[MS-VROOM]:

Versioned RESTful Open Object Model Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

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

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

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

Support. For questions and support, please contact dochelp@microsoft.com.

Preliminary Documentation. This particular Open Specifications document provides documentation for past and current releases and/or for the pre-release version of this technology. This document provides final documentation for past and current releases and preliminary documentation, as applicable and specifically noted in this document, for the pre-release version. Microsoft will release final documentation in connection with the commercial release of the updated or new version of this technology. Because this documentation might change between the pre-release version and the final

version of this technology, there are risks in relying on this preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

Revision Summary

Date	Revision History	Revision Class	Comments
7/24/2018	0.1	New	Released new document.
10/1/2018	1.0	Major	Significantly changed the technical content.
7/20/2021	1.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Versioned RESTful Open Object Model Protocol allows a protocol client to use common web technologies to access data and perform operations on a protocol server. The actions to be executed are sent by a protocol client as part of a request, and the results are returned by a protocol server as part of a response.

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

1.1 Glossary

This document uses the following terms:

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

entity type: A type that represents the structure of a top-level concept, such as a customer or order, in a conceptual model.

HTTP method: In an HTTP message, a token that specifies the method to be performed on the resource that is identified by the Request-URI, as described in [RFC2616].

Hypertext Transfer Protocol (HTTP): An application-level protocol for distributed, collaborative, hypermedia information systems (text, graphic images, sound, video, and other multimedia files) on the World Wide Web.

Hypertext Transfer Protocol Secure (HTTPS): An extension of HTTP that securely encrypts and decrypts web page requests. In some older protocols, "Hypertext Transfer Protocol over Secure Sockets Layer" is still used (Secure Sockets Layer has been deprecated). For more information, see [SSL3] and [RFC5246].

JavaScript Object Notation (JSON): A text-based, data interchange format that is used to transmit structured data, typically in Asynchronous JavaScript + XML (AJAX) web applications, as described in [RFC7159]. The JSON format is based on the structure of ECMAScript (Jscript, JavaScript) objects.

octet: A group of 8 bits often referred to as a byte.

Open Data Protocol (OData): A web protocol for querying and updating data specified in the OData protocol.

website: A group of related pages and data within a SharePoint site collection. The structure and content of a site is based on a site definition. Also referred to as SharePoint site and site.

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

XML schema: A description of a type of XML document that is typically expressed in terms of constraints on the structure and content of documents of that type, in addition to the basic syntax constraints that are imposed by XML itself. An XML schema provides a view of a document type at a relatively high level of abstraction.

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

1.2 References

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

1.2.1 Normative References

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

[DOCS-ODRESTAPI] Microsoft, "Work with files, lists, and sites in SharePoint", https://docs.microsoft.com/en-us/onedrive/developer/rest-api/?view=odsp-graph-2019

[ISO8601] ISO, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO 8601:2004, December 2004, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40874

Note There is a charge to download the specification.

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

[OData-Protocol] OASIS, "OData Version 4.0 Part 1: Protocol", OASIS Standard, http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.doc

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

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

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt

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

[RFC7230] Fielding, R., and Reschke, J., Eds., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014, http://www.rfc-editor.org/rfc/rfc7230.txt

1.2.2 Informative References

None.

1.3 Overview

This protocol enables a protocol client to send an **HTTPS** request to a protocol server using common web technologies to access data and perform operations.

This protocol defines two roles: protocol client and protocol server. A protocol client initiates communication by generating an HTTPS request. The protocol client then sends that HTTPS request to the protocol server for processing. The protocol server processes the HTTPS request and returns an HTTPS response.

