[MS-ODATA]:
Open Data Protocol (OData) Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **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 may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications 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 may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events 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 specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do 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 are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/27/2009	0.1	Major	First Release.
04/10/2009	0.2	Minor	Updated the technical content.
05/22/2009	0.2.1	Editorial	Revised and edited the technical content.
07/02/2009	1.0	Major	Updated and revised the technical content.
08/14/2009	1.1	Minor	Updated the technical content.
09/25/2009	1.2	Minor	Updated the technical content.
11/06/2009	1.3	Minor	Updated the technical content.
12/18/2009	1.3.1	Editorial	Revised and edited the technical content.
01/29/2010	1.4	Minor	Updated the technical content.
03/12/2010	2.0	Major	Updated and revised the technical content.
04/23/2010	2.0.1	Editorial	Revised and edited the technical content.
06/04/2010	3.0	Major	Updated and revised the technical content.
07/16/2010	4.0	Major	Significantly changed the technical content.
08/27/2010	5.0	Major	Significantly changed the technical content.
10/08/2010	5.1	Minor	Clarified the meaning of the technical content.
11/19/2010	6.0	Major	Significantly changed the technical content.
01/07/2011	7.0	Major	Significantly changed the technical content.
02/11/2011	8.0	Major	Significantly changed the technical content.
03/25/2011	9.0	Major	Significantly changed the technical content.
05/06/2011	10.0	Major	Significantly changed the technical content.
06/17/2011	10.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction 8
1.1 Glossary 8
1.2 References 10
1.2.1 Normative References 10
1.2.2 Informative References 12
1.3 Overview 12
1.4 Relationship to Other Protocols 13
1.5 Prerequisites/Preconditions 13
1.6 Applicability Statement 13
1.7 Versioning and Capability Negotiation 14
1.7.1 Version 2.0 Summary 15
1.8 Vendor-Extensible Fields 17
1.9 Standards Assignments 17

2 Messages 18
2.1 Transport 18
2.2 Message Syntax 18
2.2.1 Abstract Data Model 18
2.2.2 Abstract Type System 19
2.2.3 URI Format: Resource Addressing Rules 23
2.2.3.1 URI Syntax 24
2.2.3.2 Service Root (serviceRoot) and Path Prefix (pathPrefix) 27
2.2.3.3 Resource Path (resourcePath) 27
2.2.3.4 Resource Path: Construction Rules 28
2.2.3.5 Resource Path: Semantics 30
2.2.3.6 Query Options 35
2.2.3.6.1 System Query Options 35
2.2.3.6.1.1 Common Expression Syntax 37
2.2.3.6.1.1.1 Expression Construction and Evaluation Rules 40
2.2.3.6.1.1.2 Operator Precedence 58
2.2.3.6.1.1.3 Unary Numeric Promotions 59
2.2.3.6.1.1.4 Binary Numeric Promotions 59
2.2.3.6.1.1.5 Lifted Operators 60
2.2.3.6.1.1.6 Numeric Promotions for Method Call Parameters 61
2.2.3.6.1.2 Evaluating System Query Options 62
2.2.3.6.1.3 Expand System Query Option ($expand) 63
2.2.3.6.1.4 Filter System Query Option ($filter) 64
2.2.3.6.1.5 Format System Query Option ($format) 64
2.2.3.6.1.6 OrderBy System Query Option ($orderby) 65
2.2.3.6.1.7 Skip System Query Option ($skip) 65
2.2.3.6.1.8 Top System Query Option ($top) 66
2.2.3.6.1.9 Skip Token System Query Option ($skiptoken) 67
2.2.3.6.1.10 Inlinecount System Query Option ($inlinecount) 67
2.2.3.6.1.11 Select System Query Option ($select) 68
2.2.3.6.2 Custom Query Options 70
2.2.3.6.3 Service Operation Parameters 70
2.2.3.7 Data Service Metadata 71
2.2.3.7.1 Service Document 71
2.2.3.7.2 Conceptual Schema Definition Language Document for Data Services 71

2.2.3.7.2.1 Conceptual Schema Definition Language Document for Version 2.0 Data Services ... 75
2.2.3.8 URI Equivalence ... 78
2.2.3.9 Canonical URIs ... 78
2.2.4 HTTP Methods ... 79
2.2.4.1 MERGE ... 79
2.2.5 HTTP Header Fields ... 79
2.2.5.1 Accept ... 80
2.2.5.1.1 application/atom+xml ... 81
2.2.5.1.2 application/json ... 81
2.2.5.2 Content-Type ... 81
2.2.5.3 DataServiceVersion ... 82
2.2.5.4 ETag ... 83
2.2.5.5 If-Match ... 84
2.2.5.6 If-None-Match ... 84
2.2.5.7 MaxDataServiceVersion ... 85
2.2.5.8 X-HTTP-Method ... 86
2.2.6 Common Payload Syntax ... 86
2.2.6.1 Common Serialization Rules for XML-Based Formats ... 87
2.2.6.2 AtomPub Format ... 89
2.2.6.2.1 Entity Set (as an Atom Feed Element) ... 89
2.2.6.2.1.1 Inlinecount Representation (for Collections of Entities) ... 90
2.2.6.2.2 Entity Type (as an Atom Entry Element) ... 91
2.2.6.2.2.1 Entity Type (as an Atom Entry Element) with a Customizable Feed Property Mapping ... 94
2.2.6.2.3 Complex Type ... 94
2.2.6.2.4 Navigation Property ... 95
2.2.6.2.5 EDMSimpleType Property ... 95
2.2.6.2.6 Deferred Content ... 95
2.2.6.2.6.1 Inline Representation ... 95
2.2.6.2.7 Service Document ... 97
2.2.6.2.8 Additional Representations ... 98
2.2.6.3 JavaScript Object Notation (JSON) Format ... 98
2.2.6.3.1 Common JSON Serialization Rules for All EDM Constructs ... 99
2.2.6.3.2 Entity Set (as a JSON Array) ... 101
2.2.6.3.2.1 Inlinecount Representation (for Collections of Entities) ... 102
2.2.6.3.3 Entity Type (as a JSON Object) ... 103
2.2.6.3.4 Complex Type ... 106
2.2.6.3.5 Collection of Complex Type Instances ... 107
2.2.6.3.6 Navigation Property ... 108
2.2.6.3.7 Collection of EDMSimpleType Values ... 108
2.2.6.3.8 EDMSimpleType Property ... 108
2.2.6.3.9 Deferred Content ... 109
2.2.6.3.9.1 Inline Representation ... 110
2.2.6.3.10 Links ... 110
2.2.6.3.11 Inlinecount Representation (for Collections of Links) ... 112
2.2.6.3.12 Service Document ... 112
2.2.6.4 Raw Format ... 113
2.2.6.4.1 EDMSimpleType Property ... 113
2.2.6.5 XML Format ... 113
2.2.6.5.1 Complex Type ... 113
2.2.6.5.2 Collection of Complex Type Instances ... 114
2.2.6.5.3 EDMSimpleType Property ... 114

2.2.6.5.4 Collection of EDMSimpleType Values 114
2.2.6.5.5 Links 114
2.2.6.5.5.1 Inlinecount Representation (for Collections of Links) 115
2.2.7 Request Types 116
2.2.7.1 Insert Request Types 116
2.2.7.1.1 InsertEntity Request 117
2.2.7.1.1.1 Examples 119
2.2.7.1.2 InsertLink Request 124
2.2.7.1.3 InsertMediaResource Request 126
2.2.7.2 Retrieve Request Types 127
2.2.7.2.1 RetrieveEntitySet Request 127
2.2.7.2.2 RetrieveEntity Request 128
2.2.7.2.3 RetrieveComplexType Request 130
2.2.7.2.4 RetrievePrimitiveProperty Request 131
2.2.7.2.5 RetrieveValue Request 132
2.2.7.2.6 RetrieveServiceMetadata Request 133
2.2.7.2.7 RetrieveServiceDocument Request 134
2.2.7.2.8 RetrieveLink Request 136
2.2.7.2.9 RetrieveCount Request 137
2.2.7.2.10 Retrieve Request Containing a Customizable Feed Mapping 138
2.2.7.2.11 RetrieveMediaResource Request 138
2.2.7.3 Update Request Types 139
2.2.7.3.1 UpdateEntity Request 139
2.2.7.3.1.1 Example 141
2.2.7.3.2 UpdateComplexType Request 141
2.2.7.3.3 UpdatePrimitiveProperty Request 142
2.2.7.3.4 UpdateValue Request 143
2.2.7.3.5 UpdateLink Request 145
2.2.7.3.6 UpdateMediaResource Request 146
2.2.7.3.7 Update Request Containing a Customizable Feed Property Mapping 147
2.2.7.4 Delete Request Types 147
2.2.7.4.1 DeleteEntity Request 147
2.2.7.4.2 DeleteLink Request 148
2.2.7.4.3 DeleteValue Request 149
2.2.7.5 Invoke Request 150
2.2.7.6 Batch Request 152
2.2.7.6.1 Change Set Syntax 153
2.2.7.6.1.1 Referencing Requests in a Change Set 154
2.2.7.6.2 Query Operation Syntax 154
2.2.7.6.3 HTTP Request Restrictions 154
2.2.7.6.4 Batch Request Syntax 154
2.2.7.6.5 Example Batch Request 156
2.2.7.6.6 Batch Responses 157
2.2.7.6.7 Batch Response Syntax 158
2.2.7.6.8 Example Batch Response 159
2.2.7.7 Tunneled Requests 161
2.2.8 Response Types 161
2.2.8.1 Error Response 161
2.2.8.1.1 XML Error Response 162
2.2.8.1.2 JSON Error Response 163

3 Protocol Details 165
3.1 Client Details 165

3.1.1 Abstract Data Model 165
3.1.2 Timers 165
3.1.3 Initialization 165
3.1.4 Higher-Layer Triggered Events 165
3.1.4.1 Common Rules for all Requests 165
3.1.4.2 Request to Insert Resources 165
3.1.4.2.1 Sending an InsertEntity Request 166
3.1.4.2.2 Sending an InsertLink Request 166
3.1.4.3 Request to Retrieve Resources 166
3.1.4.3.1 Common Rules for Sending Retrieve Requests 166
3.1.4.4 Request to Update Resources 167
3.1.4.4.1 Common Rules for Sending Update Requests 167
3.1.4.5 Request to Delete Resources 168
3.1.4.5.1 Common Rules for Sending Delete Requests 168
3.1.4.6 Request to Invoke a Service Operation 169
3.1.4.7 Request to Send a Batch of Operations 169
3.1.5 Message Processing Events and Sequencing Rules 169
3.1.5.1 Common Rules for Receiving Responses from Data Service Requests 169
3.1.5.2 Responses from Insert Requests 170
3.1.6 Timer Events 170
3.1.7 Other Local Events 170
3.2 Server Details 170
3.2.1 Abstract Data Model 170
3.2.2 Timers 170
3.2.3 Initialization 171
3.2.4 Higher-Layer Triggered Events 171
3.2.5 Message Processing Events and Sequencing Rules 171
3.2.5.1 Common Rules for Receiving All Data Service Requests 171
3.2.5.2 Common Rules for Executing Received Insert, Update, or Delete Data Service Requests 171
3.2.5.2.1 Common Rules for Executing Requests Containing a Customizable Feeds Mapped Property 172
3.2.5.3 Executing a Received Insert Request 173
3.2.5.3.1 Executing a Received InsertEntity Request 173
3.2.5.3.2 Executing a Received InsertLink Request 174
3.2.5.3.3 Executing a Received InsertMediaResource Request 174
3.2.5.4 Executing a Received Retrieve Request 174
3.2.5.4.1 Executing a Received RetrieveEntitySet Request 175
3.2.5.4.2 Executing a Received RetrieveValue Request 175
3.2.5.4.3 Executing a Received RetrieveCount Request 175
3.2.5.5 Executing a Received Update Request 176
3.2.5.5.1 Executing a Received UpdateEntity Request 177
3.2.5.6 Executing a Received Delete Request 177
3.2.5.7 Executing a Received Invoke Request 177
3.2.5.8 Executing a Received Batch Request 178
3.2.6 Timer Events 178
3.2.7 Other Local Events 178
3.2.8 Common Response Codes 178

4 Protocol Examples 180
4.1 Insert a New Entity 180
4.2 Retrieve Resources 180
4.2.1 Retrieve a Collection of Entities 180

4.2.1.1 Retrieve a Collection of Entities Using the AtomPub Format....180
4.2.1.2 Retrieve a Collection of Entities Using the JSON Format....181
4.2.1.3 Retrieve a Partial Collection of Entities Using the AtomPub Format....182
4.2.1.4 Retrieve a Partial Collection of Entities Using the JSON Format....183
4.2.1.5 Retrieve a Collection of Entities with an Inline Count Using AtomPub Format....184
4.2.1.6 Retrieve a Collection of Entities with an Inline Count Using JSON Format....186
4.2.2 Retrieve a Single Entity Using the AtomPub Format....187
4.2.2.1 Retrieve a Single Entity with a Mapped Property Using the AtomPub Format....188
4.2.3 Retrieve a Single Entity Using the JSON Format....189
4.2.4 Retrieve a Single Entity and Its Directly Related Entities Using the AtomPub Format....189
4.2.5 Retrieve a Single Entity and Its Directly Related Entities Using the JSON Format....191
4.2.6 Retrieve a Data Service's Metadata Document (CSDL)....192
4.2.7 Retrieve the Count of a Collection of Entities....194
4.3 Update an Existing Entity....195
4.3.1 Replace-Based Update Using the AtomPub Format....195
4.3.2 Replace-Based Update Using the JSON Format....197
4.3.3 Merge-Based Update Using the AtomPub Format....198
4.3.4 Merge-Based Update Using the JSON Format....199
4.4 Update the Relationship Between Two Entities....201
4.4.1 Update a Relationship Using the AtomPub Format....201
4.4.2 Update a Relationship Using the JSON Format....201
4.4.3 Delete an Existing Entity....202
4.5 Batch Requests....202
4.6 Working with Media Resources (BLOBs)....202
4.6.1 Insert a new Media Resource....202
4.6.2 Update a Media Resource....203
4.6.3 Query an Existing Media Resource....203

5 Security....205
5.1 Security Considerations for Implementers....205
5.2 Index of Security Parameters....205

6 Appendix A: Sample Entity Data Model and CSDL Document....206

7 Appendix B: Product Behavior....210

8 Change Tracking....214

9 Index....216

1 Introduction

The **Atom Publishing Protocol (AtomPub)**, specified in [RFC5023], is an application-level protocol for publishing and editing Web **resources**. When Web services expose resources via AtomPub, a number of details are not covered by the AtomPub specification, which, if defined, can enable additional client scenarios. These details include:

- An addressing scheme by which resources are identified.
- Schema for the data content of resources.
- Payload formats and semantics for batch requests.
- A model for handling concurrent updates to resources.
- Alternate representations of resource content.

This document defines versions 1.0 and 2.0 of the Open Data protocol (OData) that specify these details.

The protocol defined in this document enables applications to expose data, using common web technologies, as a **data service** that can be consumed by clients within corporate networks and across the Internet.

Version 2.0 of the protocol defined by this specification is a superset of version 1.0 and includes incremental additions to version 1.0. Any constructs or semantics defined in this document that only exist in version 2.0 of the protocol defined by this specification are explicitly denoted as such.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

.NET Framework
binary large object (BLOB)
URI
XML Namespace

The following terms are defined in [MC-CSDL]:

alias
annotation
association
cardinality
collection
declared property
dynamic property
Entity Data Model (EDM)
facet
identifier
namespace
schema

The following terms are specific to this document:

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

AtomPub element: Any XML element defined in either "The Atom Publishing Protocol" [RFC5023] or "The Atom Syndication Format" [RFC4287].

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

bind: An EntityType instance in a **data service** (described using **Entity Data Model** constructs) that may be related to one or more other **CSDL** instances. These relationships are represented using **associations** in an **Entity Data Model**. The **cardinality** of a relationship can be determined by inspecting the **Entity Data Model** 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, then they are considered to be "bound".

Change Set: A logical group 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 Type (section 2.2.7.4)
- Invoke Request (section 2.2.7.5), which may be created using the **HTTP PUT**, **POST**, or **HTTP DELETE** method

All of the requests within a **Change Set** must be successfully processed. If any request in the **Change Set** fails, then none of the requests within the **Change Set** should be processed.

create retrieve update delete (CRUD): This term, typically represented using the abbreviation **CRUD**, is used to denote 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. This term is used to denote these conceptual actions and does not imply the associated meaning in a particular technology area (for example, databases, file systems, and so on) unless explicitly stated.

Customizable Feed: 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 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 protocol specified in this document for the purpose of enabling clients to publish and edit **resources**. The **resources** exposed by **data services** are described using the **Entity Data Model (EDM)**, as specified in [MC-CSDL].

