application/xml">
   <m:properties>
    <d:CustomerID>ALFKI</d:CustomerID>
    <d:CompanyName>Updated Company Name</d:CompanyName>
    <d:Address>
      <d:Street>Updated Street</d:Street>
      <d:City></d:City>
      <d:Location m:type="Edm.GeographyPoint"></d:Location>
    </d:Address>
    <d:Version>AAAAAAAA+gF=</d:Version>
   </m:properties>
  </content>
</entry>
```

4.3.2 Replace-Based Update by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to update an existing entity in a data service by using the Verbose JSON format and replace-based update semantics.

HTTP Request:

```
PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
If-Match: W/"X'000000000000FA01'"
Accept: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
Prefer: return-content


{"d":
 {
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street" },
 }
}
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 3.0
Preference-Applied: return-content

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA02\'\""
                   "properties" : {
                             "Orders" : {
```

```
                                "associationuri" : "
Customers(\'ALFKI\')/SampleModel.Customer/$links/Orders"
                               }
                    }
                },
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street", "City": "", "Location"="NULL" },
   "Version": "AAAAAAAA+gF=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
}
```

4.3.3 Merge-Based Update by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to update an existing entity in a data service by using the AtomPub format and merge-based update semantics.

HTTP Request:

```
PATCH /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/atom+xml
If-Match: W/"X'000000000000FA01'"
Accept: application/atom+xml
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
Prefer: return-content

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns="http://www.w3.org/2005/Atom">
  <content type="application/xml">
   <m:properties>
    <d:CompanyName>Updated Company Name</d:CompanyName>
   </m:properties>
  </content>
</entry>
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 3.0
Preference-Applied: return-content

<?xml version="1.0" encoding="utf-8"?>
<entry xml:base="http://host/service.svc/"
       xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:gml=http://schemas.opengis.net/gml/3.1.1/profiles/gmlsfProfile/1.0.0/gmlsf.xsd
       xmlns="http://www.w3.org/2005/Atom">
  <category term="SampleModel.Customer"
            scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://host/service.svc/Customers('ALFKI')</id>
  <title type="text" />
  <updated>2008-04-30T17:17:11Z</updated>
  <author>
    <name />
```

```
  </author>
  <link rel="edit" title="Customers" href="Customers('ALFKI')/SampleModel.Customer" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/Orders" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/relatedlinks/Orders"
        type="application/xml"
        title="Orders"
        href="Customers('ALFKI')/SampleModel.Customer/$links//Orders" />
 <content type="application/xml">
   <m:properties>
    <d:CustomerID>ALFKI</d:CustomerID>
    <d:CompanyName>Updated Company Name</d:CompanyName>
    <d:Address>
      <d:Street>57 Contoso St</d:Street>
      <d:City>Seattle</d:City>
      <d:Location m:type="Edm.GeographyPoint"><gml:Point srsName="4326">-127.345345
48.23423</gml:Point></d:Location>
    </d:Address>
    <d:Version>AAAAAAAA+gF=</d:Version>
   </m:properties>
  </content>
</entry>
```

4.3.4 Merge-Based Update by Using the Verbose JSON Format

The following example illustrates the exchange of messages that is required for a client to update an existing entity in a data service by using the Verbose JSON format and merge-based update semantics. The PATCH verb is supported only in the OData 3.0 protocol.

HTTP Request

```
PATCH /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
If-Match: W/"X'000000000000FA01'"
Accept: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 3.0
MaxDataServiceVersion: 3.0
Prefer: return-content

{"d":
 {
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street" }
 }
}
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/json;odata=verbose
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 3.0
Preference-Applied: return-content

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
```

```
                    "type": "SampleModel.Customer",
                    "etag": "W/\"X\'000000000000FA02\'\""
                    "properties" : {
                              "Orders" : {
                                "associationuri" :
                                              "
Customers(\'ALFKI\')/SampleModel.Customer/$links/Orders "
                                }
                     }
            },
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street", "City": "Seattle",
     "Location": {
       "crs": {
         "type": "name",
         "properties": { "name": "EPSG:4326" }
       },
       "type": "Point", "coordinates": [-127.9324, 49.2345]
     }
   },
   "Version": "AAAAAAAA+gF=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
}
```

4.4 Update the Relationship Between Two Entities

4.4.1 Update a Relationship by Using the AtomPub Format

The following example illustrates the exchange of messages that is required for a client to update the association between the Order entity with EntityKey value 1 and its associated Customer. This example binds the Order 1 to **Customer** "ASDFG" by using the AtomPub format.

HTTP Request:

```
PUT /service.svc/Order(1)/SampleModel.Customer/$links/Customer HTTP/1.1
Host: host
Content-Type: application/atom+xml
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0

<uri
xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">http://host/service.svc/Custome
r('ASDFG')</uri>
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
DataServiceVersion: 1.0
```

4.4.2 Update a Relationship by Using the Verbose JSON Format

The following example illustrates the exchange of messages required for a client to update the association between the Order entity with EntityKey value 1 and its associated Customer. This example binds the Order 1 to Customer "ASDFG" by using the Verbose JSON format (section 2.2.6.3).

HTTP Request:

```
PUT /service.svc/Order(1)/SampleModel.Customer/$links/Customer HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0

{"uri": "http://host/service.svc/Customers('ASDFG')"}
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
DataServiceVersion: 1.0
```

4.4.3 Delete an Existing Entity

The following example illustrates the exchange of messages required for a client to delete an existing entity in a data service. This example shows the deletion of the **Customer** entity with the EntityKey value equal to "ALFKI".

HTTP Request:

```
DELETE /service.svc/Customers('ALFKI')/SampleModel.Customer HTTP/1.1
If-Match: W/"X'000000000000FA01'"
Host: host
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
DataServiceVersion: 1.0
```

4.5 Batch Requests

Detailed Batch request and response examples are provided in sections Example Batch Request (section 2.2.7.6.5) and Example Batch Response (section 2.2.7.6.8).

4.6 Working with Media Resources (BLOBs)

The examples that follow in this section illustrate the exchange of messages required for a client to create a Media Resource, update it, and then retrieve the associated Media Link Entry. This example shows the deletion of the Document entity with the EntityKey value equal to 300.

4.6.1 Insert a New Media Resource

HTTP Request:

```
POST /Documents/ HTTP/1.1
Host: host
Content-Type: application/rtf
Slug: Meeting Notes
```

```
Content-Length: ####
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0


...binary data for the rtf document...
```

HTTP Response:

```
HTTP/1.1 201 Created
Date: Fri, 11 Oct 2008 04:23:49 GMT
Content-Length: ###
Content-Type:  application/atom+xml;type=entry;charset="utf-8"
DataServiceVersion: 1.0
Location: http://host/service.svc/Documents(300)

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
   <title type="text"/>
    <id> http://host/service.svc/Documents(300)</id>
    <updated>2008-11-03T04:23:49Z</updated>
    <author><name></name></author>
    <category term="SampleModel.Document"
       scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
    <summary type="text" />
    <content type="application/rtf"
       src="http://host/service.svc/Documents(300)/$value"/>
    <m:properties>
       <d:DocumentID>300</d:DocumentID>
       <d:Author>Joe Smith</d:Author>
       <d:Title>Meeting Notes</d:Title>
    </m:properties>
    <link rel="edit-media"
          href="http://host/service.svc/Documents(300)/SampleModel.Document/$value" />
    <link rel="edit"
          href="http://host/service.svc/Documents(300)/SampleModel.Document" />
</entry>
```

4.6.2 Update a Media Resource

HTTP Request:

```
PUT /Documents(300)/SampleModel.Document HTTP/1.1
Host: host
Content-Type: application/rtf
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
Content-Length: ####

...binary data for the rtf document...
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 11 Oct 2008 04:23:49 GMT
DataServiceVersion: 1.0
```

4.6.3 Query an Existing Media Resource

HTTP Request:

```
GET /Documents(300)/SampleModel.Document/$value HTTP/1.1
Host: host
Accept: application/rtf
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Fri, 11 Oct 2008 04:23:49 GMT
Content-Length: ####
DataServiceVersion: 1.0
...binary data for the rtf document...
```

4.7 Working with Named Resource Streams Instances (BLOBs)

4.7.1 Retrieving a Named Resource Stream Instance

To retrieve a named resource stream instance, a client issues an HTTP GET request against the named resource stream self-link.

To acquire a named resource stream self-link, a client can issue a request as described in section 4.2.1.7 for AtomPub, in Stream Property in [ODataJSON4.0] section 9 for JSON, or in section 4.2.1.8 for Verbose JSON. And, a client can interpret the response according to the rules that are specified in section 2.2.6.2.10 for AtomPub, in Stream Property in [ODataJSON4.0] section 9 for JSON, or in section 2.2.6.3.14 for Verbose JSON.

A client sets the Accept header value to the ContentType of the named resource stream instance.

A clients sets both the DataServiceVersion and the MaxDataServiceVersion of header values to 2.0 or higher.

When the server responds, the response header contains the current DataServiceVersion.

HTTP Request:

```
GET /Photos(1)/SampleModel.Photo/Thumbnail HTTP/1.1
Host: host
Accept: image/png
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Fri, 11 Oct 2008 04:23:49 GMT
Content-Length: ####
DataServiceVersion: 1.0

...binary data for the png thumbnail ...
```

If the server returns an ETag for the named resource stream instance and the client opts to retrieve the stream only when the known ETag no longer represents the current version, the client includes the ETag with the request in the If-None-Match (section 2.2.5.6) header.

Servers that do support the If-None-Match header return the stream only when it has been modified because the version represented by the ETag specified in the If-None-Match header is no longer the current version.

4.7.2 Updating a Named Resource Stream Instance

To update the named resource stream instance, clients use a standard HTTP [RFC2616] PUT to the URI that is specified in the named resource stream instance's edit link.

The server indicates that updates to the named resource stream instance are subject to concurrency control checks by returning an ETag (section 2.2.5.4) for the named resource stream instance whenever a client retrieves the containing EntityType instance.

If an ETag is specified, the client sets the value of the If-Match header to the known ETag value.

HTTP Request:

```
PUT /Photos(1)/SampleModel.Photo/Thumbnail HTTP/1.1
Host: host
Content-Type: application/png
DataServiceVersion: 1.0
MaxDataServiceVersion: 3.0
If-Match: ...ETag...
Content-Length: ####

...binary data for the png thumbnail ...
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 11 Oct 2008 04:23:49 GMT
DataServiceVersion: 1.0
```

4.7.3 Unsupported Operations

4.7.3.1 Inserting a New Named Resource Stream Instance

There is no direct way to insert a named resource stream instance.

Instead, a named resource stream instance comes into existence when the owning **EntityType** instance is created.

Clients can attempt to retrieve a named resource stream instance (section 4.7.2) immediately after creating the containing **EntityType** instance.

Servers can choose to initialize the named resource stream to a nonempty value when the containing **EntityType** instance is created. However, it is likely that the named resource stream instance will be empty immediately after creating the **EntityType** instance.

4.7.3.2 Deleting a New Named Resource Stream Instance

There is no direct way to delete a named resource stream instance.

Instead, you can update the named resource stream instance by using its edit link so that it contains an empty stream. Or, you can delete the containing **EntityType** instance.

4.8 Invoking an Action

To invoke an action, a client sends an HTTP POST request to the URL that represents the action. Binding allows a client to send a POST request to a URL that represents the action with the binding parameter that is already provided.

In the following example, the binding parameter (called "Customer"; see section 6) to the CreateOrder action is the resource identified by the URL preceding the fully-qualified action name. The remaining parameters are specified in the body according to the rules specified in section 2.2.7.5.1.

Additionally, in the following request, the client optionally included an ETag for the Customer's ('ALFKI') entity. As a result, the server processes the Invoke Action request only if the ETag identifies the most up-to-date version of the entity.

HTTP Request:

```
POST /Customers('ALFKI')/SampleEntities.CreateOrder HTTP/1.1
Host: host
Content-Type: application/json;odata=verbose
DataServiceVersion: 3.0
MaxDataServiceVersion: 3.0
If-Match: ...ETag...
Content-Length: ####