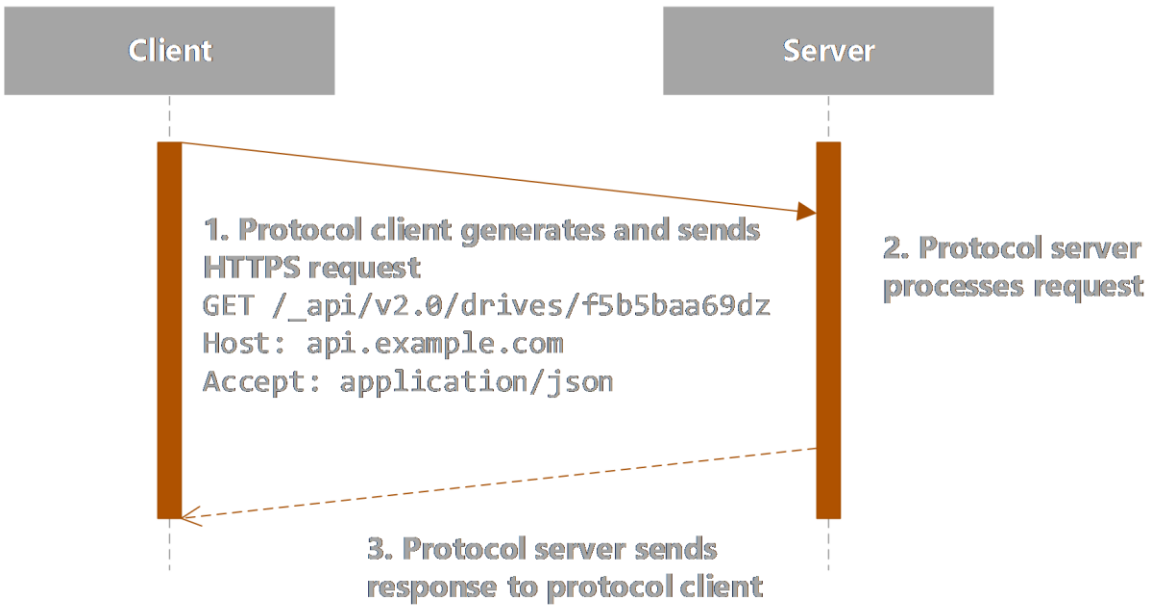

Figure 1: Overview of a request/response sequence

1.4 Relationship to Other Protocols

This protocol uses the **Open Data Protocol (OData)** for formatting request and response messages in **JSON**, as described in [OData-Protocol] and [ODataJSON4.0]. It transmits those messages by using **Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS)**, as described in [RFC2818] and [RFC7230]. The OneDrive and SharePoint VROOM Protocol described in [DOCS-ODRESTAPI] defines the properties and methods that are exposed to protocol clients through this protocol.