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, as described in URI Format: Resource Addressing Rules (section 2.2.3).

empty value: A value that is NULL or nothing.

entity: An instance of an EntityType, as specified in [MC-CSDL].

Entity Data Model Extensions (EDMX): The Entity Data Model Extensions, as specified in [MC-EDMX].

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

JavaScript Object Notation (JSON): This term refers to a small set of formatting rules for the portable representation of structured data, as specified in [RFC4627].

Link: A **Link** is similar to an **association**, as specified in [MC-CSDL], but describes a unidirectional relationship between EntityTypes instead of a bidirectional one. A **Link** can be:

- A unidirectional relationship (for example, a **Link**), which occurs when two EntityTypes are related via an **association**, but only one of the EntityTypes defines a NavigationProperty bound to the **association**.
- A reference to one direction of a bidirectional **association** between two EntityTypes, as specified in [MC-CSDL].

Primitive Property: A property of type EDMSimpleType 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 may be a **declared property** or a **dynamic property**, as specified in [MC-CSDL]. A **property** of ComplexType must be a **declared property**.

Note In **CSDL** [MC-CSDL], **dynamic properties** are only defined for use with OpenEntityType [MC-CSDL] instances.

Query Operation: A logical construct that must consist of a single Retrieve Request Types (section 2.2.7.2) or an Invoke Request (section 2.2.7.5) which uses the **GET HTTP** method.

Resource: A network-accessible data object or service 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, as described in Resource Path (section 2.2.3.3).

Service Operation: Represents a FunctionImport, as specified in [MC-CSDL], which only accepts input parameters in a **data service**.

Service Root: A **URI** that represents the root of a **data service**, as described in Service Root (section 2.2.3.2).

Unbind: See the definition for **"bind"**.

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

1.2 References

References to Microsoft Open Specification documents do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[ECMA-262] ECMA International, "ECMAScript Language Specification" ECMA-262, December 1999, http://www.ecma-international.org/publications/standards/Ecma-262.htm

[IANA-MMT] Internet Assigned Numbers Authority, "Mime Media Types", March 2007, http://www.iana.org/assignments/media-types/

[IEEE754-2008] Institute of Electrical and Electronics Engineers, "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

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

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

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

[RFC2046] Freed, N., and Borenstein, N., "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996, http://ietf.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.ietf.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.ietf.org/rfc/rfc2617.txt

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

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003, http://www.ietf.org/rfc/rfc3548.txt

[RFC3629] Yergeau, F., "UTF-8, A Transformation Format of ISO 10646", STD 63, RFC 3629, November 2003, http://www.ietf.org/rfc/rfc3629.txt

[RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004, http://www.ietf.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.org/rfc/rfc3986.txt

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

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

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

[RFC4646] A. Phillips, Ed., and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 4646, September 2006, http://www.ietf.org/rfc/rfc4646.txt

[RFC5023] Gregorio, J. Ed., and de hOra, B., Ed., "The Atom Publishing Protocol", RFC 5023, October 2007, http://www.ietf.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

[XML-BASE] World Wide Web Consortium, "XML Base", June 2001, http://www.w3.org/TR/2001/REC-xmlbase-20010627/

[XMLNS] World Wide Web Consortium, "Namespaces in XML 1.0 (Second Edition)", August 2006, http://www.w3.org/TR/2006/REC-xml-names-20060816/

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSCHEMA2/2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes Second Edition", W3C Recommendation, October 2004, http://www.w3.org/TR/xmlschema-2

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

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

The Atom Publishing Protocol 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 protocol described in this document defines a uniform, HTTP-based interface for data services that address these shortcomings of the Atom Publishing Protocol. 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 protocol defined in this document 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 **entity** that initiates the HTTP connection and sends HTTP request messages is referred to as the client, and 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 need to operate against data that is stored across Internet trust boundaries.

1.4 Relationship to Other Protocols

This document defines versions 1.0 and 2.0 of the Open Data protocol, which in turn relies on HTTP [RFC2616]. Either HTTP version 1.1 or HTTP version 1.0 may be used. The protocol uses HTTP headers defined in the HTTP specification.

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

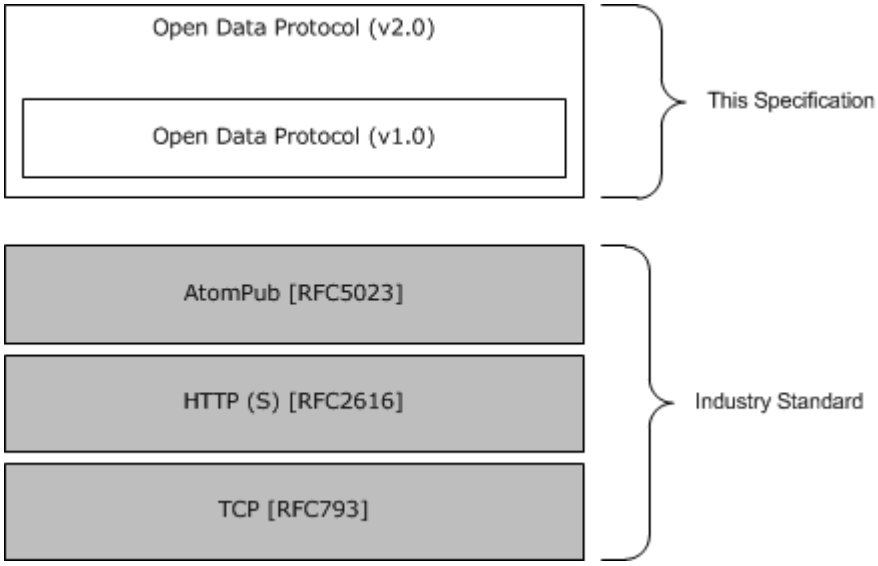

Figure 1: OData Relationship to Other Protocols

1.5 Prerequisites/Preconditions

The protocol defined in this document 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 nor the protocol defined in this document define an authentication or authorization scheme. Implementers of this protocol should 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

The protocol defined in this document is appropriate for use in Web services which 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 described in Transport (section 2.1).

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

Security and Authentication Methods: This document supports (but does not require) any authentication scheme which can be supported 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 protocol 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 protocol defined in this document.

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

If present on 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, then the server must assume a value equal to the maximum version number the server supports.

If present on 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 should set 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 must assume 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, then the server should assume the client can interpret the maximum version number the server can interpret.

When the server receives a request, it must validate 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, then the server must return a response with a 4xx response code, as described in [RFC2616]. The server should also return a description of the error using the error format defined in Error Response (section 2.2.8.1).

In addition, a server must validate 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, then the server must return an error response, described in Error Response (section 2.2.8.1).

On a response from the server to the client, the DataServiceVersion (section 2.2.5.3) header should be specified. The value states the version of the protocol the server used to generate the request and that should be 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 header should be the lowest version of the protocol the server can use to fulfill the request.

1.7.1 Version 2.0 Summary

As described in section 1.4, this document defines two versions (1.0 and 2.0) of the protocol defined by this specification. This section provides a summary of the protocol constructs defined in this document, which apply only to version 2.0. This section is structured by protocol feature. Each protocol feature described includes a list of the sections which include version 2.0-specific content.

Partial Sets of Entities: Servers may respond to RetrieveEntitySet (section 2.2.7.2.1) GET requests with a response body containing a representation of a partial list of the entities 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)
- 3.2.5.4.1 Executing a Received RetrieveEntitySet Request
- 2.2.6.2.1 Entity Set (as an Atom Feed Element)
- 2.2.6.3.2 Entity Set (as a JSON array)

RetrieveCount Request: The purpose of the RetrieveCount Request (section 2.2.7.2.9) 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
- 2.2.7.2.9 RetrieveCount Request
- 3.1.4.3.1 Common Rules for Sending Retrieve Requests
- 3.2.5.4.3 Executing a Received RetrieveCount Request
- 4.2.7 Retrieve the Count of a Collection of Entities

Inlinecount System Query Option: A data service URI with an Inlinecount System Query Option specifies that the response to the request must include the count N of the total number of entities in the EntitySet ([MC-CSDL] section 2.1.17), 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)
- 2.2.6.3.10 Links
- 2.2.6.3.11 Inlinecount Representation (for Collections of Links)
- 3.2.5.4 Executing a Received Retrieve Request
- 4.2.1.5 Retrieve a Collection of Entities with an Inline Count Using AtomPub Format
- 4.2.1.6 Retrieve a Collection of Entities with an Inline Count Using JSON Format

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 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.

- 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: The **Customizable Feed** property mappings can be used to override an Entity Types default AtomPub representation and specify how one or more properties of an Entity Type should 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 for Version 2.0 Data Services
- 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
- 2.2.7.2.10 Retrieve Request Containing a Customizable Feed Mapping
- 2.2.7.3.1 UpdateEntity Request
- 2.2.7.3.7 Update Request Containing a Customizable Feed Mapping
- 3.2.5.2.1 Common Rules for Executing Requests Containing a Customizable Feeds Mapped Property

- 4.2.2.1 Retrieve a Single Entity with a Mapped Property Using the AtomPub Format
- 4.2.6 Retrieve a Data Service's Metadata Document (CSDL)
- 6 Appendix A: Sample Entity Data Model and CSDL Document

New JSON Response Format: The 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

1.8 Vendor-Extensible Fields

The AtomPub-based messages defined in Messages (section 2) may be extended by adding additional elements or attributes. Such extensions MUST NOT be in any of the **namespaces** 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 Atom Publishing Protocol (AtomPub) [RFC5023] and the protocol defined in this specification 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, etc) 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 which serve as a basis for the protocol defined in this document. The subsequent sections each define mappings from the data model to the protocol defined.

Abstract Type System (section 2.2.2) specifies the Abstract Type System used to define the **Primitive** types (such as String, Boolean, etc.) used by the protocol defined in this document.

URI Format: Resource Addressing Rules in Abstract Type System (section 2.2.2) specifies a set of rules for constructing URIs which identify each of the constructs in the data model, described in Abstract Data Model (section 2.2.1), which are relevant to the protocol defined in this document.

HTTP Header Fields, as described in HTTP Header Fields (section 2.2.5), specify the syntax for the HTTP headers defined or used by this document.

HTTP Payload Format Syntax in Common Payload Syntax (section 2.2.6) specifies how data described using the abstract data model in Abstract Data Model (section 2.2.1) is mapped to the AtomPub and JSON for use in the payloads of the HTTP request types described in Request Types (section 2.2.7).

Request Types (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 constructs in the data model in Abstract Data Model (section 2.2.1).

NOTE: All the example URIs and message payloads used in this section as 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 MUST use 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 protocol described in this document.

The protocol defined in this document uses the **Entity Data Model** as its data modeling vocabulary. Data models can be described in Entity Data Model terms using a conceptual schema definition language (CSDL) document [MC-CSDL]. The remainder of this section provides a brief description of the Entity Data Model 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 EntityTypes (for example, Customer, Employee, etc) which are structured records consisting of named and typed properties and with a key. ComplexTypes ([MC-CSDL] section 2.1.7) are structured types also consisting of a list of properties but with no key, thus can only exist as a property of a containing entity or as a temporary value. An EntityKey ([MC-CSDL] section 2.1.5) is formed from a subset of properties of the EntityType. The EntityKey (for example, CustomerId, OrderId, and so on) is a fundamental concept to uniquely identify instances of EntityTypes and allows EntityType instances to participate in relationships. Entities are grouped in EntitySets (for example, Customers is a set of Customer instances). Associations define the relationship between two or more EntityTypes (for example, Employee Works for Department). Instances of associations are grouped in AssociationSets ([MC-CSDL] section 2.1.18). NavigationProperties ([MC-CSDL] section 2.1.4) are special properties on EntityTypes which 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 (EntitySets and AssociationSets) are grouped in an EntityContainer ([MC-CSDL] section 2.1.14).

Entity Data Model 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 instance	Entry Resource
NavigationProperty	<atom:link> element

Table: Entity Data Model Concepts Mapped to AtomPub Resource Types

2.2.2 Abstract Type System

The Abstract Type System used to define the **Primitive** types 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 it should use the "Primitive Type Literal Form" representation defined in the following Literal Form of Entity Data Model Primitive Types table. **Primitive** type representations in request and response payloads are defined in format-specific sections of this document.

The Shared ABNF [RFC5234] Grammar Rules for Primitive Types listing that follows the grammar rules in this section makes reference to the following shared ABNF [RFC5234] grammar rules.

```
SQUOTE            =  %x27             ; ' (single quote)
nonZeroDigit      =  %x31-30          ; all digits except zero
doubleZeroToSixty =  "0" DIGIT
                     / "1" DIGIT
                     / "2" DIGIT
                     / "3" DIGIT
                     / "4" DIGIT
```

```
                    / "5" DIGIT
                    / "6" DIGIT
nan                 =  "Nan"
negativeInfinity    =  "-INF"
postiveInfinity     =  "INF"
sign                =  "-" / ""
DIGIT               =  ; see [RFC5234] Appendix A
UTF8-char           =  ; see [RFC3629]
```

Listing: Shared ABNF Grammar Rules for 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 binary is not`
		
		`hexDigPair = 2*HEXDIG [hexDigPair]`

Edm.Boolean	booleanLiteral	`booleanLiteral = true / false`
		
		`true = "true" / "1"`
		
		`false = "false" / "0"`

Edm.Byte	byteLiteral	`byteLiteral = 1*3DIGIT;`
		
		`; For further information on the value range for`
		`; the Edm.Byte type see <MSCSDL>`

Edm.DateTime	dateTimeUriLiteral	`datetimeUriLiteral = "datetime" SQUOTE dateTimeLiteral SQUOTE`
		
		`dateTimeLiteral = year "-" month "-" day "T" hour ":" minute [":" second ["."`

EDM Primitive Type	ABNF Rule for Primitive Type Representation in URIs and HTTP Headers	Primitive Type Literal Form (ABNF Definition)
		nanoSeconds]] year = 4 *Digit; month = <any number between 1 and 12 inclusive> day = nonZeroDigit /("1" DIGIT) / ("2" DIGIT) / "3" ("0" / "1") hour = nonZeroDigit / ("1" DIGIT) / ("2" zeroToFour) zeroToFour= <any nuumber between 0 and 4 inclusive> minute =doubleZeroToSixty second = doubleZeroToSixty nanoSeconds= 1*7Digit
Edm.Decimal	decimalUriLiteral	decimalUriLiteral = decimalLiteral ("M"/"m") decimalLiteral = sign 1*29DIGIT ["." 1*29DIGIT]
Edm.Double	doubleLiteral	doubleLiteral = nonDecimalPoint / nonExp / exp / nan / negativeInfinity / postiveInfinity ("D" / "d") nonDecimalPoint= sign 1*17DIGIT nonExpDecimal = sign* DIGIT "." *DIGIT expDecimal = sign 1*DIGIT "." 16DIGIT ("e" / "E") sign 1*3DIGIT ; for additional information on the value range ; of the Edm.Double type, see [MC-CSDL]

EDM Primitive Type	ABNF Rule for Primitive Type Representation in URIs and HTTP Headers	Primitive Type Literal Form (ABNF Definition)

Edm.Single	singleUriLiteral	`singleUriLiteral = singleLiteral ("F" / "f")`
		`singleLiteral = nonDecimalPoint / nonExp / exp / nan / negativeInfinity / postiveInfinity`
		
		`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.Guid	guidUriLiteral	`guidUriLiteral= "guid" SQUOTE guidLiteral SQUOTE`
		
		`guidLiteral = 8*HEXDIG "-" 4*HEXDIG "-" 4*HEXDIG "-" 12*HEXDIG`

Edm.Int16	int16Literal	`int16Literal= sign 1*5DIGIT`
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 Primitive Type	ABNF Rule for Primitive Type Representation in URIs and HTTP Headers	Primitive Type Literal Form (ABNF Definition)

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 duration in [XMLSCHEMA2/2]>`

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

Table: Literal Form of Entity Data Model Primitive Types

2.2.3 URI Format: Resource Addressing Rules

The Atom Publishing Protocol specifies operations for publishing and editing resources 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, described using a conceptual schema definition language (CSDL) document, to the resource types defined in section 4.2 of [RFC5023]. 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 (IRI)** embedded in those representations; it 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 Entity Data Model.

Servers and clients can use alternate URI path construction rules as HTTP [RFC2616] specifies that the URI space of each server is controlled by that server. The protocol defined in this specification imposes no further constraints on that control when constructing the authority and path segments of a URI. However, servers conforming 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) defined in this specification when possible, as 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 which has been converted to a URI.

2.2.3.1 URI Syntax

The Augmented BNF for URI Construction listing that follows 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 ComplexType in an Entity Data Model.