{
   "quantity": 2,
   "discountCode": "BLACKFRIDAY"
}
```

HTTP Response:

```
HTTP/1.1 204 OK
Date: Fri, 11 Oct 2008 04:23:49 GMT
```

Notice in this example that because the CreateOrder action has no ReturnType, the response code is 204 (No Content).

4.9 Invoking a Function

To invoke a function, a client sends an HTTP GET request to the URL that represents the function. Binding allows the client to send the GET request to a URL that represents the function with the binding parameter that is already specified.

In the following example, the binding parameter (called "Customer"; see section 6) to the HasOrderFor function is the resource identified by the URL preceding the fully-qualified function name. In this example, the remaining parameters (productType) are specified in the query string according to the rules specified in section 2.2.7.5.2.

In this example, the request asked for a response formatted in Verbose JSON, so the Boolean return value for this function is returned in a standard OData Verbose JSON response, with a name/value pair, where the name is the name of the function and value is returnType serialized in OData Verbose JSON format.

HTTP Request:

```
GET /Customers('ALFKI')/SampleEntities.HasOrderFor?productType='luxury goods' HTTP/1.1
Host: host
Accept: application/json;odata=verbose
DataServiceVersion: 3.0
MaxDataServiceVersion: 3.0
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Fri, 11 Oct 2008 04:23:49 GMT
DataServiceVersion: 3.0
 {
     "d" : {
       "HasOrderFor": true
     }
  }
```

5 Security

5.1 Security Considerations for Implementers

Implementers of the protocol defined in this document are encouraged to review sections 14 and 15 of [RFC5023], which outline security considerations for the Atom Publishing Protocol. Such considerations apply directly to the protocol defined in this document.

5.2 Index of Security Parameters

None.

6 Appendix A: Sample Entity Data Model and CSDL Document

An Entity Data Model conceptual schema, as specified in [MC-CSDL], is an XML document written with the conceptual schema definition language (CSDL), which describes entities and the associations between entities.

The following example conceptual schema definition language (CSDL) document defines seven EntityTypes (Customer, Order, OrdersLine, Employee, Document, Company, and Photo), each with **Primitive** type properties and some with NavigationProperties (which refer to the associations between the entities and NamedStream properties). In the example conceptual schema definition language (CSDL) document, a 1-to-many association exists between Customer and Order entities and a 1-to-many association exists between Order and OrderLine entities.

The conceptual schema definition language (CSDL) document also defines a single **FunctionImport**, as described in [MC-CSDL], named "CustomersByCity", which returns a collection of Customer **EntityType** instances in a particular city.

All examples in this document use the conceptual schema definition language (CSDL) and sample data set defined below.

Sample Data:

- A Customer **EntityType** instance exists with **EntityKey** value ALFKI.
- A total of 91 Customer **EntityType** instances exist.
- An Employee **EntityType** instance exists with EntityKey value 1.
- Two Order **EntityType** instances exist, one with **EntityKey** value 1 and the other with **EntityKey** value 2. Order 1 and 2 are associated with Customer ALFKI.
- Two OrderLine **EntityType** instances exist, one with **EntityKey** value 100 and the other with **EntityKey** value 200. OrderLine 100 is associated with Order 1 and OrderLine 200 with Order 2.
- Two Document **EntityType** instances exist, one with **EntityKey** value 300 and the other with **EntityKey** value 301.

URI: The scheme and Service Root for this sample is `http://host/service.svc`.

Valid URIs that identify resources described using the conceptual schema definition language (CSDL) below are:

All Customers:

```
http://host/service.svc/Customers
```

Customer with key "ALFKI":

```
http://host/service.svc/Customers('ALFKI')
```

Orders for the Customer with key "ALFKI":

```
http://host/service.svc/Customers('ALFKI')/Orders
```

OrderLines for Order 1 associated with Customer ALFKI:

```
http://host/service.svc/Customers('ALFKI')/Orders(1)/OrderLines
```

The metadata document CSDL for the service:

```
http://host/service.svc/$metadata
```

The rules for constructing URIs which address aspects of an Entity Data Model are defined in Abstract Type System (section 2.2.2).

CSDL Document:

Note The conceptual schema definition language (CSDL) document below is shown within an <edmx:DataServices> element, as specified in [MC-EDMX].

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<edmx:Edmx Version="1.0"
       xmlns:edmx="http://schemas.microsoft.com/ado/2010/02/edmx"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<edmx:DataServices m:DataServiceVersion="3.0">
   <Schema Namespace="SampleModel"
     xmlns="http://schemas.microsoft.com/ado/2006/04/edm">
      <EntityContainer Name="SampleEntities"
                       m:IsDefaultEntityContainer="true">
        <EntitySet Name="Customers" EntityType="SampleModel.Customer" />
        <EntitySet Name="Orders" EntityType="SampleModel.Order" />
        <EntitySet Name="OrderLines" EntityType="SampleModel.OrderLine" />
        <EntitySet Name="Employees" EntityType="SampleModel.Employee" />
        <EntitySet Name="Documents" EntityType="SampleModel.Document" />
        <EntitySet Name="Companies" EntityType="SampleModel.Company" />
        <EntitySet Name="Photos" EntityType="SampleModel.Photo" />

        <AssociationSet Name="Orders_Customers"
             Association="SampleModel.Orders_Customers">
            <End Role="Customers" EntitySet="Customers" />
            <End Role="Orders" EntitySet="Orders" />
        </AssociationSet>
        <AssociationSet Name="OrderLines_Orders"
             Association="SampleModel.OrderLines_Orders">
            <End Role="OrderLine" EntitySet="OrderLines" />
            <End Role="Order" EntitySet="Orders" />
        </AssociationSet>

        <!-- Service Operation -->
        <FunctionImport Name="CustomersByCity"
                        EntitySet="Customers"
                        ReturnType="Collection(SampleModel.Customer)"
                        m:HttpMethod="GET">
           <Parameter Name="city" Type="Edm.String" Mode="In" />
        </FunctionImport>

        <!-- Functions -->
        <FunctionImport Name="GetRelatedCustomers"
                        EntitySet="Customers"
                        IsBindable="true"
                        ReturnType="Collection(SampleModel.Customer)"
                        IsSideEffecting="false">
           <Parameter Name="company" Type="SampleModel.Company" Mode="In" />
        </FunctionImport>


        <FunctionImport Name="TopTenCustomers"
                        EntitySet="Customers"
                        IsBindable="true"
                        ReturnType="Collection(SampleModel.Customer)"
```

```
                IsSideEffecting="false">
      <Parameter Name="customers" Type="Collection(SampleModel.Customer)" Mode="In" />
    </FunctionImport>

    <FunctionImport Name="Best"
                EntitySet="Customers"
                IsBindable="true"
                ReturnType="SampleModel.Customer"
                IsSideEffecting="false"
                m:IsAlwaysBindable="false">
      <Parameter Name="customers" Type="Collection(SampleModel.Customer)" Mode="In" />
    </FunctionImport>

    <FunctionImport Name="TopTenOrders"
                EntitySet="Orders"
                IsBindable="true"
                ReturnType="Collection(SampleModel.Order)"
                IsSideEffecting="false">
      <Parameter Name="customer" Type="SampleModel.Customer" Mode="In" />
    </FunctionImport>

    <FunctionImport Name="TopTenCustomersInCity"
                EntitySet="Customers"
                IsBindable="true"
                ReturnType="Collection(SampleModel.Customer)"
                IsSideEffecting="false">
      <Parameter Name="city" Type="Edm.String" Mode="In" />
    </FunctionImport>

    <FunctionImport Name="HasOrderFor"
                IsBindable="true"
                ReturnType="Edm.Boolean"
                IsSideEffecting="false">
      <Parameter Name="customer" Type="SampleModel.Customer" Mode="In" />
      <Parameter Name="productType" Type="Edm.String" Mode="In" />
    </FunctionImport>

    <!-- Actions -->
    <FunctionImport Name="CreateOrder"
                EntitySet="Orders"
                IsBindable="true"
                IsSideEffecting="true"
                m:IsAlwaysBindable="true">
      <Parameter Name="customer" Type="SampleModel.Customer" Mode="In" />
      <Parameter Name="quantity" Type="Edm.Int32" Mode="In" />
      <Parameter Name="discountCode" Type="Edm.String" Mode="In" />
    </FunctionImport>

    <FunctionImport Name="Audit"
                IsBindable="true"
                IsSideEffecting="true">
      <Parameter Name="company" Type="SampleModel.Company" Mode="In" />
    </FunctionImport>


  </EntityContainer>

  <EntityType Name="Order">
    <Key>
      <PropertyRef Name="OrderID" />
    </Key>
    <Property Name="OrderID" Type="Edm.Int32" Nullable="false" />
    <Property Name="ShippedDate" Type="Edm.DateTime" Nullable="true"
      DateTimeKind="Unspecified" PreserveSeconds="true" />
    <NavigationProperty Name="Customer"
      Relationship="SampleModel.Orders_Customers"
      FromRole="Order" ToRole="Customer" />
    <NavigationProperty Name="OrderLines"
      Relationship="SampleModel.OrderLines_Orders"
      FromRole="Order" ToRole="OrderLine" />
```

```
      </EntityType>

      <EntityType Name="OrderLine">
        <Key>
          <PropertyRef Name="OrderLineID" />
        </Key>
        <Property Name="OrderLineID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Quantity" Type="Edm.Int32" Nullable="false" />
        <Property Name="UnitPrice" Type="Edm.Decimal" Nullable="false" />
        <NavigationProperty Name="Order"
           Relationship="SampleModel.OrderLines_Orders"
           FromRole="OrderLine" ToRole="Order" />
      </EntityType>

      <EntityType Name="Customer">
        <Key>
          <PropertyRef Name="CustomerID" />
        </Key>
        <Property Name="CustomerID" Type="Edm.String" Nullable="false"
           MaxLength="5" Unicode="true" FixedLength="true" />
        <Property Name="CompanyName" Type="Edm.String" Nullable="false"
           MaxLength="40" Unicode="true" FixedLength="false" />
        <Property Name="Address" Type="Sample.CAddress" Nullable="true" />
        <Property Name="Version" Type="Edm.Binary" Nullable="true" MaxLength="8"
           FixedLength="true" ConcurrencyMode="Fixed" />
        <Property Name="EmailAddresses" Type="Collection"  Nullable="false">
          <TypeRef Type ="Edm.String" Nullable="false"/>
        </Property>
        <Property Name="AlternateAddresses" Type="Collection" Nullable="false">
            <TypeRef Type ="SampleModel.Address" Nullable="false"/>
         </Property>
        <NavigationProperty Name="Orders"
           Relationship="NorthwindModel.Orders_Customers"
           FromRole="Customer" ToRole="Order" />
      </EntityType>
<EntityType Name="VipCustomer" baseType="SampleModel.Customer"
 m:HasStream="true" >
        <Property Name="CreditPurchases" Type="SampleModel.CustomerCredit.Int32"
Nullable="false"/>
        <Property Name="Logo" Type="Edm.Stream" />
        <Property Name="CountriesOfOperation" Type="Collection" Nullable="false">
            <TypeRef Type ="Edm.String" Nullable="false"/>
         </Property>
        <NavigationProperty Name="InHouseStaff"
           Relationship="NorthwindModel.Employee_VipCustomer"
           FromRole="VipCustomer" ToRole="Employee" />
      </EntityType>
      <EntityType Name="GovernmentOrder" baseType="SampleModel.Order" >
        <Property Name="Country" Type="SampleModel.String"  Nullable="false"/>