Servers that conform to this specification can follow the grammar below when constructing the scheme, Service Root, and Resource Path URI components of a data service URI. All servers MUST follow the grammar rules shown 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 ] [ navPath ] [ count ])
                      / serviceOperation

paren               = "()"

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

count               = "/$count"
```

```
      ; count is supported only in version 2.0 of the protocol defined by this
      ; specification

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

  navPath-options     = [ navPath-np / propertyPath / propertyPath-ct / value ]

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

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

  propertyPath        = "/" entityProperty [ value ]

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

  keyPredicate        = keyPredicate-single
                        / keyPredicate-cmplx

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

  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
              *("&"(sysQueryOption / serviceOpParam
              / customQueryOption))

  sysQueryOption    = expandQueryOp
                    / filterQueryOp
                    / orderbyQueryOp
                    / skipQueryOp
                    / topQueryOp
                    / formatQueryOp
                    / countQueryOp
                    / selectQueryOp
                    / skiptokenQueryOp

  customQueryOption   = *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

  serviceOpParam      = ; 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 version 2.0 of the
     ; protocol defined by this specification

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     = *pchar   ; section 3.3 of [RFC3986]
     ; 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

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

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.

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.
```

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 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), which 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 the `http://` scheme be used to address elements on an Entity Data Model.

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 EntitySets, EntityTypes, entity NavigationProperties, Members, and **Service Operations** may be composed to generate a URI that identifies a resource exposed by a data service.

Using the example Entity Data Model in Appendix A: Sample Entity Data Model and CSDL Document (section 6), example URIs including 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 which can be constructed 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).

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

entitySet: The name of an EntitySet in the EDM model associated with the data service. EntitySet names MAY<2> be directly followed by open and close parenthesis (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**, then 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 model associated with the data service. The EntityType identified can be an OpenEntityType ([MC-CSDL] section 2.2.8).

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 model 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 model 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.

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", then 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, there must not exist a path segment after 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.

keyPredicate: Identifies an EntityKey.

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

The Entity Data Model defines that each such key value MUST be non-nullable, immutable, and be an EDMSimpleType. The representation of an EDMSimpleType value in a data service URI MUST 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 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-collEt: Identifies a FunctionImport ([MC-CSDL] section 2.1.15) in an Entity Data Model, as seen in [MC-CSDL], which returns a collection of entities with each entity 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 may follow identifying entities or relationships on entities within the collection identified by the Service Operation.

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

- **Primitive** type
- collection of **Primitive** types
- a single ComplexType instance
- collection of ComplexType instances
- a single EntityType instance

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

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 defined in Appendix A: Sample Entity Data Model and CSDL Document.

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 Entity Data Model associated with the data service does not include an EntitySet with the name specified, then 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, then 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 may 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 may 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** may define a NavigationProperty named "OrderedBy" that represents the **Customer** instance associated with that particular **Order** instance. Similarly, the **Customer Entity Type** may 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 may 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 Entity Data Model 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 which 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 Entity Data Model associated with the data service which has the same name as specified by the serviceOperation-et rule, then 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 which returns a collection of ComplexType instances.

If no FunctionImport exists in the Entity Data Model associated with the data service that has the same name as specified by the serviceOperation-collCt rule, then 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 which returns a ComplexType instance.

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

A path segment containing only the rule serviceOperation-prim may 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 Entity Data Model associated with the data service does not include an EntitySet with the name specified, then 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 version 2.0 of the protocol defined by this specification.

- `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 Entity Data Model associated with the data service does not include an EntitySet instance with the keyPredicate specified, then 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 $value segment is supported only in version 2.0 of the protocol defined by this specification.

- `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
```

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 "$", as seen in System Query Option (section 2.2.3.6.1).
- Custom Query Options (section 2.2.3.6.2) MUST NOT begin with a "$".

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, 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 Supported System Query Options 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, then the URI MUST be considered malformed.

System Query Option	Description	Additional Details
$expand	This option indicates entities associated with the EntityType instance or EntitySet, identified by the Resource Path section of the URI, and MUST be represented inline in the data service's response, as opposed to being represented with Deferred Content markers in Deferred Content (section 2.2.6.2.6) and Deferred Content (section 2.2.6.3.9).	See Expand System Query Option ($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).

System Query Option	Description	Additional Details
$orderby	This option specifies the sort properties and sort direction (ascending or descending) that the data service MUST use to order the entities in the EntitySet, identified by the Resource Path section of the URI.	See OrderBy System Query Option ($orderby) (section 2.2.3.6.1.6).
$format	This option specifies the media type acceptable in a response. If present, this value SHOULD take precedence over value(s) specified in an Accept (section 2.2.5.1) request header.	See Format System Query Option ($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 $orderby option, that the data service should skip when returning the entities in the EntitySet, which is identified by the Resource Path section of the URI. The data service SHOULD return all subsequent entities, starting from the one in position N+1.	See Skip System Query Option ($skip) (section 2.2.3.6.1.7).
$top	This option specifies a positive integer N that is the maximum number of entities in the EntitySet, identified by the Resource Path section of the URI, that the data service MUST return.	See Top System Query Option ($top) (section 2.2.3.6.1.8).
$skiptoken	This query option applies only to version 2.0 of the protocol defined by this specification. 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 MUST include the count of the number of entities in the EntitySet, identified by the Resource Path section of the URI after all $filter System Query Options have been applied. For a value of "none", this option indicates that the response to the request MUST NOT include the count value.	See Inlinecount System Query Option ($inlinecount) (section 2.2.3.6.1.10)
$select	This option is used to specify that a subset of the properties of the entities identified by the path of the request URI and $expand query option SHOULD 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 System Query Options Supported per URI table, the column 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 can be used with the URI type associated with the column. A cell value of "no" indicates that if the System Query Option is present on a URI of the form indicated by the associated column, the URI MUST be considered malformed.

	URI1	URI2	URI3	URI4	URI5	URI6 (Note1)	URI6 (Note2)	URI7	URI8	URI9	URI10	URI11	URI12	URI13	URI14	URI15	URI16	URI17
$expand	yes	yes	no	no	no	yes	yes	no	no	no	no	no	no	no	no	yes	yes	no
$filter	yes	yes	yes	no	no	yes	yes	no	no	no	no	no	no	no	no	yes	yes	no
$format	yes	yes	yes	yes	yes	yes	yes	yes	no	no	yes	yes	yes	yes	yes	no	no	yes
$orderby	yes	no	no	no	no	no	yes	no	no	no	no	no	no	no	no	yes	no	no
$skip	yes	no	no	no	no	no	yes	yes	no	no	no	no	no	no	no	yes	no	no
$top	yes	no	no	no	no	no	yes	yes	no	no	no	no	no	no	no	yes	no	no
$skiptoken	yes	no	no	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	no
$inlinecount (Note 3)	yes	no	no	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	no
$select	yes	yes	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	no	no

Figure 2: Table: System Query Options Supported Per URI

Note 1: The NavigationProperty in the final path segment of the URI should be assumed to identify a single EntityType instance.

Note 2: The NavigationProperty in the final path segment of the URI should be assumed to identify a set of entities.

Note 3: The $inlinecount System Query Option is supported only in version 2.0 of the protocol defined by this specification.

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) [WSP]

boolCommonExpression = [WSP] (boolLiteralExpression / andExpression /
              orExpression /
              boolPrimitiveMemberExpression / eqExpression / neExpression /
              ltExpression / leExpression / gtExpression /
              geExpression / notExpression / isofExpression/
              boolCastExpression / boolMethodCallExpression  /
              firstBoolPrimitiveMemberExpression / boolParenExpression) [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]
                 stringLiteral [WSP] ")"

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

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

firstMemberExpression = [WSP] entityNavProperty /
                        ; section 2.2.3.1
                        entityComplexProperty /
                        ; section 2.2.3.1
                        entitySimpleProperty
                        ; section 2.2.3.1

firstBoolPrimitiveMemberExpression = entityProperty
                                     ; section 2.2.3.1

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

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

literalExpression = stringLiteral ; section 2.2.2
          / dateTimeLiteral ; 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

boolLiteralExpression = boolLiteral ; section 2.2.2

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

boolMethodExpression = endsWithMethodCallExpression
                     / startsWithMethodCallExpression
                     / substringOfMethodCallExpression

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] ] ")"
```

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

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

lengthMethodCallExpression = "length" [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] ")"
```

Listing: Augmented BNF for Query Option Expressions

A data service MAY<7> support some or all of the boolCommonExpressions for the filter ($filter) system query option. A data service MAY<8> support some or all of the commonExpressions for the orderby ($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 can 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 (section 2.2.3.6.1.1) for each specific expression type.

2.2.3.6.1.1.1 Expression Construction and Evaluation Rules

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

boolCommonExpression: A data service can support the boolCommonExpression common expression. If supported, a boolCommonExpression MUST be a common expression that evaluates to the EDM **Primitive** type **Edm.Boolean**.

parenExpression: A data service can 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 can 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 can 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.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 can 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 can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

mulExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

divExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

modExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

negateExpression: A data service can 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.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 Operator Precedence (section 2.2.3.6.1.1.2) 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 can support evaluating an operand with a null value following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

andExpression: A data service can 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 boolCommonExpressions 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

orExpression: A data service can 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 boolCommonExpressions 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

memberExpression: A data service can 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 which is the target of the memberExpression MUST be a known **Edm Entity** or **ComplexType**. If supported, the memberExpression can reference an entity NavigationProperty (entityNavProperty, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an **Entity Complex** type property (entityComplextProperty, 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 can 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 can support the referencing of a Navigation, Complex, or Simple Property of the **Edm Entity** 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 memberExpression can reference an **Entity Navigation** property (entityNavProperty, as specified in Resource Path: Construction Rules (section 2.2.3.4)), or an Entity Complex type property (entityComplextProperty, 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 can 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).

boolPrimitiveMemberExpression: A data service can support the referencing of a Boolean Simple Property of an **Edm Entity** 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 can 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 can support the referencing of a Boolean Simple Property of an **Edm Entity** 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 can 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 can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

neExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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, 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 can support evaluating operands with null values following the rules defined in Binary numeric promotions (section 2.2.3.6.1.1.4).

ltExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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, 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 can support evaluating operands with null values following the rules defined in Binary numeric promotions (section 2.2.3.6.1.1.4).

leExpression: A data service can 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.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 Unary numeric promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

gtExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

geExpression: A data service can 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.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 Unary Numeric Promotions (section 2.2.3.6.1.1.3) 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 can support evaluating operands with null values following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

notExpression: A data service can 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 can support evaluating an operand with a null value following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

isofExpression: A data service can 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 can 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 stringLiteral 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 can be converted to the specified type. If the conversion is not allowed, then the expression MUST be evaluated to false.

The data service can support evaluating an operand with a null value following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

castExpression: A data service can 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 can 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 stringLiteral which 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 can 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 can support evaluating an operand with a null value following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

boolCastExpression: A data service can 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 can 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 stringLiteral "Edm.Boolean".

The type of the result of evaluating the boolCastExpression MUST be EDM **Primitive** type **Edm.Boolean**. A data service can 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 can support evaluating an operand with a null value following the rules defined in Binary Numeric Promotions (section 2.2.3.6.1.1.4).

boolLiteralExpression: A data service can support expressions which 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 can support expressions which 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 can support the methodCallExpression common expression. If supported, a methodCallExpression MUST represent a method call in the common expression syntax.

boolMethodCallExpression: A data service can 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 can 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 can 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 can 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 can 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 can 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 can 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 can 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 can 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 SHOULD be the length (in characters) of the third parameter value. Otherwise, the entire string from the specified starting index is returned.

subStringOfMethodCallExpression: A data service can 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 second parameter string value occurs in the first 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 can 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 which is the first and second parameter values merged together with the first parameter value coming first in the result.

lengthMethodCallExpression: A data service can 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.

yearMethodCallExpression: A data service can 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<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.DateTime**.

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 can 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<39> 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.DateTime**.

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 can 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<40> 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.DateTime**.

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 can 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<41> 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.DateTime**.

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 can 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<42> 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.DateTime**.

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 can 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<43> 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.DateTime**.

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 can 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<44> 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 can 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<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 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 can 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<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 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.

2.2.3.6.1.1.2 Operator Precedence

The following Operator Precedence for Query Option Expressions 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

Category	Expression	Common Expression
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
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<47> 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 can 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 **Edm.Single** to **Edm.Double**.

2.2.3.6.1.1.4 Binary Numeric Promotions

A data service can 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

Operation	Common Expression
Less Than	ltExpression
Greater Than or Equal	geExpression
Less Than or Equal	leExpression

Table: Operations Which Support Binary Numeric Promotion

If supported, binary numeric promotion SHOULD implicitly convert both operands to a common type and, in the case of the non-relational operators, also become the return type.

If supported, a data service SHOULD support binary numeric promotion for the following 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 can support the allowance of operators that operate on 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 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 value null if either operand value is null.
- 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 can support numeric promotions for method call parameters.

If supported, a data service SHOULD support binary numeric promotions for the following 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 the method parameter is of type **Edm.Decimal**, the parameter value expression is converted to **Edm.Decimal**, unless it is of type **Edm.Single** or **Edm.Double**.
- Otherwise, if the method parameter is **Edm.Double**, the parameter value expression is converted to type **Edm.Double**.
- Otherwise, if the method parameter is **Edm.Single**, the parameter value expression is converted to type **Edm.Single**.
- Otherwise, if the method parameter is **Edm.Int64**, the parameter value expression is converted to type **Edm.Int64**.
- Otherwise, if the method parameter is **Edm.Int32**, the parameter value expression is converted to type **Edm.Int32**
- Otherwise, if the method parameter is **Edm.Int16**, the parameter value expression is converted to type **Edm.Int16**.
- If numeric promotion for method calls is supported, a data service SHOULD use a castExpression to promote a parameter value expression to the target type.

2.2.3.6.1.2 Evaluating System Query Options

Any combination of the System Query Options defined in this document can 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 resource(s) identified by the Resource Path section of the URI, in the following order: $format, $inlinecount, $filter, $orderby, $skiptoken, $skip, $top, $expand.

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 entity 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 following rules supplement the grammar below, which represents the syntax of this System Query Option.

```
expandQueryOp = "$expand=" expandClause *("," expandClause)
expandClause  = entityNavProperty *("/" entityNavProperty)
              ; section 2.2.3.1
```

The left most entityNavProperty in an expandClause MUST represent a NavigationProperty defined in the EntityType, or a sub type 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 sub type thereof, represented by the prior NavigationProperty in the expandClause.

Redundant expandClause rules on the same data service URI can 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 should be represented inline.

```
http://host/service.svc/Orders?$expand=OrderLines/Product,Customer
```

For each Order within the Orders EntitySet, the following should be 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.

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 = Customers/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).

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="
                ("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", then the media type used in the response MUST be "application/atom+xml".
- If the value of the query option is "json", then the media type used in the response MUST be "application/json".
- If the value of the query option is "xml", then the media type used in the response MUST be "application/xml".

Examples

```
http://host/service.svc/Orders?$format=json
```

The set of Order entities represented using the JSON media type, as specified in [RFC4627].

The $format query option can 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=example
```

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 equal or 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).

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 should be considered 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 should be considered 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*DIGIT
```

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)

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 can 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 version 2.0 of the rpotocol defined by this specification.

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 4xx error response code.
- If a value of "none" is specified, the data service MUST NOT include the count in the response.

For information on including the count in the AtomPub and JSON Serialization Formats, refer to sections 2.2.6.2.8 and 2.2.6.3.9.1.

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
```

The first 10 order instances and the count of all order instances returned.

```
http://host/service.svc/Orders?$inlinecount=none&$top=10
```

The first 10 order instances and the count of all order instances returned.

```
http://host/service.svc/Orders?$inlinecount=allpages&$filter=ShipCountry eq 'France'
```

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)

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 can 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 version 2.0 of the protocol defined by this specification.

```
selectQueryOp  = "$select=" selectClause
selectClause   = [WSP] selectItem [[WSP] "," selectClause] [WSP]
selectItem     = star / selectedProperty / (selectedNavProperty ["/" selectItem])
selectedProperty = entityProperty / entityComplexProperty
selectedNavProperty = entityNavProperty-es / entityNavProperty-et
star           = "*"
```

- The left most selectedProperty or selectedNavProperty in a selectedClause MUST be a star or represent a property 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: 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, then 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 selectedItem 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 can be considered valid, but MUST NOT alter the meaning of the URI.

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 reference all properties of the entity or collection of entities identified by the path of the URI or all properties of a navigation property. In other words, the * syntax causes all properties on an entity to be included without traversing associations.