      </EntityType>

<!-- The Employee EntityType has Web Customizable Feed property mappings that are supported
only in OData 2.0 and OData 3.0 -->
      <EntityType Name="Employee" m:FC_KeepInContent="true"
           m:FC_TargetPath="Location" m:FC_SourcePath="Address/City"
           m:FC_NsUri="http://www.microsoft.com" m:FC_NsPrefix="emp">
        <Key>
          <PropertyRef Name="EmployeeID" />
        </Key>
        <Property Name="EmployeeID" Type="Edm.String" Nullable="false"
           MaxLength="5" Unicode="true" FixedLength="true" />
        <Property Name="EmployeeName" Type="Edm.String" Nullable="false"
           MaxLength="40" Unicode="true" FixedLength="false"
           m:FC_KeepInContent="false"
           m:FC_TargetPath="SyndicationTitle"/>
        <Property Name="Address" Type="Sample.EAddress" Nullable="true" />
        <Property Name="Version" Type="Edm.Binary" Nullable="true" MaxLength="8"
           FixedLength="true" ConcurrencyMode="Fixed" />
      </EntityType>
```

```
      <EntityType Name="Document" m:HasStream="true">
        <Key>
          <PropertyRef Name="DocumentID" />
        </Key>
        <Property Name="DocumentID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Title" Type="Edm.String" Unicode="true" />
        <Property Name="Author" Type="Edm.String" Unicode="true" />
      </EntityType>
      <EntityType Name="Company">
        <Key>
          <PropertyRef Name="CompanyID" />
        </Key>
        <Property Name="CompanyID" Type="Edm.String" Nullable="false"
           MaxLength="5" Unicode="true" FixedLength="true" />
        <Property Name="CompanySize" Type="Edm.String" Nullable="true"/>
      </EntityType>

      <ComplexType Name="Address" BaseType="SampleModel.EAddress">
        <Property Name="Apartment" Type="Edm.Int"/>      </ComplexType>
      <EntityType Name="Photo" m:HasStream="true">
        <Key>
          <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Name" Type="Edm.String" Nullable="true" />
        <Property Name="Thumbnail" Type="Edm.Stream" />
        <Property Name="PrintReady" Type="Edm.Stream" />
      </EntityType>

      <ComplexType Name="EAddress">
        <Property Name="Street" Type="Edm.String" Unicode="true" />
        <Property Name="City" Type="Edm.String" Unicode="true"/>
      </ComplexType>

      <ComplexType Name="CAddress">
        <Property Name="Street" Type="Edm.String" Unicode="true" />
        <Property Name="City" Type="Edm.String" Unicode="true"/>
        <Property Name="Location" Type="Edm.GeographyPoint" SRID="4326"/>
      </ComplexType>
      <ComplexType Name="CustomerCredit">
        <Property Name="CreditLimit" Type="Edm.Int32" />
        <Property Name="Balance" Type="Edm.Int32" />
      </ComplexType>

      <Association Name="Orders_Customers">
        <End Role="Customer" Type="SampleModel.Customer"
             Multiplicity="0..1" />
        <End Role="Order" Type="SampleModel.Order" Multiplicity="*" />
      </Association>

      <Association Name="OrderLines_Orders">
        <End Role="Order" Type="SampleModel.OrderLine"
             Multiplicity="*" />
        <End Role="OrderLine" Type="SampleModel.Order" Multiplicity="0..1" />
      </Association>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

This document specifies version-specific details in the Microsoft .NET Framework. For information about which versions of .NET Framework are available in each released Windows product or as supplemental software, see [MS-NETOD] section 4.

- Microsoft .NET Framework 3.5 Service Pack 1 (SP1)
- Microsoft .NET Framework 4.0
- Microsoft .NET Framework 4.5
- Microsoft .NET Framework 4.6
- Microsoft .NET Framework 4.7

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 1.7: The following table describes, as specified in this document, which versions of the OData protocol are supported by which versions of .NET Framework.

OData version	.NET Framework 3.5 SP1	Data Services Update for .NET Framework 3.5 SP1, KB#976127	.NET Framework 4.0/4.5/4.6/4.7	WCF Data Services 5* for .NET Framework 4.0/4.5/4.6/4.7
OData 1.0	X	X	X	X
OData 2.0		X	X	X
OData 3.0				X

*WCF Data Services 5 includes WCF Data Services Server, WCF Data Services Client, and ODataLib.

<2> Section 2.2.3.4: The data service client library always appends parentheses after **EntitySet** names. The data service server library accepts **EntitySet** names appended with parentheses, but never generates such URIs in response payloads or HTTP response headers.

<3> Section 2.2.3.4: The data service client and server libraries generate key predicates of the form "<Entity Type property name> = <Entity Type property value>" in the case where multiple properties form the key. Otherwise, the canonical form for a single property **EntityKey**, as seen in section 2.2.3.4, is used.

<4> Section 2.2.3.5: The data service client library always appends parentheses after **EntitySet** names or NavigationProperty names identifying a collection of entities. The data service server library accepts **EntitySet** names and NavigationProperty names that identify a collection of entities appended with parentheses, but never generates such URIs in response payloads or HTTP response headers.

<5> Section 2.2.3.6: The data service client library always generates a single system query option of a particular type. For example, if a complex filter expression is defined, a single **$filter** system query option (section 2.2.3.6.1.4) will be present.

<6> Section 2.2.3.6.1: The data service client library and server libraries support all system query options except the **$format** option, as specified in Format System Query Option ($format) (section 2.2.3.6.1.5). No support is provided in .NET Framework 3.5 SP1 for **$skiptoken**, as specified in Skip Token System Query Option ($skiptoken) (section 2.2.3.6.1.9); **$inlinecount**, as specified in InlineCount System Query Option ($inlinecount) (section 2.2.3.6.1.10); or **$select**, as specified in Select System Query Option ($select) (section 2.2.3.6.1.11).

<7> Section 2.2.3.6.1.1: The data service server library supports all of the expressions.

<8> Section 2.2.3.6.1.1: The data service server library supports all of the expressions.

<9> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<10> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<11> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<12> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<13> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<14> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<15> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<16> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<17> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<18> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<19> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<20> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<21> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<22> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<23> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<24> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<25> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<26> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<27> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<28> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<29> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<30> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<31> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<32> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<33> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<34> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<35> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<36> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<37> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<38> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<39> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<40> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<41> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<42> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<43> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<44> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<45> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<46> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<47> Section 2.2.3.6.1.1.1: The data service server library supports all of the expressions.