```
http://host/service.svc/$select=CustomerID,Orders/*&$expand=Orders/OrderDetails
```

In a response from a data service, the CustomerID is included as are Order entities with all properties; however, rather than including the fully expanded OrderDetail entities referenced in the expand clause, each Order will contain a link that references the corresponding collection of Order Detail entities.

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 Function Import 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 may 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

2.2.3.7 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 constructs are represented in an AtomPub Service Document. As per [RFC5023], AtomPub Service Documents MUST be identified with the "application/atomsvc+xml" media type (see [RFC5023] section 8).

Service Document (section 2.2.6.3.12) describes a JSON representation of the data provided by an AtomPub Service Document. The section also describes how Entity Data Model constructs are represented in the JSON-based Service Document. JSON Service Documents MUST be identified using the "application/json" media type.

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 conceptual schema definition language (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 which contains one or more conceptual schema definition language (CSDL) documents, as specified in [MC-CSDL], with data service **annotations**. The data service conceptual schema definition language (CSDL) annotations are described and highlighted in the XML schema below, as specified in [XMLSCHEMA1]. 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 conceptual schema definition language (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 only
 supported in version 2.0 of the protocol defined by this specification (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: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: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 only supported in version 2.0 of the protocol defined by this specification (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 only supported in version 2.0 of the protocol defined by this
         specification  (see section 2.2.3.7.2.1)  -->
      <xs:attributeGroup ref="m: TWebCustomizableFeedsAttributes" />
   </xs:complexType>
 </xs:schema>
```

Listing: Conceptual Schema Definition Language Document XSD Schema with Data Service Annotations

The following listing describes the extensions to conceptual schema definition language (CSDL) [MC-CSDL] defined by this document (and shown in the XML schema preceding).

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 conceptual schema definition language (CSDL) document used to describe a data service MUST mark exactly one EntityContainer with this attribute to denote it is the default.

The valid values for this attribute are "true" or "false". True signifies the container is the default container and thus does not need to be specified in a Resource Path (section 2.2.3.3). False signifies the container is not the default and needs to be specified in the URI according to the URI construction rules noted in section 2.2.3.4.

MimeType: This attribute MUST be used on a <Property> element [MC-CSDL] to indicate the media type of the content to be stored in the property being defined by the XML element. Each <Property> element defining an EDMSimpleType property MAY<48> 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, then any RetreiveValue Request (RetrieveValue Request (section 2.2.7.2.5)) for the property MUST return a response which 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] to indicate the **HTTP** method which is to be used to invoke the ServiceOperation exposing the FunctionImport. If this attribute is present, the FunctionImport must be callable using the **HTTP** method specified.

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 conceptual schema definition language (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' conceptual schema definition language (CSDL) document. The "Name" attribute on the element MUST be equal to the name of the Service Operation 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 entity data model for data services packaging format [MC-EDMX] 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 a <edmx:DataServices> element [MC-EDMX] to indicate the version of the data service CSDL annotations (attributes in the data service metadata namespace) used by the document. Consumers of a data-service metadata endpoint SHOULD 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.

In the absence of a DataServiceVersion, consumers of the Conceptual Schema Definition Language Document SHOULD assume the highest DataServiceVersion they can handle.

2.2.3.7.2.1 Conceptual Schema Definition Language Document for Version 2.0 Data Services

This section defines version 2.0 specific extensions (shown in the XML Schema below) to the data service specific metadata document defined in the prior section. These attributes define Customizable Feed property mappings for the AtomPub Format.

```
<?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:attributeGroup>
</xs:schema>
```

Listing: Conceptual Schema Definition Language Document XSD Schema for Version 2.0 Data Services

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 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 Using the AtomPub Format (section 4.2.2.1).

The following example shows a Conceptual Schema Definition Langue (CSDL) definition of an Entity Type which includes Customizable Feed property mappings. The Customizable Feed annotations defined on the Entity Type maps the EmployeeName property of the Employee Entity Type to the <atom:title> element in the Atom namespace. The example also includes a mapping that specifies the City property of the Address Complex Type property to be represented 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 attributes on the Property element of an Entity Type (as in the above 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 conceptual schema definition language (CSDL) [MC-CSDL] defined by this section (and shown in the XML schema preceding). These extensions are only supported in version 2.0 of the protocol defined by this specification.

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:

```
FC_ContentKind = "text " \ "html" \ "xhtml"
```

If the FC_ContentKind property is not defined for an EntityType property, the value of the property should be assumed to be "text ".

FC_KeepInContent: The FC_KeepInContent attribute specifies if the value of the property on the EntityType of this attribute should be included in both the element specified via the Customizable Feed mapping as well as the original <m:properties> elements inside of 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 **AtomPub element**, the FC_NsPrefix attribute MUST be specified. If the Entity Type property is being mapped to an AtomPub element, the FC_NsPrefix attribute MUST be ignored if specified. If the FC_TargetPath attribute of the property mapping does not specify an AtomPub element and the FC_NsPrefix attribute is not supplied, this document does not mandate that 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, the FC_NsUri attribute MUST be specified on the Entity Type's definition. If the property is being mapped to an AtomPub element, the FC_NsUri attribute MUST be ignored.

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 AtomPub element, the value of the FC_TargetPath specifies a path to either an element or an attribute found inside the resulting Atom entry. All paths' expressions MUST be rooted at the <entry> element in the Atom feed.

The syntax of the target path is defined as:

```
targetPath = targetPathExpression ["/" attribute] \ atomSpecificElement

targetPathExpresion = targetPathExpression "/" targetPathExpression
                \ element

element = <the name of an element in the AtomPub document>

atomSpecificElementName = "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 XMLelements with mixed content:

```
TargetPath1: "a/b/c"
TargetPath2: "a/b/d"
TargetPath3: "a/b"
```

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 using the <Property> element) in the CSDL and the property is a Primitive Property, the mapping MUST NOT 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, then 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.

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 using the rules specified in [RFC3987] and [RFC3986] and then compared for equality 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 URIs in the example is `http://host/service.svc/Orders(1)`.

2.2.4 HTTP Methods

This section describes only the **HTTP** methods defined by this document. All additional **HTTP** methods used by this document are specified in [RFC2616].

2.2.4.1 MERGE

Data services support two types of update operations: merge and replace. As per [RFC5023] and described in Update Request Types (section 2.2.7.3), the **HTTP PUT** method specifies that an update operation SHOULD be carried out using replace semantics.

This section defines a custom **HTTP MERGE** method to specify that an update is to be completed 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 requests using MERGE and PUT is client intent.

Since MERGE is not one of the verbs defined in the HTTP specification [RFC2616], using the MERGE verb will likely not flow through network intermediaries as seamlessly as methods defined in HTTP, as specified in [RFC2616]. Data services 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 (JSON name/value pairs for JSON serializations, such as XML elements/attributes for XML-based requests) of the request body to the entity as it exists on the server.

If there is a component in the request body that is not defined on the entity being updated then the request MAY be considered malformed.

If there is a component in the request body that does match a component on the entity to be updated then the value of the component in the request body SHOULD 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 syntax of the **HTTP MERGE** method is defined as follows.

```
Method =/ "MERGE"    ; see [RFC2616] section 5.1.1
```

Listing: ABNF Grammar for MERGE HTTP Method

2.2.5 HTTP Header Fields

The protocol defined in this document uses existing headers, as specified in [RFC2616], as well as custom HTTP headers defined in this document. Some of the headers specified in [RFC2616] are further restrained by this document in how they can be used. These additionally restrained headers, as well as the custom headers, are defined in this section.

Unless otherwise specified, the headers defined in this document and any tokens, also called tags or directives, 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, receiving a request-only header in a response), the header MUST be interpreted as specified in [RFC2616].

If a client or server receives an HTTP header 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 defined in this section, but the header contains a token that is not defined in the current context (for example, receiving a request-only token 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 defined in this section 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 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 protocol defined in this document enables exchanging resources in multiple serialization formats (AtomPub, as specified in [RFC5023], 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 may find JSON easier to use because this format can be directly turned into JavaScript objects. On the other hand, a client application may be written with a language/runtime library that has a rich, built-in XML parser, making an XML-based format an appropriate choice.

The protocol defined in this document 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

Value of Accept Request Header	Value of Content-Type Response Header
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]	application/json

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 4xx response.

The protocol defined in this document 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) and application/json (section 2.2.5.1.2) formats. A data service can accept requests with Accept header values other than those shown in the table earlier in this section. 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 in a request to a data service to request the data service format for the response payload according to the application/atom+xml (section 2.2.5.1.1) format using the formatting rules outlined in AtomPub Format (section 2.2.6.2).

2.2.5.1.2 application/json

This content type is used in a request to a data service to request the data service format for the response payload using the application/json (section 2.2.5.1.2) format according to the formatting rules outlined in JavaScript Object Notation (JSON) Format (section 2.2.6.3).

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, application/atom+xml, 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"
/ "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 can 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 on a request, the VersionNum section of the header value states the version of the protocol defined in this document that the client used to generate the request, as specified in the preceding DataServiceVersion Header ABNF Grammar listing.

If present on a response, the VersionNum section of the header value states the version of the protocol defined in this document 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 versions 1.0 and 2.0 of the protocol defined by this document.

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 associated with a service, some EntityTypes may 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<49> define concurrency tokens on the base EntityType associated with each EntitySet in the Entity Data Model 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 MUST NOT 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 2.2.7.2.6) 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 may be provided. Additional constraints are also added to the syntax of the header value.

This header MAY<50> only be present on **GET**, **MERGE**, 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<51> 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". This header MUST NOT be 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-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 may be used. Additional constraints are also added to the syntax of the header value.

This header MAY<52> be present only on **GET**, **MERGE**, 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<53> 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 on 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, see section 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 (**HTTP PUT**, **HTTP 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 only valid when on **HTTP POST** requests. A server MAY<54> 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 must 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 will interpret the request as an insert request since **HTTP 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"
                 / "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

2.2.6 Common Payload Syntax

The protocol defined in this document enables clients and servers to perform actions (for example, CRUD operations) on entities in an Entity Data Model, as specified in [MC-CSDL], represented using

one of multiple possible formats (AtomPub, as specified in [RFC5023], JSON, and so on). Each serialization format or representation of an entity may be used in the payload of request and response messages, as specified in Request Types (section 2.2.7).

The AtomPub Format (section 2.2.6.2) specifies how to represent Entity Data Model constructs (single EntityType instance, multiple EntityType instances in an EntitySet, NavigationProperties, and so on) using the AtomPub [RFC5023] format.

JavaScript Object Notation (JSON) Format specifies how to represent Entity Data Model constructs (single EntityType instance, multiple EntityType instances in an EntitySet, NavigationProperties, and so on) using the JavaScript Object Notation (JSON) [RFC4627] format.

It should be noted 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 outlined in this section.

2.2.6.1 Common Serialization Rules for XML-Based Formats

AtomPub and custom XML formatted payloads representing 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], shown in the following Protocol Namespace Definitions table. All XML elements and attributes associated with the protocol defined in this document, as well as custom XML formats that hold data, are defined in the "Data Service" namespace.

All metadata-related elements 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 using the notation "atom:<elementName>".
Data Service Namespace	This namespace URI may 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 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 using the notation "m:<elementName>".

Table: Protocol Namespace Definitions

XML payloads defined in this document may 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** 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 [RFC3548] 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.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	int32Literal	See int32Literal in the Literal Form of Entity Data Model Primitive Types table in Abstract Type System (section 2.2.2).
Edm.String	stringLiteral	See stringLiteral 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

EDM Primitive Type	ABNF Rule for Primitive Type Representation in XML-based Payloads	Serialization Format (ABNF Grammar)
		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).

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 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 must have a title.

The following subsections define the mapping of constructs in the Entity Data Model to <Atom format> elements 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 described then an associated Atom-based representation is not defined by this document. For such constructs, servers and clients MAY<55> 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 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 MUST be 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 MUST adhere to the rules defined in this section.

<atom:feed> Element

This element is specified in [RFC4287] section 4.1.1

<atom:feed> Sub Elements

The <atom:id> element, as specified in [RFC4287] section 4.2.6, MUST contain the URI that identifies the EntitySet 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, can contain the name of the EntitySet represented by the parent <atom:feed> element. The set name can be qualified with the name of the EDM namespace in which it is defined, as specified in see [MC-CSDL]. If the URI in the sibling <atom:id> element is of the same form as URI 6, 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 can contain the name of the NavigationProperty instead of the name of the EntitySet 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 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).

<atom:entry> elements, as specified in [RFC4287] section 4.1.2, within the <atom:feed> element, are 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 identified by the associated URI, then the <atom:feed> element represents a partial collection as defined in AtomPub [RFC5023] section 10.1. The href attribute of the <atom:link rel="next"> element 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.

Implementers of this protocol should note that the inclusion of an <atom:link rel="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)

This section defines an extended representation of a collection of entities from that described in section 2.2.6.2.1. This representation is only supported in version 2.0 of the protocol defined by this specification.

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 MUST occur before any <atom:entry> elements in the feed.

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/related/CustomerDemographics" type="application/atom+xml;type=feed"
       title="CustomerDemographics" href="Customers('ALFKI')/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>
```

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 Sub 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 MUST also contain one <m:properties> child

element. The <m:properties> element MUST contain one child element for each EDMSimpleType and ComplexType property of the EntityType instance 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). If the Entity Type instance being represented was identified with a URI that includes a Select System Query Option (section 2.2.3.6.1.11), then 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 representing 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 must be the same as the property it represents. Rules for representing an Entity Type as an <atom:entry> element with Customizable Feed property mappings defined 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 and ComplexType properties of the EntityType instance, represented as described by 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 an **atom:rel="edit-media" attribute.** If such an <atom:link> element identifies a Media Resource with an associated concurrency token, then 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 an **atom:term** attribute and an **atom: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, then the <atom:category> element can be included. The value of the **atom:term** attribute MUST be the namespace qualified name of the EntityType of the instance represented by the <atom:entry> element. The value of the **atom: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 can 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 an **atom:rel="edit"** or **atom:rel="self"** attribute. The **atom:rel** attribute MAY<56> 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 **atom: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 each 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 an **atom: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 an **atom:title** attribute with the value equal to the NavigationProperty name and MUST contain an **atom:href** attribute with value equal to the URI which identifies the NavigationProperty on the EntityType. Implementers should note that Atom also requires an **atom:type** attribute, which should 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
```

Listing: Entity Type Atom Representation URIs (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: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" />
  <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:Address>
    <d:Version>AAAAAAAA+gE=</d:Version>
    </m:properties>
  </content>
</entry>
```

Listing: 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 Entity Type instance represented includes Customizable Feeds annotations in the data services metadata document, then the properties with custom mappings must be represented as directed by the mappings information specified in Conceptual Schema Definition Language Document for Version 2.0 Data Services (section 2.2.3.7.2.1). Properties that do not have Customizable Feeds mappings defined are represented according to the previous section, Entity Type (as an Atom Entry Element) (section 2.2.6.2.2).

If the property of an Entity Type instance in a Data Service response includes Customizable Feed annotations in the data services metadata document and has a value of null, then the element or attribute being mapped to can be present and MUST be empty.

For example, the Employee Entity Type 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:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
       xmlns:m="http://schemas.microsoft.com/ado/2008/11/dataservices/metadata"
       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: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

2.2.6.2.3 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.

An Atom representation of a ComplexType outside of 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 using XML independent from the content of the defining EntityType.

2.2.6.2.4 Navigation Property

See the description of the <atom:link> element in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) and Deferred Content (section 2.2.6.2.6) that follows.

2.2.6.2.5 EDMSimpleType Property

For a description of how properties are serialized in request/response payloads representing 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, may be large. For resource conservation purposes (bandwidth, CPU, and so on) a data service will generally 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 should 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 Orders 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 URI `service.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 may include the $expand System Query Option to explicitly request that the entity or entities represented by a NavigationProperty be serialized inline (rather than deferred), as described in Expand System Query Option ($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 Orders 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 the value of a NavigationProperty is null, then an empty <m:inline> element MUST appear under the <atom:link> element which represents the NavigationProperty, indicating that the element has been expanded but that there was no content 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/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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/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/related/OrderLines"
                 type="application/atom+xml;type=feed" title="OrderLines"
                 href="Orders(2)/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: Atom-formatted Customer Entity with the Orders 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 SHOULD identify such a service document. In this service document, a data service SHOULD 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, 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 can 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 Entity Data Model 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 and JSON serialization rules for such resources, see XML Format (section 2.2.6.5) and JavaScript Object Notation (JSON) Format (section 2.2.6.3).

2.2.6.3 JavaScript Object Notation (JSON) Format

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]. 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 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 JSON representation is not defined by this document.