<48> Section 2.2.3.6.1.1.1: The data service server library supports all of the allowable expressions.

<49> Section 2.2.3.6.1.1.2: The data service server library supports all of the expressions.

<50> Section 2.2.3.6.1.1.5: The data service library in the .NET Framework will not return null as a result of a logical operator if an operand is also null.

<51> Section 2.2.3.7.2: The data service server library includes the **mimeType** attribute only if the developer who authors the service explicitly defines a media type for the property. By default, the **mimeType** attribute is not included on the definition of a property.

<52> Section 2.2.5.4: In the data service server library the default behavior is that each **EntityType** does not define a concurrency token. Definition of a concurrency token requires an explicit step by the developer who authors the data service that uses the server library.

<53> Section 2.2.5.5: The data service client library includes the If-Match header only if the **EntityType** associated with the request defines a concurrency token.

<54> Section 2.2.5.5: The data service client library includes the If-Match header only if the **EntityType** associated with the request defines a concurrency token.

<55> Section 2.2.5.6: The data service client library includes the If-None-Match header only if the **EntityType** associated with the request defines a concurrency token.

<56> Section 2.2.5.6: The data service client library includes the If-None-Match header only if the EntityType associated with the request defines a concurrency token.

<57> Section 2.2.5.8: The data service client library and server library support verb tunneling.

<58> Section 2.2.6.2: The data service client library and server library use only the request and response messages that are defined in this document.

<59> Section 2.2.6.2.2: The data service libraries do not write the **m:properties** element if it is empty.

<60> Section 2.2.6.2.2: The data service client library and server library do not generate or parse self links.

<61> Section 2.2.6.3: The data service client library and server library in the .NET Framework support the Verbose JSON format in OData 1.0, OData 2.0, and OData 3.0.

<62> Section 2.2.6.3: The data service client library and server library in the .NET Framework support the preferred OData 3.0 JSON format only in OData 3.0.

<63> Section 2.2.6.3.3: The data service server library generates and parses the "__metadata" name/value pair.

<64> Section 2.2.6.4.1: The data service server library alters the media type associated with an **EntityType** property only if the developer who authors the service explicitly defines a media type for the property.

<65> Section 2.2.6.6: The data service client library and server library in the .NET Framework support the preferred OData 3.0 JSON format only in OData 3.0.

<66> Section 2.2.7.4.1: By default, the server library does not cause any additional side effects on the data model. Lower or higher layers can cause side effects.

<67> Section 2.2.7.6: The data service server library does implement Batch request

<68> Section 2.2.7.6: The data service client library does not support creating a Batch request that includes both a query operation and a change set.

<69> Section 3.2.5.1: The data service libraries do not implement any access control policies. Such policies can be applied by a higher or lower layer.

<70> Section 3.2.5.2: By default, the server library does not cause any additional side effects on the data model. Lower or higher layers can cause side effects.

<71> Section 3.2.5.3: The data service server library in the .NET Framework will return a 500 response code instead of a 4*xx* response code if an Insert request is received with an empty value for a data service resource and the type of that resource does not permit an empty value.

<72> Section 3.2.5.3.1: By default, the data service client library does not omit any constructs.

<73> Section 3.2.5.5: The server library returns a 500 response code rather than the required 4*xx* response code when an Update request is received that would set the value of a resource to empty when the type of that resource does not define an empty state.

<74> Section 3.2.8: The data service server library in the .NET Framework will return an empty collection of entities with a 200 response code instead of a 404 response code if the last URI path segment in the request URI is a NavigationProperty name that identifies a collection of entities.

8 Change Tracking

~~No table of~~ This section identifies changes ~~is available. The~~ that were made to this document ~~is either new or has had no changes~~ since ~~its~~the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
1.7 Versioning and Capability Negotiation	Added .NET Framework 4.7 to the appropriate products listed in the table headings.	Major
7 Appendix B: Product Behavior	Added .NET Framework 4.7 to product applicability list.	Major

9 Index

A

A single entity exposing a function by using the Verbose JSON format
  retrieving count 260
A single entity exposing an action by using the AtomPub format
  retrieving count 257
A single entity exposing an action by using the Verbose JSON format
  retrieving count 258
A single entity exposing function by using the AtomPub format
  retrieving count 259
Abstract data model
  actions 26
  client 222
  functions 28
  message syntax 24
  named resource stream instances 26
  named resource streams 26
  server 228
  service operations 29
Abstract Data Model message 24
Abstract type system 30
Abstract Type System message 30
Accept HTTP header field
  application/atom+xml 114
  application/json 115
  application/json;odata=verbose 115
  overview 113
Actions
  abstract data model 26
Annotations (section 2.2.6.2.12 140, section 2.2.6.3.16 167)
  AtomPub format 164
  Verbose JSON format 167
Applicability 15
AtomPub format
  additional representations 137
  annotations 164
  collection of entities with Inline Count 243
  collection of entities with named resource streams 245
  collection property 137
  Complex Type property 133
  deferred content
    inline representation 135
    overview 134
  EDMSimpleType property 134
  Entity Set element 125
  Entity Type element 129
  links and subtypes 140
  merge-based update 263
  named resource streams 138
  navigation property 134
  overview 124
  replace-based update 261
  retrieving collection of entities 238
  retrieving single entity (section 4.2.1.3 240, section 4.2.2 248)
  retrieving single entity and related entities 251
  retrieving single entity with a mapped property 249
  service document 137
  updating relationship 265

B

Batch request
  Change Set syntax 211
  example (section 2.2.7.6.5 214, section 4.5 266)
  example response (section 2.2.7.6.8 217, section 4.5 266)
  executing received 235
  HTTP request restrictions 213
  overview 210
  Query Operation syntax 212
  response syntax 216
  responses 215
  syntax 213

C

Canonical URIs 111
Capability negotiation 15
Change Set syntax
  overview 211
  referencing requests 212
Change tracking 282
Client
  abstract data model 222
  higher-layer triggered events 222
    common rules for all requests 222
    overview 222
    request to delete resources 225
    request to insert resources 222
    request to invoke a function 227
    request to invoke an action 226
    request to invoke service operation 225
    request to retrieve resources 223
    request to send batch of operations 226
    request to update resources 224
  initialization 222
  local events 228
  message processing
    receiving responses to data service requests 227
    receiving responses to insert requests 228
  other local events 228
  sequencing rules
    receiving responses to data service requests 227
    receiving responses to insert requests 228
  timer events 228
  timers 222
Collection of entities
  retrieving count 257
Collection property
  AtomPub format 137
Common expression syntax
  binary numeric promotions 86
  construction and evaluation 64
  lifted operators 87
  numeric promotions for method call parameters 88
  operator precedence 85
  overview 59
  unary numeric promotions 86
Common Payload Syntax message 121
Complex Type property
  AtomPub format 133
  Verbose JSON format 158
  XML format 168
Construction rules 44
Containment
  abstract data model 29
Content-Type HTTP header field 115

CSDL document 273

D

Data model
  abstract
    client 222
    message syntax 24
    server 228
  sample Entity 273
Data model - abstract
  client 222
  server 228
Data service metadata
  Conceptual Schema Definition Language document for data services 104
  service document 104
Data service request
  receiving responses to 227
  rules for receiving 228
Dataserviceid HTTP header field
  overview 121
DataServiceVersion HTTP header field 115
Delete request types
  DeleteEntity 204
  DeleteLink 205
  DeleteValue 206
  executing received request 235
  rules for executing received request 229
  sending request 225

E

EDMSimpleType property
  AtomPub format 134
  raw format 168
  Verbose JSON format 160
  XML format 169
Entity data model 273
Entity Set element (section 2.2.6.2.1.2 127, section 2.2.6.2.1.3 128)
  AtomPub format 125
  Verbose JSON format 148
Entity type 132
Entity Type element
  AtomPub format 129
  Verbose JSON format 153
ETag HTTP header field 116
Examples
  inserting new entity 238
  invoking a function 270
  invoking an action 270
  overview 238
  retrieving resources
    a single entity exposing a function by using the AtomPub format 259
    a single entity exposing a function by using the Verbose JSON format 260
    a single entity exposing an action by using the AtomPub format 257
    a single entity exposing an action by using the Verbose JSON format 258
    collection of entities 238
    collection of entities with Inline Count by using AtomPub format 243
    collection of entities with Inline Count by using Verbose JSON format 244
    collection of entities with named resource streams by using AtomPub format 245
    collection of entities with named resource streams by using Verbose JSON format 247
    data service's metadata document 254
    partial collection of entities by using Verbose JSON format 242
    single entity and related entities by using AtomPub format 251
    single entity and related entities by using Verbose JSON format 253

single entity by using AtomPub format (section 4.2.1.3 240, section 4.2.2 248)
single entity by using Verbose JSON format 250
single entity with a mapped property by using AtomPub format 249
the count of collection of entities 257
updating existing entity
merge-based update by using Verbose JSON format 264
merge-based update using AtomPub format 263
replace-based update by using AtomPub format 261
replace-based update by using Verbose JSON format 262
updating relationship between two entities
by using Verbose JSON format 265
deleting existing entity 266
using AtomPub format 265

F

Fields - vendor-extensible 23
Functions
abstract data model 28

G

Glossary 10

H

Higher-layer triggered events
client 222
common rules for all requests 222
overview 222
request to delete resources 225
request to insert resources 222
request to invoke a function 227
request to invoke an action 226
request to invoke service operation 225
request to retrieve resources 223
request to send batch of operations 226
request to update resources 224
server 228
HTTP header fields
Accept 113
Content-Type 115
dataserviceid 121
DataServiceVersion 115
ETag 116
If-Match 117
If-None-Match 118
MaxDataServiceVersion 118
overview 113
prefer 120
preference-applied 120
X-HTTP-Method 119
HTTP Header Fields message 113
HTTP methods
MERGE 112
overview 112
PATCH 112
HTTP Methods message 112
HTTP request restrictions 213

I

If-Match HTTP header field 117
If-None-Match HTTP header field 118
Implementer - security considerations 272

Index of security parameters 272
Informative references 14
Initialization
  client 222
  server 228
Insert examples 238
Insert request types
  InsertEntity
    examples 174
    executing received request 231
    overview 172
    sending request 223
  InsertLink
    executing received request 231
    overview 179
    sending request 223
  overview 172
  receiving responses to requests 228
  rules for executing received request (section 3.2.5.2 229, section 3.2.5.3 230)
  UpdateEntity - executing received request 234
Introduction 10
Invoke Action request
  executing received 236
Invoke Function request
  executing received 236
Invoke request
  executing received 235
  overview 207
Invoke request types
  action 208
  Function 210
Invoking a function example 270
Invoking an action example 270

L

Links and subtypes
  AtomPub format 140
Local events
  client 228
  server 237

M

MaxDataServiceVersion HTTP header field 118
Media resource
  insert new 266
  query existing 268
  update 267
MERGE HTTP method 112
Message processing
  client
    receiving responses to data service requests 227
    receiving responses to insert requests 228
  server
    executing received Batch request 235
    executing received data service request 229
    executing received Delete request 235
    executing received Insert request 230
    executing received Invoke Action request 236
    executing received Invoke Function request 236
    executing received Invoke request 235
    executing received Retrieve request 232
    executing received Update request 233
    rules for receiving data service requests 228

Messages
Abstract Data Model 24
Abstract Type System 30
Common Payload Syntax 121
HTTP Header Fields 113
HTTP Methods 112
Request Types 171
syntax
abstract data model 24
abstract type system 30
common payload 121
HTTP header fields 113
HTTP methods 112
overview 24
request types 171
resource addressing rules 38
response types 219
transport 24
URI Format: Resource Addressing Rules 38

N

Named resource stream instance
retrieve 268
update 269
named resource stream instance - new
deleting 269
inserting 269
Named resource stream instances 26
Named resource streams 26
AtomPub format 138
Navigation property
AtomPub format 134
Verbose JSON format 159
Normative references 12
Numeric promotions
binary 86
for method call parameters 88
unary 86

O

Operations - unsupported
deleting new named resource stream instance 269
inserting new named resource stream instance 269
Other local events