2.2.6.3.1 Common JSON Serialization Rules for All EDM Constructs

Literal values of the EDM **Primitive** types are represented as JSON literal values, as defined by the rules in the following Common 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 JSON Payloads	JSON Serialization Format (ABNF Grammar)
Edm.Binary	**jsonBinary**	`jsonBinary = <Base64 encoded value of the EDM.Binary property represented as a JSON string. See [RFC4627] for further details>`

Edm.Boolean	**jsonBoolean**	`jsonBoolean = false / true`
		
		`false = ; see [RFC4627] section 2.1`
		
		`true = ; see [RFC4627] section 2.1`

Edm.Byte	**jsonByte**	`jsonByte = ; see the byteLiteral rule in ; section 2.2.2`

Edm.DateTime	**jsonDateTime**	`jsonDateTime= 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`
		
		`;Note: This format is the same used by the ASP.NET ;AJAX framework, described in http://msdn2.microsoft.com/en- ;us/library/bb299886.aspx`

Edm.Decimal	**jsonDecimal**	`jsonDecimal = quotation-mark decimalLiteral`

EDM Primitive Type	ABNF Rule for Primitive Type Representation in JSON Payloads	JSON Serialization Format (ABNF Grammar)
		quotation-mark decimalLiteral = ; see section 2.2.2 quotation-mark = ; see [RFC4627] section 2.5
Edm.Double	**jsonDouble**	jsonDouble = quotation-mark doubleLiteral quotation-mark doubleLiteral = ; see section 2.2.2
Edm.Guid	**jsonGuide**	jsonGuid = quotation-mark guidLiteral quotation-mark guidLiteral = ; see section 2.2.2
Edm.Int16	**jsonInt16**	jsonInt16 = ; see int16Literal in section 2.2.2
Edm.Int32	**jsonInt32**	jsonInt32 = ; see int32Literal in section 2.2.2
Edm.Int64	**jsonInt64**	jsonInt64 = quotation-mark int64Literal quotation-mark int64Literal = ; see section 2.2.2
Edm.SByte	**jsonSByte**	jsonSByte = ; see sByteLiteral in section 2.2.2
Edm.Single	**jsonSingle**	jsonSingle = quotation-mark singleLiteral quotation-mark singleLiteral = ; see section 2.2.2
Edm.String	**jsonString**	jsonString = string string = ; see [RFC4627] section 5.2
Edm.Time	**jsonTime**	jsonTime = quotation-mark timeLiteral quotation-mark

EDM Primitive Type	ABNF Rule for Primitive Type Representation in JSON Payloads	JSON Serialization Format (ABNF Grammar)
		timeLiteral = ; see section 2.2.2
Edm.DateTypeOffset	**jsonDateTimeOffset**	jsonDateTimeOffset = jsonDateTime

Table: Common JSON Serialization Rules for All EDM Constructs

2.2.6.3.2 Entity Set (as a JSON Array)

An EntitySet or collection of entities MUST be represented as an array of JSON objects, with one object for each EntityType instance within the set. A JSON-based format for EntityTypes is defined in Entity Type (as a 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 version 1.0 and version 2.0 compliant JSON representations of a collection of entities is defined by the grammar listed in this section. The grammar rule "entitySetInJson" defines the version 1.0 JSON representation of a collection of entities that may be used in both request and response payloads. The grammar rule "entitySetInJson2" defines the version 2.0 JSON representation of a collection of entities for response payloads only. No version 2.0 format for use in request payloads is defined by this specification.

```
; version 1.0 JSON representation of a collection of entities:

entitySetInJson =  begin-array
                  [entityTypeInJson *(value-seperator entityTypeInJson)]
                   end-array

; version 2.0 JSON representation of a collection of entities:
entitySetInJson2 = begin-object
                   [countNVP value-seperator]
                   resultsNVP
                   [value-seperator nextLinkNVP]
                   end-object


resultsNVP      = quotation-mark "results" quotation-mark
                  name-seperator
                  begin-array
                  [entityTypeInJson *(value-seperator entityTypeInJson)]
                  end-array

; see section 2.2.6.3.2.1 for additional details
countNVP        = quotation-mark "__count" quotation-mark
                  name-seperator
                  quotation-mark
                  <count value as defined in section 2.2.6.3.2.1>
                  quotation-mark


nextLinkNVP     = quotation-mark "__next" quotation-mark
                  name-seperator
```

```
                    begin-object
                    nextUriNVP
                    end-object

nextUriNVP      =   quotation-mark "uri" quotation-mark
                    name-seperator
                    quotation-mark
                    resourcePath "?" [skiptokenQueryOp]
                    quotation-mark

entityTypeInJson = ; see section 2.2.6.3.3
resourcePath     = ; see section 2.2.3.1
skiptokenQueryOp = ; see section 2.2.3.6.1.9
```

Listing: Entity Set JSON Representation

In response payloads representing a collection of entities, if the server does not include an entityTypeInJson 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 nextURINVP 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.

Implementers of this protocol should note that the inclusion of a nextLinkNVP name value pair in a 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)

This section defines the semantics of the "countNVP" grammar rule in section 2.2.6.3.2, which is only supported in version 2.0 of the protocol defined by this specification.

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 should 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 (before the results 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.

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 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.

```
{
  "__count": "91",
  "results":[
{
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "CustomerID": "ALFKI",
```

```
    "CompanyName": "Alfreds Futterkiste",
    "Address": { "Street": "57 Contoso St", "City": "Seattle" },
    "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(2)",
                             "type": "SampleModel.Order",
                           },
             "OrderID": 2,
             "ShippedDate": "\/Date(875836800000)\/",
             "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
             "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }

          }
    ]  }
  ]
}
```

2.2.6.3.3 Entity Type (as a JSON Object)

An instance of an EntityType MUST be serialized as a 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 Entity Type instance being represented is identified with a URI that includes a Select System Query Option (section 2.2.3.6.1.11), then 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 sub-sections below describe additional formatting rules for each type of property defined on an EntityType.

The JavaScript Object Notation (JSON) serialization of an EntityType instance MAY<57> 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 value of the "__metadata" property contains seven name/value pairs: "uri", "type," "etag", "edit_media", "media_src", "media_etag", and "content_type". The order of these name/value pairs is insignificant. The value of the "uri" name/value pair MUST be the canonical URI 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, then the "type" name/value pair can 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 can 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 can 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' conceptual schema definition language 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, then the "edit_media", "media_src", "media_etag", and "content_type" name/value pairs MUST NOT be included.

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:link rel="edit-media"> AtomPub element if the entity was to be represented by the AtomPub [RFC5023] format, instead of 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 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 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" or "media_src" name/value pair.

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 version 1.0 and version 2.0 compliant JSON representations of an entity is defined by the grammar listed in this section. The grammar rule "entityTypeInJson" defines the version 1.0 and version 2.0 JSON representation of an entity that can be used in both request and response payloads.

```
; version 1.0 JSON representation of an entity:
entityTypeInJson  =  entityTypeBody

resultsNVP        = quotation-mark "results" quotation-mark
                    name-seperator
                    entityTypeBody

entityTypeBody    = begin-object
                    (
                      metadataNVP
                      / (metadataNVP
                           (value-seperator entityTypeProperty))
                      /  (entityTypeProperty)
                    )
                    *(value-seperator entityTypeProperty)
                    end-object

metadataNVP       =  quotation-mark "__metadata" quotation-mark
                     name-seperator
                     begin-object
                     ( uriNVP
                       [value-seperator typeNVP]
                       [value-seperator etagNVP]
                       [mleMetadata])
```

```
                    /
                    ( typeNVP
                      [value-seperator etagNVP])
                    /
                      etagNVP
                    end-object

entityTypeProperty = entityPropertyInJson
                       /entityCTInJson
                       /deferredNavProperty

uriNVP           =  quotation-mark "uri" quotation-mark
                    name-seperator
                    quotation-mark resourcePath quotation-mark

typeNVP          =  quotation-mark "type" quotation-mark
                    name-seperator
                    quotation-mark entityType quotation-mark

etagNVP          =  quotation-mark "etag" 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


mleMetadata      =  [value-seperator editMediaNVP]
                    value-seperator media_srcNVP
                    value-seperator contentTypeNVP


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
entityCTInJson            =    ; section 2.2.6.3.4
entityPropertyInJson      =    ; section 2.2.6.3.8
entityPropertyValueInJson =    ; 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: Entity Type JSON Representation

For example, the Customer EntityType instance described in Appendix A: Sample Entity Data Model and CSDL Document (section 6) is represented in JSON, as shown in the following listing.

```
{
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
```

Listing: JSON-formatted Customer Entity

2.2.6.3.4 Complex Type

An instance of a ComplexType MUST be represented as a 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 version 1.0 and version 2.0 compliant JSON representations of a ComplexType is defined by the grammar listed in this section. The grammar rule "entityCTInJson" defines the version 1.0 JSON representation of a ComplexType that can be used in both request and response payloads. The grammar rule "entityCTInJson2" defines the version 2.0 JSON representation of a ComplexType for response payloads only. This specification does not define a version 2.0 JSON representation of an entity for use in request payloads.

```
; version 1.0 JSON representation of a ComplexType:
entityCTInJson  = entityCTBody

; version 2.0 JSON representation of a ComplexType:
entityCTInJson2 = begin-object
                  resultsNVP
                  end-object

resultsNVP      = quotation-mark "results" quotation-mark
                  name-seperator
                  begin-object

                  entityCTBody
```

```
                  end-object

entityCTBody    = quotation-mark entityComplexProperty quotation-mark
                  name-seperator
                  entityCTValue

entityCTValue   = begin-object
                  [
                    ((entityPropertyInJson)
                    / (entityCTBody))

                    *( (value-seperator entityPropertyInJson)
                      /(value-seperator entity CTBody)
                    )
                  ]
                  end-object

entityPropertyInJson   = ; see section 2.2.6.3.8
```

Listing: Complex Type 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 section 2.2.6.3.4.

The syntax of version 1.0 and version 2.0 compliant JSON representations of a collection of ComplexType instances is defined by the grammar listed in this section. The grammar rule "entityCollCTInJson" defines the version 1.0 JSON representation of a collection of ComplexType instances that can be used in both request and response payloads. The grammar rule "entityCollCTInJson2" defines the version 2.0 JSON representation of a collection of ComplexType instances for response payloads only. This specification does not define a version 2.0 JSON representation of a collection of ComplexType instances for use in request payloads.

```
; version 1.0 JSON representation of a collection of ComplexType instances:
entityCollCTInJson =  begin-array
                      entityCTValue      ; see section 2.2.6.3.4
                      [value-seperator entityCTValue]
                      end-array

; version 2.0 JSON representation of a collection of ComplexType instances:
entityCollCTInJson2 = begin-object
                      resultsNVP
                      end-object

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 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 version 1.0 and version 2.0 compliant JSON representations of a collection of EDMSimpleType values is defined by the grammar listed in this section. The grammar rule "entityCollPrimValueInJson" defines the version 1.0 JSON representation of a collection of EDMSimpleType values that can be used in both request and response payloads. The grammar rule "entityCollPrimValueInJson2" defines the version 2.0 JSON representation of a collection of EDMSimpleType values for response payloads only. This specification does not define a version 2.0 JSON representation of a collection of EDMSimpleType values for use in request payloads.

```
; version 1.0 JSON representation of a collection of EDMSimpleType values:
entityCollPrimValueInJson =  begin-array
                             entityPropertyValueInJson
                             ; see section 2.2.6.3.8
                             [value-seperator entityPropertyValueInJson]
                             end-array

; version 2.0 JSON representation of a collection of EDMSimpleType values:
entityCollPrimValueInJson2 = quotation-mark "results" quotation-mark
                             name-seperator
                             begin-array
                             entityPropertyValueInJson
                             ; see section 2.2.6.3.8
                             [value-seperator entityPropertyValueInJson]
                             end-array
```

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 EDM property and the value must be set to the value of the property. The value MUST be formatted, as specified in Common JSON Serialization Rules for All EDM Constructs (section 2.2.6.3.1).

When represented as part of the JSON representation of an EntityType or ComplexType, the syntax of an EDMSimpleType Property formatted in JSON is as follows.

```
entityPropertyInJson =  quotation-mark entityProperty quotation-mark
                        name-seperator
                        entityPropertyValueInJson
```

```
entityPropertyValueInJson = <EDMSimple type serialized as per section 2.2.6.3.1>
```

When represented as a standalone construct, the syntax of version 1.0 and version 2.0 compliant JSON representations of an EDMSimpleType is defined by the grammar listed as follows. The grammar rule "entityPropertyInJson" defines the version 1.0 JSON representation of property that can be used in both request and response payloads. The grammar rule "entityPropertyInJson2" defines the version 2.0 JSON representation of a property for response payloads only. This specification does not define a version 2.0 JSON representation of a property for use in request payloads.

```
;version 1.0 JSON representation of a property:
entityPropertyInJson  = quotation-mark entityProperty quotation-mark
                        name-seperator
                        entityPropertyValueInJson

; version 2.0 JSON representation of a property:
entityPropertyInJson2 = quotation-mark "results" quotation-mark
                        name-seperator
                        begin-object
                        quotation-mark entityProperty quotation-mark
                        name-seperator
                        entityPropertyValueInJson
                        end-object
```

2.2.6.3.9 Deferred Content

The serialized representation of an entity and its related entities, identified by NavigationProperties, may 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 should 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 JSON-formatted EntityType instances (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 equal to the URI 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 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 Navigation Property 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 may 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 below uses the same data model as the Deferred Content example referenced above; however, this example shows the value of the Orders NavigationProperty serialized inline.

A NavigationProperty which 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).

```
{
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders": [
         {
            "__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(2)",
                            "type": "SampleModel.Order"
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }

         }
   ]
}
```

Listing: Version 1.0 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 JSON format, Links are serialized as an array of URIs, each of which identifies a single linked entity.

When represented in JSON, Links MUST be formatted, as shown in the table in the following listing, ABNF Grammar for Links Represented in 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 version 1.0 and version 2.0 compliant JSON representations of a collection of links is defined by the grammar listed in this section. The grammar rule "linkCollJson" defines the version 1.0 JSON representation of a collection of links that can be used in both request and response payloads. The grammar rule "linkCollJson2" defines the version 2.0 JSON representation of a collection of links for response payloads only. This specification does not define a version 2.0 JSON representation of a collection of links for use in request payloads.

```
; version 1.0 JSON representation of a collection of  links
linkCollJson2       =  begin-array
                       *linkJson
                       end-array

; version 2.0 JSON representation of a collection of links
linkCollJson2       =  begin-object
                       [countNVP value-seperator]
                       linkCollResultsNVP
                       end-object

linkCollResultsNVP  =  quotation-mark "results" quotation-mark
                       name-seperator
                       begin-array
                       *linkJson
                       end-array

countNVP            =  ; see section 2.2.6.3.2

;Grammar rules common to version 1.0 and 2.0
linkJson            =  begin-object
                       [linkUriNVP
                        *(value-seperator linkUriNVP) ]
                       end-object

linkUriNVP          =  quotation-mark "uri" quotation-mark
                       name-seperator
                       quotation-mark dataSvcAbsNqo-URI quotation-mark
                       ; see section 2.2.3.1
```

Listing: ABNF Grammar for Links Represented in JSON

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 "ALFKI" to Order instances is represented as shown in the following Example of Links Formatted using JSON 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 using Version 1.0 JSON Representation

2.2.6.3.11 Inlinecount Representation (for Collections of Links)

This section defines the semantics of the "countNVP" grammar rule in section 2.2.6.3.10, which is only supported in version 2.0 of the protocol defined by this specification.

A request **URI** MAY contain an $inlinecount System Query Option to indicate that the count of the number of links represented by the query should be 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 (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. This section defines a 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 JSON serialization, see Service Document (section 2.2.3.7.1).

The syntax of a JSON-serialized Service Document is as shown in the grammar that follows.

```
jsonServiceDocument = 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 using the JSON format, as specified in [RFC4627], representation of the information provided by an AtomPub, as specified in [RFC5023], Service Document for the Entity Data Model described in Appendix A: Sample Entity Data Model and CSDL Document (section 6).

```
{
  "EntitySets": [
     "Customers",
     "Orders",
     "OrderDetails"
  ]
}
```

Listing: JSON Service Document Describing a Data Service

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 data service MAY<58> 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 identifying 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 should 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, then there does not exist a serialization of the property's value in this form. In such cases, it is left to the protocol's interaction model to signal such values, as described in Message Processing Events and Sequencing Rules (section 3.2.5) .

```
<?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 MUST be 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 must be represented as a child element of the root element and be 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 MUST specify the namespace qualified type name of the ComplexType.

Each property of the ComplexType instance MUST be 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 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 returning the values. The root element and all its child elements MUST exist in the Data Service Metadata namespace, as described in Common Serialization Rules for XML-based Formats (section 2.2.6.1).