client 228
server 237
Overview (synopsis) 14

P

Parameters - security index 272
PATCH HTTP method 112
Payload syntax
AtomPub format 124
common serialization rules for XML-based formats 121
overview 121
raw format 167
Verbose JSON format 143
XML format 168
Preconditions 15
Prefer HTTP header field
overview 120
Preference-applied HTTP header field

overview 120
Prerequisites 15
Product behavior 278

Q

Query Operation syntax 212
Query options
custom 101
overview 56
service operation parameters 101
system
common expression syntax 59
evaluating 91
expanding 91
filter 92
format 93
OrderBy 94
overview 57
skip 95
top 95

R

Raw format
EDMSimpleType property 168
overview 167
References 12
informative 14
normative 12
Relationship to other protocols 14
Request types
Batch 210
delete 204
insert 172
Invoke 207
overview 171
retrieve 182
tunneled 219
update 194
Request Types message 171
Requests - client
common rules 222
deleting resources
common rules for sending all Delete requests 225
overview 225
inserting resources
overview 222
sending InsertEntity request 223
sending InsertLink request 223
invoking a function 227
invoking an action 226
invoking service operation 225
retrieving resources
common rules for sending Retrieve requests 223
overview 223
sending batch of operations 226
updating resources
common rules for sending Update requests 224
overview 224
Resource addressing rules
canonical URIs 111
data service metadata 104
overview 38
query options 56

resource path 44
service root 44
URI equivalence 111
URI syntax 39
Resource path 44
Response codes 237
Response types - error
overview 219
Verbose JSON 221
XML 220
Retrieve request examples
a single entity exposing a function by using the AtomPub format 259
a single entity exposing a function by using the Verbose JSON format 260
a single entity exposing an action by using the AtomPub format 257
a single entity exposing an action by using the Verbose JSON format 258
collection of entities with Inline Count by using AtomPub format 243
collection of entities with Inline Count by using Verbose JSON format 244
collection of entities with named resource streams by using AtomPub format 245
collection of entities with named resource streams by using Verbose JSON format 247
partial collection of entities by using Verbose JSON format 242
retrieving collection of entities
by using AtomPub format 238
by using Verbose JSON format 239
retrieving data service's metadata document 254
retrieving single entity and related entities by using AtomPub format 251
retrieving single entity and related entities by using Verbose JSON format 253
single entity by using AtomPub format (section 4.2.1.3 240, section 4.2.2 248)
single entity by using Verbose JSON format 250
single entity with a mapped property by using AtomPub format 249
the count of collection of entities 257
Retrieve request types
executing received request
overview 232
RetrieveValue 232
RetrieveCollectionProperty 188
RetrieveComplexType 185
RetrieveEntity 183
RetrieveEntitySet 182
RetrieveLink 191
RetrievePrimitiveProperty 186
RetrieveServiceDocument 190
RetrieveServiceMetadata 189
RetrieveValue 187
sending request 223

S

Security
implementer considerations 272
parameter index 272
Security - implementer considerations 272
Semantics 47
Sequencing rules
client
receiving responses to data service requests 227
receiving responses to insert requests 228
server
executing received Batch request 235
executing received data service request 229
executing received Delete request 235
executing received Insert request 230
executing received Invoke Action request 236
executing received Invoke Function request 236
executing received Invoke request 235
executing received Retrieve request 232

executing received Update request 233
rules for receiving data service requests 228
Serialization rules
EDM constructs 143
XML formats 121
Server
abstract data model 228
higher-layer triggered events 228
initialization 228
local events 237
message processing
executing received Batch request 235
executing received data service request 229
executing received Delete request 235
executing received Insert request 230
executing received Invoke Action request 236
executing received Invoke Function request 236
executing received Invoke request 235
executing received Retrieve request 232
executing received Update request 233
rules for receiving data service requests 228
other local events 237
response codes 237
sequencing rules
executing received Batch request 235
executing received data service request 229
executing received Delete request 235
executing received Insert request 230
executing received Invoke Action request 236
executing received Invoke Function request 236
executing received Invoke request 235
executing received Retrieve request 232
executing received Update request 233
rules for receiving data service requests 228
timer events 237
timers 228
Service operations
abstract data model 29
Service root 44
Standards assignments 23
Syntax
abstract data model 24
abstract type system 30
common payload 121
HTTP header fields 113
HTTP methods 112
overview 24
request types 171
resource addressing rules 38
response types 219

T

Timer events
client 228
server 237
Timers
client 222
server 228
Tracking changes 282
Transport 24
Triggered events - higher-layer
client 222
common rules for all requests 222
overview 222

request to delete resources 225
request to insert resources 222
request to invoke a function 227
request to invoke an action 226
request to invoke service operation 225
request to retrieve resources 223
request to send batch of operations 226
request to update resources 224
server 228
Tunneled request type 219

U

Update request examples
merge-based update by using Verbose JSON format 264
merge-based update using AtomPub format 263
replace-based update by using AtomPub format 261
replace-based update by using Verbose JSON format 262
Update request types
rules for executing received request (section 3.2.5.2 229, section 3.2.5.5 233)
sending request 224
UpdateComplexType 196
UpdateEntity
example 196
overview 194
UpdateLink 201
UpdatePrimitiveProperty 198
UpdateValue 200
Updating relationship examples
by using Verbose JSON format 265
deleting existing entity 266
using AtomPub format 265
URI format
canonical URIs 111
data service metadata 104
equivalence 111
overview 38
query options 56
resource path 44
service root 44
syntax 39
URI Format: Resource Addressing Rules message 38

V

Vendor-extensible fields 23
Verbose JSON format
deferred content
overview 161
Verbose JSON format
annotations 167
collection of Complex Type instances 158
collection of EDMSimpleType values 160
collection of entities with Inline Count 244
collection of entities with named resource streams 247
collection property 165
common serialization rules for all EDM constructs 143
Complex Type property 158
deferred content
inline representation 161
overview 161
EDMSimpleType property 160
Entity Set element 148
Entity Type element 153
error response 221

links 163
links and subtypes 167
merge-based update 264
named resource streams 165
Navigation property 159
overview 143
replace-based update 262
retrieving collection of entities 239
retrieving partial collection of entities 242
retrieving single entity 250
retrieving single entity and related entities 253
service document 164
updating relationship 265
Versioning 15

X

X-HTTP-Method HTTP header field 119
XML format
collection of complex type 170
collection of Complex Type instances 168
collection of EDMSimpleType 171
collection of EDMSimpleType values 169
Complex Type property 168
EDMSimpleType property 169
error response 220
links 169
overview 168