Each value in the collection must be represented as a child element of the root element and be named "element". The text value of the XML element MUST be 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 from the Customer. Such Link information MUST be serialized as an XML document that conforms to the XSD Schema [XMLSCHEMA1] shown in the following XSD 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 can 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: XSD 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 XSD Schema [XMLSCHEMA1] shown in following XSD Schema for a single Link Represented listing using XML. The definition of the URI element in this case is unchanged from above. The targetNamespace in the XSD can 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: XSD 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 shown below.

```
<?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)

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 version 2.0 of the protocol defined by this specification.

A request URI MAY contain an $inlinecount System Query Option to indicate that the count of the number of links represented by the query should 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.

2.2.7 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 [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 constructs represented as data service or AtomPub resources using HTTP's **GET**, **PUT**, and **DELETE** methods. For further details see Retrieve Request Types (section 2.2.7.2), Update Request Types (section 2.2.7.3), and Delete Request Type (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 may 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.

Section 2.2.6.1.1 defines the InsertEntity request type which enables a client to insert a new EntityType instance into an EntitySet.

Section 2.2.6.1.2 defines the InsertLink request type which 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 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 must represent an AtomPub Collection. Because a collection maps to a conceptual schema definition language (CSDL) in an Entity Data Model, 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.

RFC [RFC5023], section 9.2 states that the request body in the POST MAY be an AtomPub Entry Resource (which maps to an EntityType instance in an Entity Data Model) 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 two such representations of EntityTypes that map to Entry Resources: AtomPub, as specified in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2), its subsections, and JSON (Entity Type (as a 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 if the resource/entity should be automatically linked to other already-existing entities in the data service. For example, a new order entity may need to be bound to an existing customer entity in a customer relationship management focused data service. Such linking MUST 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 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 must represent the URI of the entity to be linked to.

To bind the new entity to an existing entity using 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 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]
                           [MaxDataServiceVersion]
                           [Accept]
                           [Content-Type]
                           *(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 Type in 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 version 1.0 and 2.0 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
                            <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)

; Version 1.0 insertEntity-RespBody grammar rule:
insertEntity-RespBody     = <Entity Type in Atom format as per section 2.2.6.2.2>
                            / (begin-object
                               quotation-mark "d" quotation-mark
                               name-seperator
                               entityTypeInJson
                               end-object)
```

```
                              ; see section 2.2.6.3.3

; Version 2.0 insertEntity-RespBody grammar rule:
insertEntity-RespBody     = <Entity Type in Atom format as per section 2.2.6.2.2>
                           / (begin-object
                              quotation-mark "d" quotation-mark
                              name-seperator
                              singleEntityResultNVP
                              end-object)
                             ; see section 2.2.6.3.3

singleEntityResultNVP     = quotation-mark "results" quotation-mark
                            name-seperator
                            begin-object
                            [entityTypeInJson *(value-separator entityTypeInJson)]
                            end-object
```

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 using the Atom format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml
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
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" />
  <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 using the JSON format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
```

```
Host: host
Content-Type: application/json
Accept: application/json
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:

```
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
DataServiceVersion: 1.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

{
 "d":
    {
      "__metadata": { "uri": "Customers(\'ASDFG\')",
                      "type": "SampleModel.Customer",
                      "etag": "W/\"X\'000000000000FA01\'\""
                    },
      "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 using the Atom format.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/atom+xml
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:

```
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
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" />
  <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 using the JSON.

HTTP Request:

```
POST /service.svc/Customers HTTP/1.1
Host: host
Content-Type: application/json
Accept: application/json
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:

```
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
DataServiceVersion: 1.0
Content-Length: nnn
ETag: W/"X'000000000000FA01'"

{"d":
 {
   "__metadata": { "uri": "Customers(\'ASDFG\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "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 URI 7 grammar rule in Resource Path: Semantics (section 2.2.3.5). For example, 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 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 according to the "linkJson" rule in EDMSimpleType Property (section 2.2.6.3.8) or the XML schema for a single link in EDMSimpleType Property (section 2.2.6.5.3)) which 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
                            *(HTTP-Header-Types)

insertLink-ReqBody      =   linkJson                    ; 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
                             *(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 constructs are mapped directly to data model concepts used in AtomPub. If an Entity Type instance is created via an InsertMediaResource request, then 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 must represent an AtomPub Collection. Because a collection maps to an EntitySet (or collection of entities) in an Entity Data Model, the HTTP request line URI MUST be any URI that identifies a collection of entities. If the Entity Type associated with the implicitly created Media Link Entry defines a concurrency token, then a successful response MUST contain an ETag header with a value equal to the value of the concurrency token of the Media Link Entry 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]
                         [MaxDataServiceVersion]
                         [Accept]
                         [Content-Type]
                         [<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
                            *(HTTP-Header-Types)
```

```
insertEntity-RespBody     = <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
                              entityTypeInJson
                              end-object)
                             ; see section 2.2.6.3.3
                             ; the entityTypeInJson 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 must represent an AtomPub Collection. Because a collection maps to an EntitySet in an Entity Data Model, 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 must 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 two representations of EntitySets: AtomPub, as specified in Entity Set (as an Atom Feed Element) (section 2.2.6.2.1) and JSON, as specified in Entity Set (as a 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 or JSON, according to the rules defined in AtomPub Format (section 2.2.6.2) and JavaScript Object Notation (JSON) Format (section 2.2.6.3).

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   =  <Entity Set formatted as per section
                                2.2.6.2.1>
                                / (begin-object
                                   quotation-mark "d" quotation-mark
                                   name-seperator
                                   (entitySetInJson / entitySetInJson2)
                                   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, 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 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 JSON object) (section 2.2.6.3.3).

AtomPub [RFC5023] describes retrieving an AtomPub Entry Resource, which maps to an EntityType instance in an Entity Data Model, 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 using the Accept (section 2.2.5.1) request header. This document defines two representations of EntityTypes: Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) and Entity Type (as a 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 using Atom or JSON according to the rules defined in Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) and Entity Type (as a 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 Type instance (possibly with
                                  related instances) formatted 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
                                  entityTypeInJson
                                  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 or JSON according to the rules defined in Complex Type (section 2.2.6.5.1) and Complex Type (section 2.2.6.3.4).

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      =  <ComplexType property value formatted using XML as per
                             section 2.2.6.5.1>
                            /(begin-object
                              quotation-mark "d" quotation-mark
                              name-seperator
                              (entityCTInJson / entityCTInJson2)
                              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 2xx status code as specified in [RFC2616]. The payload of such a response MUST be formatted using XML or JSON, as defined in EDMSimpleType Property (section 2.2.6.5.3) and EDMSimpleType Property (section 2.2.6.3.8).

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 XML as per section
                            2.2.6.5.3>
                           / (begin-object
                              quotation-mark "d" quotation-mark
                              name-seperator
                              begin-object
                              (entityPropertyInJson / entityPropertyInJson2)
                              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 2xx 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 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
```

2.2.7.2.7 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 using the Accept (section 2.2.5.1) request header. This document defines two such representations of Service Documents: Service Document (section 2.2.6.2.7) for AtomPub and Service Document (section 2.2.6.3.12) for 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 and Service Document (section 2.2.6.3.12) for 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   = <AtomPub Service Document formatted as per
                                      section 2.2.6.2.7>
                                     /(begin-object
                                       quotation-mark "d" quotation-mark
                                       name-seperator
                                       jsonServiceDocument
                                       end-object)
```

```
                                ; see section 2.2.6.3.11
```

2.2.7.2.8 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 2xx 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 two such formats for Links: XML and JSON, as defined in Links (section 2.2.6.5.5) and Links (section 2.2.6.3.10).

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     =  quotation-mark "d" quotation-mark

<Representation of the links or link addressed in the
                               request URI formatted as per section 2.2.6.3.10 or
                               2.2.6.5.5>
```

2.2.7.2.9 RetrieveCount Request

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 2xx 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 version 2.0 of the protocol defined by this specification.

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>
```

2.2.7.2.10 Retrieve Request Containing a Customizable Feed Mapping

In version 2.0 of the protocol defined by this specification, 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.

2.2.7.2.11 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 must represent an AtomPub Entry resource. Given that an Entry Resource maps to an EntityType instance in an Entity Data Model, 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). This document extends the AtomPub semantics for this request type to also allow UpdateEntity Requests to use the **HTTP MERGE** method instead of **PUT** to specify that a merge-based update should be performed, as described in MERGE (section 2.2.4.1).

[RFC5023] section 5.4.2, states that the request body MAY be an AtomPub Entry Resource (that maps to an EntityType instance in an Entity Data Model) 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 two such representations: Entity Type (as an Atom Entry Element) (section 2.2.6.2.2) and its subsections and Entity Type (as a 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) should 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 may 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 must equal the URI that identifies the existing entity that is the target of the link.

To bind the new entity to an existing entity using the JSON format, the NavigationProperty MUST represent the entity that is the target of the link 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 was successful, the response MUST have a 204 status code, as specified in [RFC2616], and have 0 bytes in the response body. If the UpdateEntity 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 UpdateEntity Request is defined as follows:

```
updateEntity-Req         = updateEntity-ReqLine
                           updateEntity-ReqHeaders
                           CRLF
                           updateEntity-ReqBody

updateEntity-ReqLine     = ("PUT" / "MERGE")
                           SP entityTypeInstanceUri updateEntity-QueryOps
                           SP HTTP-Version CRLF

updateEntity-ReqHeaders  = [DataServiceVersion]
                           [MaxDataServiceVersion]
                           [If-Match]
                           [Content-Type]
                           *(HTTP-Header-Types)

entityTypeInstanceUri = ; see section 2.2.7.4.1

updateEntity-QueryOps    = ["?" (customQueryOption / filterQueryOp) *("&" customQueryOption)]
                           ; see section 2.2.3.1

updateEntity-ReqBody     = entityTypeInJson       ; 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-RespHeaders  = DataServiceVersion   ; see section 2.2.5.3
                           [ETag]                ; see section 2.2.5.4
                           *(HTTP-Header-Types)
```

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
Accept: application/json
Content-Length: nnn

{
   "__metadata":{ "uri": "Orders(1)" },
   "OrderID": 1,
   "ShippedDate": "\/Date(872467200000)\/",
   "Category" : { __metdata: {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** or **MERGE** method, as specified in 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, 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 XML or JavaScript Object Notation (JSON) Format, according to the rules defined in Complex Type (section 2.2.6.5.1) and Complex Type (section 2.2.6.3.4). A server accepting a request of this type using the **HTTP 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 **HTTP MERGE** method MUST merge the value of the ComplexType addressed via the request URI with the value provided in the payload, as specified in MERGE (section 2.2.4.1).

If the UpdateComplexType Request was successful, the response MUST have a 204 status code, as specified in [RFC2616], and have 0 bytes in the response body. If the UpdateComplexType 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 UpdateComplexType Request is defined as follows:

```
updateCT-Req            = updateCT-ReqLine
                          updateCT-ReqHeaders
                          CRLF
                          updateCT-ReqBody

updateCT-ReqLine        = "PUT" / "MERGE"
                          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
                          *(HTTP-Header-Types)

updateCT-ReqBody        = <ComplexType value formatted in XML as
                           per section 2.2.6.5.1>
                          / (begin-object entityCTInJson 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-RespHeaders    =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           *(HTTP-Header-Types)
```

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** or **HTTP MERGE** (as specified in 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 using XML or JSON, according to the rules defined in EDMSimpleType Property (section 2.2.6.5.3) and EDMSimpleType Property (section 2.2.6.3.8). 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 **HTTP MERGE** or **HTTP 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 was successful, the response MUST have a 204 status code, as specified in [RFC2616], and have 0 bytes in the response body. If the UpdatePrimitiveProperty 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 UpdatePrimitiveProperty Request is defined as follows:

```
updatePP-Req            = updatePP-ReqLine
                          updatePP-ReqHeaders
                          CRLF
                          updatePP-ReqBody

updatePP-ReqLine        = "PUT" / "MERGE"
                          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
                          *(HTTP-Header-Types)

updatePP-ReqBody        = <Property value formatted in XML as per section 2.2.6.5.3>
                          / (begin-object entityPropertyInJson 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-RespHeaders    =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           *(HTTP-Header-Types)
```

2.2.7.3.4 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** or **HTTP MERGE** method (as specified in 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), 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** or **HTTP PUT** method.

If the UpdateValue Request was successful, the response MUST have a 204 status code, as specified in [RFC2616], and have a 0 byte response body. If the UpdateValue 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 UpdateValue Request is defined as follows:

```
updateValue-Req         = updateValue-ReqLine
                          updateValue-ReqHeaders
                          CRLF
                          updateValue-ReqBody

updateValue-ReqLine     = "PUT" | "MERGE"
                          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
                          *(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-RespHeaders =  DataServiceVersion         ; see section 2.2.5.3
                           [ETag]                     ; see section 2.2.5.4
                           *(HTTP-Header-Types)
```

2.2.7.3.5 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** or **HTTP MERGE** method (as specified in 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 that identifies a single Link between two EntityType instances, as described in URI Format: Resource Addressing Rules (section 2.2.3). However, the URI MUST NOT identify both entities by key, as then the update operation itself becomes redundant.

For example, using Appendix A: Sample Entity Data Model and CSDL Document (section 6), the following is a valid URI for this type of request:
`http://host/service.svc/Orders(1)/$links/Customer`.

Here the URI identifies the Order by its key but the specific Customer is unknown.

However, the following URI is not valid for this type of request:
`http://host/service.svc/Customers('ALFKI')/$links/Orders(1)`

Here the URI identifies both entities by key, at which point doing an update has no semantic value; hence it is not supported.

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 according to the "linkJson" rule in Links (section 2.2.6.3.10) or the XML schema for a single link in Links (section 2.2.6.5.5)) which contains a URI identifying the EntityType instance to be linked to from the source entity.

If an UpdateLink Request is successful, the response MUST have a 204 status code as specified in [RFC2616] and contain 0 bytes in the response body.

If the UpdateLink 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 UpdateLink Request is defined as follows:

```
updateLink-Req          =   updateLink-ReqLine
                            updateLink-ReqHeaders
                            CRLF
                            updateLink-ReqBody

updateLink-ReqLine      =   "PUT" / "MERGE"
                            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
                            *(HTTP-Header-Types)

updateLink-ReqBody      =   linkJson                    ; 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-RespHeaders  =    DataServiceVersion       ; see section 2.2.5.3
                             *(HTTP-Header-Types)
```

2.2.7.3.6 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 must 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, the HTTP request line URI MUST be equal to any valid URI that identifies a Media Resource associated with an existing Entity Type instance. The **MERGE HTTP** method defined in this document is not supported for requests of this type.

If the UpdateMediaResource Request was successful, the response MUST have a 204 status code, as specified in [RFC2616], and have 0 bytes in the response body. If the UpdateMediaResource 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 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  = ; 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-RespHeaders    = DataServiceVersion    ; see section 2.2.5.3
                          [ETag]                ; see section 2.2.5.4
                          *(HTTP-Header-Types)
```

2.2.7.3.7 Update Request Containing a Customizable Feed Property Mapping

In version 2.0 of the protocol defined by this specification, 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 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), then 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 should address 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<59> 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 Requests 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 a malformed request (the server should respond with the HTTP 4xx 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

The purpose of the Invoke Request is to enable a client to call a FunctionImport as specified in [MC-CSDL], exposed by a Service Operation in a data service. 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], 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
                           / invoke-RespBody-prim

; 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 Type instances formatted using XML
                            as per section 2.2.6.5.2>
                           /(begin-object
                             quotation-mark "d" quotation-mark
                             name-seperator
                             entityCollCTInJson
                             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 type values formatted using XML
                            as per section 2.2.6.5.4>
                           /(begin-object
```

```
                                quotation-mark "d" quotation-mark
                                name-seperator
                                entityCollPrimValueInJson)
                                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
```

2.2.7.6 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 which reduces the number of roundtrips to a data service for applications that need to make numerous requests and a ChangeSet syntax as a way to logically group a set of requests into a single unit within a batch.

A data service MAY support requests of this type. If a data service does not implement support for a Batch Request, it must return a 4xx 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 URI 9 in Resource Path: Semantics (section 2.2.3.5).

Logically, a Batch Request is made up of a series of **Change Sets** and **Query Operations**. Change Sets and Query Operations MAY<60> 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 an Invoke Requests (section 2.2.7.5), which 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 (section 2.2.7.3)
- Delete Requests (section 2.2.7.4)
- Invoke Requests (section 2.2.7.5)), using the **HTTP PUT**, **HTTP POST**, or **HTTP DELETE** methods.

Requests 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 may 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 may have been applied before another request in that same Change Set failed and thereby honor this all-or-nothing requirement.

Batch Requests are represented 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 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 representing a request within a Change Set, in the multipart/mixed message defining a single Change Set, MUST include a Content-Type (see **contentTypeMime-Part**) and a Content-Transfer-Encoding (see **contentTransferEncodingMime-Part**) MIME header, described in the grammar 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
```

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 representing 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 representing a request in a Change Set MAY include a Content-ID MIME header, as specified in [RFC2045] section 7. If a MIME part defines an InsertEntity Request and includes a Content-ID header, then the new entity, defined by the InsertEntity Request, MAY be referenced by subsequent requests in the Change Set by using the "$<Content-ID value of previous request>" token in place of a Resource Path identifying the new resource. 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 MAY choose to disallow additional HTTP constructs in HTTP requests serialized within MIME part bodies. For example, a data service may 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
                             [id CRLF]              ; see [RFC2045] section 7

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
                              <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

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

PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json
If-Match: xxxxx
Content-Length: ###

<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 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 (section 2.2.7.6) body may subsequently fail or be malformed; however, 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 (section 2.2.7.6) with an invalid set of headers, it MUST return 4xx response code. For example, Batch Request (section 2.2.7.6) 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 (section 2.2.7.6) must result in a response containing a 4xx response code.

All responses to Batch Request (section 2.2.7.6) 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).

Structurally, a Batch Request (section 2.2.7.6) 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

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
                                      <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

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

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

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 should be 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) 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. This section defines two types of error conditions: "in-stream" and "top-level" errors.

"In-stream" errors occur when a data service 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. In this case, the HTTP response code, headers, and payload cannot be used to indicate an error has occurred. This document does not prescribe an Error Response generation method for errors of this type.

In the context of this document, "top-level" errors SHOULD be used when the data service 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 two formats for top-level error messages (XML and 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 formatted using XML. Servers MUST use this format when sending top-level error responses to requests that include an Accept (section 2.2.5.1) request header with the values "application/xml"

or "application/atom+xml", or to requests that do not include an Accept (section 2.2.5.1) request header. When formatting error responses using XML, servers SHOULD include a Content-Type response header with the value "application/xml".

JSON Error Response (section 2.2.8.1.2) defines the rules for a top-level error in a response body formatted using JSON, as specified in [RFC4627]. Servers MUST use this format when sending top-level error responses to requests that include an Accept (section 2.2.5.1) request header with the value "application/json". When formatting error responses using JSON, servers SHOULD include a Content-Type response header with the value "application/json".

Servers sending responses indicating top-level errors MUST use one of these two formats in the response payload.

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     =   <XML representation of a top-level error as defined in
                           section 2.2.8.1.1>
                          / stdErrorJson-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. This XML format MUST be used in response payloads representing top-level errors, as specified in Error Response (section 2.2.8.1), when the server does not use the JSON format for errors defined in JSON Error Response (section 2.2.8.1.2).

The structure of a top-level error message using XML (application/xml) is defined in the following XSD Schema for Top-level Error Payloads Formatted 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">
   xmlns:xml="http://www.w3.org/2001/XMLSchema"
<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:string"
      minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  </xs:schema>
```

code: A data service-defined string which serves as a substatus to the HTTP response code.

message: A human readable message describing the error.

innererror: This optional element contains data service specific debugging information to assist a service implememenTer in determining the cause of an error.

Note The <innererror> element should only be used in development environments and should not be present in a response from a production data service to guard against information disclosure security concerns.

Listing: XSD Schema for Top-level Error Payloads Formatted Using XML

2.2.8.1.2 JSON Error Response

This section defines the structure of an error message represented in JSON, as specified in [RFC4627]. This JSON format MUST be used in response payloads representing top-level errors, as specified in Error Response (section 2.2.8.1), when the server does not use the XML Format for errors defined in XML Error Response (section 2.2.8.1.1).

The syntax of a top-level error using JSON (application/json) is shown in the following ABNF Grammar for Top-level Error Payloads Formatted Using JSON listing.

```
stdErrorJson-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 MUST only be used in development environments and MUST
   ; NOT be present in a response from a production data service.

  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 Using 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 may 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 it 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 the new EntityType instance is to be inserted into.

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) or application/json (section 2.2.5.1.2).

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, then 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, then 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) or application/json (section 2.2.5.1.2).

Rules for processing the response to an InsertEntity Request (section 2.2.7.1.1) 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 it 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 resource(s) 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 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, then the media type(s) specified by the client in the Accept (section 2.2.5.1) header or the Format System Query Option ($format) (section 2.2.3.6.1.5) SHOULD include application/atom+xml (section 2.2.5.1.1) or application/json (section 2.2.5.1.2). If the request type is RetrieveComplexType Request, RetrievePrimitiveProperty or RetrieveLink, then the media type(s) specified in the Accept (section 2.2.5.1) header or the $format query string option SHOULD include application/xml or application/json (section 2.2.5.1.2). If the request is a RetrieveValue or RetrieveMediaResource Request, then the required media type(s) are data dependent and are not defined by this document. If the request is a RetrieveServiceDocument Request, then the required media types SHOULD include application/xml and application/atomsvc+xml. Finally, if the request is a RetrieveCount Request, then 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, then 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, then 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, then the media type specified by the client in the Content-Type header MUST be application/atom+xml (section 2.2.5.1.1) or application/json (section 2.2.5.1.2). If the request type is UpdateComplexType or UpdatePrimitiveProperty, then the media type specified MUST be application/xml or application/json (section 2.2.5.1.2). Finally, if the request

is an UpdateValue or UpdateMediaResource Request, then 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, 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 header MUST be a token 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 then the request MAY include an If-Match (section 2.2.5.5) header. If 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, then any of the Update Request Types (section 2.2.7.3) MAY be sent as a tunneled request (Tunneled Requests (section 2.2.7.7)).

Rules for processing responses to **Update requests** are specified in Request to Update Resources (section 3.1.4.4).

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 it 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 Type MAY be sent as a Tunneled Requests (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 is 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, then System Query Options and additional Resource Path segments may 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 MAY be sent as a tunneled request, see Tunneled Requests (section 2.2.7.7).

Rules for processing the response to Service Operation requests, are specified in and 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 it 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 (section 2.2.7.6).

Each operation sent within a Batch Request MUST follow the guidelines for the associated request types as specified in section 3.1.4.

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 a client MUST adhere to 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) 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) 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) 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 less 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, then the client should continue processing the request. If the version number in the header is higher than the highest version of this document that the client can interpret, then 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 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 may be used in a future request, 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 (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.

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 follows when processing requests. In addition, further directives specific to each request type are defined in the following sections.<61>

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 can 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, then the request should be processed as if the concurrency check succeeded.

Servers MAY<62> 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 follows 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 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, as specified in [RFC2616].

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, then the server should return a 4xx response code.

The execution of an HTTP request that is intended to change the state of a resource MAY<63> 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

In version 2.0 of the protocol defined by this specification, it is possible to map the value of a property on an EntityType to another location in the feed, as described in section 2.2.6.2.2.1.

If a property mapping that has been defined in the CSDL contains an FC_KeepInContent attribute with value "false", then 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 addressed by the request contain an FC_KeepInContent attribute with value "true", then the Data Service SHOULD respond with a 1.0 version response provided there is no other aspect of the response that 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, then the Data Service MUST respond with a 2.0 or greater version response.

If an EntityType instance included in a Data Service response contains a mapped property that has a value of null, then the property element for that property MUST be included in the <m:properties> element and the property element MUST 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, then the element being mapped to can 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) then 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 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 4xx 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 permit an empty value, the server MUST return a 4xx response code, as specified in Common Response Codes (section 3.2.8).

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<64> 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.

If the insert succeeds in full, the server MUST return a response with a 201 (Created) status code and a request body which 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, etc.). The server MAY alter the values of the resource received from the client before the resource is inserted on the server.

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, see UpdateEntity Request (section 2.2.7.3.1).

If the insert succeeds in full, the server MUST return a 204 (No Content) response code.

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 Entity Type instance which represents the Media Link Entry for the newly inserted Media Resource.

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.

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 Entity Type in the prior sentence is the Entity Type 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 version 2.0 of the protocol.

In addition to validating the request headers, the server MUST validate that the HTTP request URI and payload adheres to the syntax for Retrieve Request Types (section 2.2.7.2). The server should validate the request based on its knowledge of the Entity Data Model associated with the service. For example, the server should validate 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 formatted according to Error Response (section 2.2.8.1).

If the validation succeeds, the server MUST obtain the requested resource(s) and return a response (with a 2xx 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. This specification does not define an algorithm for determining if a full or partial set of entities is returned to the client.

Partial collections of entities are supported only in version 2.0 of the protocol defined by this specification. 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) and Entity Set (as a JSON array) (section 2.2.6.3.2). Such a response MUST also include a DataServiceVersion (section 2.2.5.3) response header indicating the response is using version 2.0 of this protocol. 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

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 CountRetrieveCount request, see RetrieveCount Request (section 2.2.7.2.9).

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 using an HTTP verb other than GET, then the data service MUST return a 4xx error response code.

RetrieveCount requests are supported only in version 2.0 of the protocol defined by this specification. A response to a RetrieveCount request MUST also include a DataServiceVersion (section 2.2.5.3) response header indicating the response is using version 2.0 of the protocol.

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 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 4xx 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 **MERGE HTTP** method, the request MUST be processed using a merge-based update, as specified in MERGE (section 2.2.4.1).

The **HTTP MERGE** and **HTTP PUT** methods are defined by this document to have identical semantics for UpdateProperty (section 2.2.7.3.3), UpdateLink (section 2.2.7.3.5), and UpdateValue (section 2.2.7.3.4) 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 using the **HTTP MERGE** method and the same request made 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 4xx 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 permit an empty value, the server MUST return a 4xx response code, as specified in Common Response Codes (section 3.2.8).<65>

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.

If the update succeeds in full, the server MUST return a 204 (No Content) response code.

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, then the server MUST rebind the EntityType instance being updated to the existing entities specified in the request payload. A rebind request may 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, EntityType instance property value, or Link. 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 delete the resource (entity or Link) identified by the request URI and return a response following the rules for the request type as specified in Retrieve Request Types (section 2.2.7.2).

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 Types (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 4xx 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 4xx response code MUST be returned. If the request URI is valid, then the server MUST invoke the FunctionImport, as specified in [MC-CSDL], associated with the Service Operation 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 4xx 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 4xx 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 must 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 SHOULD 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).

3.2.6 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 SHOULD use 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] should be consulted to determine the most appropriate response code. Additional details for each response code in this section are 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<66> 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 Insert examples 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 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: 1.0
```

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: 1.0

<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')/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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/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)" />
            <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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/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 Using the 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 from a data service.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
```

```
Content-Type: application/json
Content-Length: nnn
DataServiceVersion: 1.0

{"d": [
         {
            "__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(2)",
                            "type": "SampleModel.Order"
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }

         }
]}
```

4.2.1.3 Retrieve a Partial Collection of Entities 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. This example assumes that the server has a limit to return, at the most, two Order entities at a time and assumes that there exist more than two Order entities-associated Customer entities that are identified by the key values "ALFKI".

Request:

```
GET /service.svc/Customers('ALFKI')/Orders 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=feed
Content-Length: nnn
DataServiceVersion: 2.0

<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')/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/related/OrderLines"
          type="application/atom+xml;type=feed" title="OrderLines"
          href="Orders(1)/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/related/OrderLines"
          type="application/atom+xml;type=feed" title="OrderLines"
          href="Orders(2)/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=3" />
</feed>
```

4.2.1.4 Retrieve a Partial Collection of Entities Using the 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 from a data service.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 2.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
DataServiceVersion: 2.0

{"d": [
         {
            "__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(2)",
                            "type": "SampleModel.Order"
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }
         },
         "__next":
                "http://host/service.svc/Customers('ALFKI')/Orders?$skiptoken=3"
      ]
   }
```

4.2.1.5 Retrieve a Collection of Entities with an Inline Count Using 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 only supported in version 2.0 of the protocol defined by this specification.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders?$inlinecount=allpages HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 2.0
MaxDataServiceVersion: 2.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=feed
Content-Length: nnn
DataServiceVersion: 2.0

<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')/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)" />
          <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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(2)/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 Using 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 only supported in version 2.0 of the protocol defined by this specification.

Request:

```
GET /service.svc/Customers('ALFKI')/Orders?$inlinecount=allpages HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 2.0
MaxDataServiceVersion: 2.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
DataServiceVersion: 2.0

{"d": [
         {
            "__count": "2"
         },
         {
            "__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(2)",
                            "type": "SampleModel.Order",
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }

         }
]}
```

4.2.2 Retrieve a Single Entity Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to retrieve a Customer entity with an EntityKey value equal to "ALFKI" using the AtomPub Format.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

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: 1.0

<?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" 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')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/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:Address>
    <d:Version>AAAAAAAA+gE=</d:Version>
   <m:properties>
  </content>
</entry>
```

4.2.2.1 Retrieve a Single Entity with a Mapped Property 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 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 using web Customizable feeds attributes in the CSDL. One of the property mappings on the Employee EntityType has the FC_KeepInContent attribute set to "false" and, as a result, the following exchanges of messages is only supported in version 2.0 of the protocol defined by this specification.

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="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)" />
  <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: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 Using the JSON Format

The following example illustrates the exchange of messages required for a client to retrieve a Customer entity with EntityKey value equal to "ALFKI" using the JSON.

Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
ETag: W/"X'000000000000FA01'"
DataServiceVersion: 1.0

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": " Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
}
```

4.2.4 Retrieve a Single Entity and Its Directly Related Entities Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to retrieve a **Customer** entity (with EntityKey value equal to "ALFKI") and its associated **Order Entity Type** 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: 1.0
```

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: 1.0

<?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">
  <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">
    <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/related/OrderLines"
                type="application/atom+xml;type=feed" title="OrderLines"
                href="Orders(1)/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)" />
            <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/related/OrderLines"
                  type="application/atom+xml;type=feed" title="OrderLines"
                  href="Orders(2)/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:Address>
      <d:Version>AAAAAAAA+gE=</d:Version>
    </content>
  </entry>
```

4.2.5 Retrieve a Single Entity and Its Directly Related Entities Using the JSON Format

The following example illustrates the exchange of messages required for a client to retrieve a **Customer** entity (with EntityKey value equal to "ALFKI") and its associated **Order Entity Type** instances. This example uses the JSON format for all messages.

Request:

```
GET /service.svc/Customers('ALFKI')?$expand=Orders HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

Response:

```
HTTP/1.1 200 OK
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
```

```
DataServiceVersion: 1.0

{"d": {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA01\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": "Alfreds Futterkiste",
   "Address": { "Street": "57 Contoso St", "City": "Seattle" },
   "Version": "AAAAAAAA+gE=",
   "Orders": [
         {
            "__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(2)",
                            "type": "SampleModel.Order"
                          },
            "OrderID": 2,
            "ShippedDate": "\/Date(875836800000)\/",
            "Customer":   { "__deferred": { "uri": "Orders(2)/Customer" } }
            "OrderLines": { "__deferred": { "uri": "Orders(2)/OrderLines" } }

         }
   ]
} }
```

4.2.6 Retrieve a Data Service's Metadata Document (CSDL)

The following example illustrates the exchange of messages required for a client to obtain a description document of a data service, as specified in RetrieveServiceMetadata Request (section 2.2.7.2.6).

The following Data Service metadata document contains Customizable Feed property mappings that are supported only in version 2.0 of the protocol defined by this specification.

Request:

```
GET /service.svc/$metadata HTTP/1.1
Host: host
Accept: application/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/xml
```

```
Content-Length: nnn
DataServiceVersion: 2.0

<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<edmx:Edmx Version="1.0"
       xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<edmx:DataServices m:DataServiceVersion="2.0">
   <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" />
        <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>
      </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>
      <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 only supported in version 2.0 of the protocol defined by this specification.

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
```

4.3 Update an Existing Entity

4.3.1 Replace-Based Update Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to update an existing entity in a data service 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: 1.0

<?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/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 1.0
```

The following Retrieve request is not required to update the entity, but is included to show the value of the **Customer** entity after the update completed. Note that because replace-based semantics were used, property values that were not supplied in the **Update request** payload have been set to their default values (empty strings, in this case).

HTTP Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

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: 1.0

<?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">
  <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')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/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:Address>
    <d:Version>AAAAAAAA+gF=</d:Version>
   <m:properties>
  </content>
```

```
</entry>
```

4.3.2 Replace-Based Update Using the JSON Format

The following example illustrates the exchange of messages required for a client to update an existing entity in a data service using the JSON and replace-based update semantics.

HTTP Request:

```
PUT /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Content-Type: application/json
If-Match: W/"X'000000000000FA01'"
Accept: application/json
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0

{"d":
 {
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street" },
 }
}
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 1.0
```

The following Retrieve request is not required to update the entity, but is included to show the value of the **Customer** entity after the update completed. Note that because replace-based semantics were used, properties that were not supplied in the **Update request** payload have been set to their default values.

HTTP Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
```

```
DataServiceVersion: 1.0

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA02\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street", "City": "" },
   "Version": "AAAAAAAA+gF=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
 }
}
```

4.3.3 Merge-Based Update Using the AtomPub Format

The following example illustrates the exchange of messages required for a client to update an existing entity in a data service using the AtomPub Format and merge-based update semantics.

- HTTP Request:

```
MERGE /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: 1.0

<?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>
  </content>
</entry>
```

- HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
Content-Type: application/atom+xml;type=entry
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 1.0
```

The following Retrieve request is not required to update the entity, but is included to show the value of the **Customer** entity after the update completed. Note that because merge-based semantics were used, properties for which no values were supplied in the **Update request** payload have not been altered.

- HTTP Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/atom+xml
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

- 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: 1.0

<?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">
  <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')" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders"
        type="application/atom+xml;type=feed"
        title="Orders"
        href="Customers('ALFKI')/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:Address>
    <d:Version>AAAAAAAA+gF=</d:Version>
   <m:properties>
  </content>
</entry>
```

4.3.4 Merge-Based Update Using the JSON Format

The following example illustrates the exchange of messages required for a client to update an existing entity in a data service using the JSON and merge-based update semantics.

HTTP Request:

```
PUT /service.svc/Customers('ALFKI') HTTP/1.1
```

```
Host: host
Content-Type: application/json
If-Match: W/"X'000000000000FA01'"
Accept: application/json
Content-Length: nnn
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0

{"d":
 {
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street" }
 }
}
```

HTTP Response:

```
HTTP/1.1 204 No Content
Date: Fri, 12 Dec 2008 17:17:11 GMT
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 1.0
```

The following Retrieve request is not required to update the entity, but is included to show the value of the **Customer** entity after the update completed. Note that because merge-based semantics were used, properties for which no values were supplied in the **Update request** payload have not been altered.

HTTP Request:

```
GET /service.svc/Customers('ALFKI') HTTP/1.1
Host: host
Accept: application/json
DataServiceVersion: 1.0
MaxDataServiceVersion: 1.0
```

HTTP Response:

```
HTTP/1.1 200 OK
Date: Thurs, 4 Dec 2008 17:17:11 GMT
Content-Type: application/json
Content-Length: nnn
ETag: W/"X'000000000000FA02'"
DataServiceVersion: 1.0

{"d":
 {
   "__metadata": { "uri": "Customers(\'ALFKI\')",
                   "type": "SampleModel.Customer",
                   "etag": "W/\"X\'000000000000FA02\'\""
                 },
   "CustomerID": "ALFKI",
   "CompanyName": "Updated Company Name",
   "Address": { "Street": "Updated Street", "City": "" },
   "Version": "AAAAAAAA+gF=",
   "Orders":  { "__deferred": { "uri": "Customers(\'ALFKI\')/Orders" } }
```

```
        }
      }
```

4.4 Update the Relationship Between Two Entities

4.4.1 Update a Relationship Using the AtomPub 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" using the AtomPub Format.

HTTP Request:

```
PUT /service.svc/Order(1)/$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 Using the 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" using the AtomPub Format.

HTTP Request:

```
PUT /service.svc/Order(1)/$links/Customer HTTP/1.1
Host: host
Content-Type: application/json
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') 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)/$value"/>
    <m:properties>
       <d:DocumentID>300</d:DocumentID>
       <d:Author>Joe Smith</d:Author>
       <d:Title>Meeting Notes</d:Title>
    </m:properties>
    <link rel="edit-media"
          href="http://host/service.svc/Documents(300)/$value" />
    <link rel="edit"
          href="http://host/service.svc/Documents(300)" />
</entry>
```

4.6.2 Update a Media Resource

HTTP Request:

```
PUT /Documents(300) 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)/$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...
```

5 Security

5.1 Security Considerations for Implementers

Implementers of the protocol defined in this document should 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 five EntityTypes (Customer, Order, OrdersLine, Employee, and Document), each with **Primitive** type properties and some with NavigationProperties (which refer to the associations between the entities). 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 Entity Type instance exists with EntityKey value ALFKI.
- A total of 91 Customer Entity Type instances exist.
- An Employee Entity Type instance exists with EntityKey value 1.
- Two Order Entity Type 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 Entity Type 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 Entity Type 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/2007/06/edmx"
       xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<edmx:DataServices m:DataServiceVersion="2.0">
   <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" />

        <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>
      </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="NorthwindModel.Orders_Customers"
            FromRole="Customer" ToRole="Order" />
        </EntityType>

  <!-- The Employee EntityType has Web Customizable Feed property mappings that are supported
  only in version 2.0 of the protocol defined by this specification -->
        <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>


      <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"/>
      </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

This document specifies version-specific details in the Microsoft® .NET Framework. The following versions of .NET Framework are available in the following released Microsoft Windows® product or as supplemental software, see **.NET Framework**.

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® .NET Framework 3.5 Service Pack 1 (SP1)
- Microsoft® .NET Framework 4.0

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: A description of constructs that are supported only by version 2 of this protocol is specified in section 1.7.1. The following table illustrates versions of the protocol defined by this specification as supported by .NET versions.

	.NET Framework 3.5 SP1	**Data Services Update for .NET Framework 3.5 SP1, KB # 976127**	**.NET Framework 4.0**
OData version 1	x	x	x
OData version 2		x	x

<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 never generate key predicates of the form "<Entity Type property name> = <Entity Type property value>". Instead, the canonical form, as seen in 2.2.3.4, for single property EntityKeys, 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 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). .NET Framework 3.5 SP1 has no support 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 2.2.3.6.1.1.2: The data service server library supports all of the expressions.

<48> Section 2.2.3.7.2: The data service server library only includes the mimeType attribute if the developer authoring the service explicitly defines a media type for the property. By default, this attribute is not included on the definition of a property.

<49> 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 authoring the data service using the server library.

<50> 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.

<51> 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.

<52> 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.

<53> 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.

<54> Section 2.2.5.8: The data service client library and server library support verb tunneling.

<55> Section 2.2.6.2: The data service client library and server library use only the request and response messages defined in this document.

<56> Section 2.2.6.2.2: The data service client library and server library do not generate or parse self links.

<57> Section 2.2.6.3.3: The data service server library generates and parses the "__metadata" name/value pair.

<58> Section 2.2.6.4.1: The data service server library only alters the media type associated with an EntityType property if the developer authoring the service explicitly defines a media type for the property.

<59> 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 may cause side effects.

<60> Section 2.2.7.6: The data service client library does not support creating a Batch Request which includes both Query Operation (Batch Request (section 2.2.7.6) and Batch Request (section 2.2.7.6)) and Change Sets (Batch Request (section 2.2.7.6)).

<61> Section 3.2.5.1: The server library returns a 200 status instead of a 4xx when the entity exists and has no concurrency token defined and the client request includes the If-None-Match header.

<62> Section 3.2.5.1: The data service libraries do not implement any access control policies. Such policies may be applied by a higher or lower layer.

<63> 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 may cause side effects.

<64> Section 3.2.5.3.1: By default, the data service client library does not omit any constructs.

<65> Section 3.2.5.5: The server library returns a 500 response code rather than the required 4xx 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.

<66> 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

This section identifies changes that were made to the [MS-ODATA] protocol document between the May 2011 and June 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

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 or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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 **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
1.2 References	Added explanatory statement regarding the removal of the publishing year from Microsoft Open Specification document references.	N	Content updated.

9 Index

A

Abstract data model
  client 165
  message syntax 18
  server 170
Abstract type system 19
Accept HTTP header field
  application/atom+xml 81
  application/json 81
  overview 80
Applicability 13
AtomPub format
  additional representations 98
  collection of entities with Inline Count 184
  Complex Type property 94
  deferred content
    inline representation 95
    overview 95
  EDMSimpleType property 95
  Entity Set element 89
  Entity Type element 91
  merge-based update 198
  Navigation property 95
  overview 89
  replace-based update 195
  retrieving collection of entities 180
  retrieving single entity (section 4.2.1.3 182, section 4.2.2 187)
  retrieving single entity and related entities 189
  service document 97
  updating relationship 201

B

Batch request
  Change Set syntax 153
  example (section 2.2.7.6.5 156, section 4.5 202)
  example response (section 2.2.7.6.8 159, section 4.5 202)
  executing received 178
  HTTP request restrictions 154
  overview 152
  Query Operation syntax 154
  response syntax 158
  responses 157
  syntax 154

C

Canonical URIs 78
Capability negotiation 14
Change Set syntax
  overview 153
  referencing requests 154
Change tracking 214
Client
  abstract data model 165
  higher-layer triggered events
    common rules for all requests 165
    overview 165
    request to delete resources 168
    request to insert resources 165
    request to invoke service operation 169
    request to retrieve resources 166
    request to send batch of operations 169
    request to update resources 167
  initialization 165
  local events 170
  message processing
    receiving responses to data service requests 169
    receiving responses to insert requests 170
  sequencing rules
    receiving responses to data service requests 169
    receiving responses to insert requests 170
  timer events 170
  timers 165
Collection of entities
  retrieving count 194
Common expression syntax
  binary numeric promotions 59
  construction and evaluation 40
  lifted operators 60
  numeric promotions for method call parameters 61
  operator precedence 58
  overview 37
  unary numeric promotions 59
Complex Type property
  AtomPub format 94
  JSON format 106
  XML format 113
Construction rules 28
Content-Type HTTP header field 81
CSDL document 206

D

Data model
  abstract
    client 165
    server 170
  sample Entity 206
Data service metadata
  Conceptual Schema Definition Language document for data services 71
  service document 71
Data service request
  receiving responses to 169
  rules for receiving 171
DataServiceVersion HTTP header field 82
Delete request types
  DeleteEntity 147
  DeleteLink 148

DeleteValue 149
executing received request 177
rules for executing received request 171
sending request 168

E

EDMSimpleType property
AtomPub format 95
JSON format 108
raw format 113
XML format 114
Entity data model 206
Entity Set element
AtomPub format 89
JSON format 101
Entity Type element
AtomPub format 91
JSON format 103
ETag HTTP header field 83
Examples
inserting new entity 180
overview 180
retrieving resources
collection of entities 180
collection of entities with Inline Count using AtomPub format 184
collection of entities with Inline Count using JSON format 186
data service's metadata document 192
partial collection of entities using JSON format 183
single entity and related entities using AtomPub format 189
single entity and related entities using JSON format 191
single entity using AtomPub format (section 4.2.1.3 182, section 4.2.2 187)
single entity using JSON format 189
the count of collection of entities 194
updating existing entity
merge-based update using AtomPub format 198
merge-based update using JSON format 199
replace-based update using AtomPub format 195
replace-based update using JSON format 197
updating relationship between two entities
deleting existing entity 202
using AtomPub format 201
using JSON format 201

F

Fields - vendor-extensible 17

G

Glossary 8

H

Higher-layer triggered events
client
common rules for all requests 165
overview 165
request to delete resources 168
request to insert resources 165
request to invoke service operation 169
request to retrieve resources 166
request to send batch of operations 169
request to update resources 167
server 171
HTTP header fields
Accept 80
Content-Type 81
DataServiceVersion 82
ETag 83
If-Match 84
If-None-Match 84
MaxDataServiceVersion 85
overview 79
X-HTTP-Method 86
HTTP methods
MERGE 79
overview 79
HTTP request restrictions 154

I

If-Match HTTP header field 84
If-None-Match HTTP header field 84
Implementer - security considerations 205
Index of security parameters 205
Informative references 12
Initialization
client 165
server 171
Insert examples 180
Insert request types
InsertEntity
examples 119
executing received request 173
overview 117
sending request 166
InsertLink
executing received request 174
overview 124
sending request 166
overview 116
receiving responses to requests 170
rules for executing received request (section 3.2.5.2 171, section 3.2.5.3 173)
UpdateEntity - executing received request 177
Introduction 8
Invoke request
executing received 177
overview 150

J

JavaScript Object Notation (JSON) format
collection of Complex Type instances 107
collection of EDMSimpleType values 108

common serialization rules for all EDM constructs 99
Complex Type property 106
deferred content
inline representation 110
overview 109
EDMSimpleType property 108
Entity Set element 101
Entity Type element 103
error response 163
links 110
merge-based update 199
Navigation property 108
overview 98
replace-based update 197
retrieving collection of entities 181
retrieving single entity 189
retrieving single entity and related entities 191
service document 112
updating relationship 201
JSON format
collection of entities with Inline Count 186
retrieving partial collection of entities 183

L

Local events
client 170
server 178

M

MaxDataServiceVersion HTTP header field 85
Media resource
insert new 202
query existing 203
update 203
MERGE HTTP method 79
Message processing
client
receiving responses to data service requests 169
receiving responses to insert requests 170
server
executing received Batch request 178
executing received data service request 171
executing received Delete request 177
executing received Insert request 173
executing received Invoke request 177
executing received Retrieve request 174
executing received Update request 176
rules for receiving data service requests 171
Messages
syntax
abstract data model 18
abstract type system 19
common payload 86
HTTP header fields 79
HTTP methods 79
overview 18
request types 116
resource addressing rules 23
response types 161
transport 18

N

Navigation property
AtomPub format 95
JSON format 108
Normative references 10
Numeric promotions
binary 59
for method call parameters 61
unary 59

O

Overview (synopsis) 12

P

Parameters - security index 205
Payload syntax
AtomPub format 89
common serialization rules for XML-based formats 87
JSON format 98
overview 86
raw format 113
XML format 113
Preconditions 13
Prerequisites 13
Product behavior 210

Q

Query Operation syntax 154
Query options
custom 70
overview 35
service operation parameters 70
system
common expression syntax 37
evaluating 62
expanding 63
filter 64
format 64
OrderBy 65
overview 35
skip 65
top 66

R

Raw format
EDMSimpleType property 113
overview 113
References
informative 12
normative 10
Relationship to other protocols 13
Request types
Batch 152

delete 147
insert 116
Invoke 150
overview 116
retrieve 127
tunneled 161
update 139
Requests - client
common rules 165
deleting resources
common rules for sending all Delete requests 168
overview 168
inserting resources
overview 165
sending InsertEntity request 166
sending InsertLink request 166
invoking service operation 169
retrieving resources
common rules for sending Retrieve requests 166
overview 166
sending batch of operations 169
updating resources
common rules for sending Update requests 167
overview 167
Resource addressing rules
canonical URIs 78
data service metadata 71
overview 23
query options 35
resource path 27
service root 27
URI equivalence 78
URI syntax 24
Resource path 27
Response codes 178
Response types - error
JSON 163
overview 161
XML 162
Retrieve request examples
collection of entities with Inline Count using AtomPub format 184
collection of entities with Inline Count using JSON format 186
partial collection of entities using JSON format 183
retrieving collection of entities
using AtomPub format 180
using JSON format 181
retrieving data service's metadata document 192
retrieving single entity and related entities using AtomPub format 189
retrieving single entity and related entities using JSON format 191
single entity using AtomPub format (section 4.2.1.3 182, section 4.2.2 187)
single entity using JSON format 189
the count of collection of entities 194
Retrieve request types
executing received request
overview 174
RetrieveValue 175
RetrieveComplexType 130
RetrieveEntity 128
RetrieveEntitySet 127
RetrieveLink 136
RetrievePrimitiveProperty 131
RetrieveServiceDocument 134
RetrieveServiceMetadata 133
RetrieveValue 132
sending request 166

S

Security
parameter index 205
Security - implementer considerations 205
Semantics 30
Sequencing rules
client
receiving responses to data service requests 169
receiving responses to insert requests 170
server
executing received Batch request 178
executing received data service request 171
executing received Delete request 177
executing received Insert request 173
executing received Invoke request 177
executing received Retrieve request 174
executing received Update request 176
rules for receiving data service requests 171
Serialization rules
EDM constructs 99
XML formats 87
Server
abstract data model 170
higher-layer triggered events 171
initialization 171
local events 178
message processing
executing received Batch request 178
executing received data service request 171
executing received Delete request 177
executing received Insert request 173
executing received Invoke request 177
executing received Retrieve request 174
executing received Update request 176
rules for receiving data service requests 171
response codes 178
sequencing rules
executing received Batch request 178
executing received data service request 171
executing received Delete request 177
executing received Insert request 173
executing received Invoke request 177
executing received Retrieve request 174
executing received Update request 176
rules for receiving data service requests 171
timer events 178
timers 170

Service root 27
Standards assignments 17
Syntax
  abstract data model 18
  abstract type system 19
  common payload 86
  HTTP header fields 79
  HTTP methods 79
  overview 18
  request types 116
  resource addressing rules 23
  response types 161

T

Timer events
  client 170
  server 178
Timers
  client 165
  server 170
Tracking changes 214
Transport 18
Triggered events - higher-layer
  client
    common rules for all requests 165
    overview 165
    request to delete resources 168
    request to insert resources 165
    request to invoke service operation 169
    request to retrieve resources 166
    request to send batch of operations 169
    request to update resources 167
  server 171
Tunneled request type 161

U

Update request examples
  merge-based update using AtomPub format 198
  merge-based update using JSON format 199
  replace-based update using AtomPub format 195
  replace-based update using JSON format 197
Update request types
  rules for executing received request (section 3.2.5.2 171, section 3.2.5.5 176)
  sending request 167
  UpdateComplexType 141
  UpdateEntity
    example 141
    overview 139
  UpdateLink 145
  UpdatePrimitiveProperty 142
  UpdateValue 143
Updating relationship examples
  deleting existing entity 202
  using AtomPub format 201
  using JSON format 201
URI format
  canonical URIs 78
  data service metadata 71
  equivalence 78
  overview 23
  query options 35
  resource path 27
  service root 27
  syntax 24

V

Vendor-extensible fields 17
Versioning 14

X

X-HTTP-Method HTTP header field 86
XML format
  collection of Complex Type instances 114
  collection of EDMSimpleType values 114
  Complex Type property 113
  EDMSimpleType property 114
  error response 162
  links 114
  overview 113