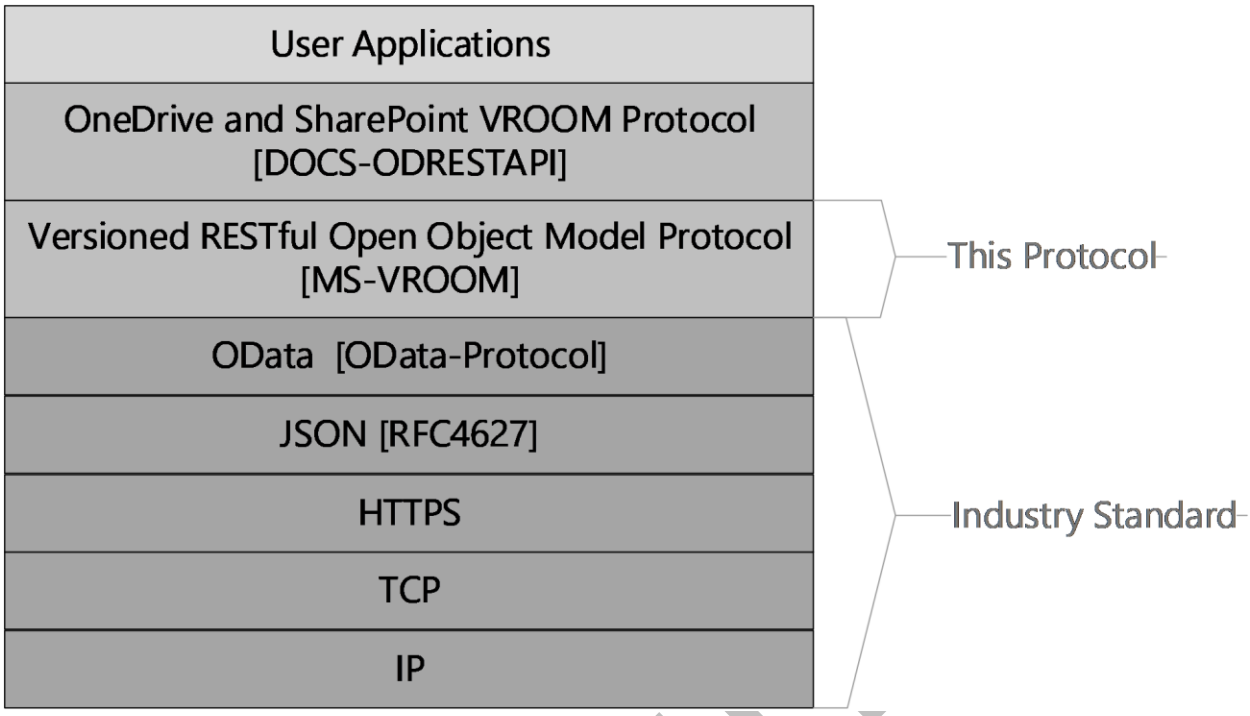

Figure 2: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

This protocol operates against a protocol server endpoint that is identified by a URL that is known by protocol clients. The protocol assumes that authentication has been performed by the underlying protocols.

1.6 Applicability Statement

This protocol can be used to perform create, retrieve, update, and delete operations using common web technologies against a protocol server that implements the OneDrive and SharePoint VROOM Protocol as described in [DOCS-ODRESTAPI].

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** This protocol uses **HTTPS** as a transport. For more information, see section 2.1.
- **Protocol Versions:** Currently, there is only one version of this protocol. The version is specified as a segment in the URL for the protocol server endpoint.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

Protocol servers MUST support **HTTPS**, as specified in [RFC2818]. Messages sent between protocol clients and protocol servers MUST be formatted as HTTP messages, as specified in [RFC7230], section 3. Protocol clients MUST use the DELETE, GET, HEAD, PATCH, POST, or PUT method to send messages to protocol servers. A message containing structured data MUST be formatted as a JSON object, as specified in [ODataJSON4.0], section 4. A message containing unstructured data MUST be formatted as a stream of **octets**.

2.2 Common Data Types

This section contains common definitions that are used by this protocol. The syntax of the definitions uses **JavaScript Object Notation (JSON)**, as defined in [RFC4627].

2.2.1 Namespaces

This specification does not define any common **XML namespaces**.

2.2.2 HTTP Methods

This specification does not define any custom **HTTP methods**.

2.2.3 HTTP Headers

This specification does not define any custom **HTTP** headers.

2.2.4 URI Parameters

The following table summarizes the set of common URI parameters used by this protocol.

URI Parameter	Description
$select	Comma-separated list of properties to include in the response, as specified in [OData-Protocol], section 11.2.4.1.
$expand	Comma-separated list of related **entity types** to include in the response, as specified in [OData-Protocol], section 11.2.4.2.
$orderby	Comma-separated list of properties used to sort the items in the response, as specified in [OData-Protocol], section 11.2.5.2.
$top	The number of items to return in a response, as specified in [OData-Protocol], section 11.2.5.3.

2.2.5 Elements

This specification does not define any common **XML schema** element definitions.

2.2.6 Complex Types

The following table summarizes the set of complex type definitions defined by this specification.

Complex type	Description
VROOM collection	Any of the following: A **Collection of Primitive Values** as specified in [ODataJSON4.0], section 7.3. A **Collection of Complex Values** as specified in [ODataJSON4.0], section 7.4. A **Collection of Entities** as specified in [ODataJSON4.0] , section 12.
VROOM object	Either of the following: An **Entity** as specified in [ODataJSON4.0], section 6. A **Complex Value** as specified in [ODataJSON4.0], section 7.2.
VROOM error	An **Error Response** as specified in [ODataJSON4.0], section 19.

2.2.6.1 Vroom Collection

The VROOM collection type is used for OData instances of Collection of Primitive Values, Collection of Complex Values, and Collection of Entities, as specified in [ODataJSON4.0].

If the elements in the VROOM collection are **OData primitive values**, the VROOM collection formats as a Collection of Primitive Values as specified in [ODataJSON4.0], section 7.3.

If the elements in the VROOM collection are **OData complex values**, the VROOM collection formats as a Collection of Complex Values as specified in [ODataJSON4.0], section 7.4.

If the VROOM collection represents an **OData entity set**, the VROOM collection formats as a Collection of Entities as specified in [ODataJSON4.0], section 12.

2.2.6.2 VROOM Object

The VROOM object type is used for OData Entity and Complex Value instances as specified in [ODataJSON4.0].

If the VROOM object represents an OData entity, the VROOM object formats as an Entity as specified in [ODataJSON4.0], section 6.

Otherwise, the VROOM object represents an OData complex type and is formatted as a Complex Value as specified in [ODataJSON4.0], section 7.2.

2.2.6.3 VROOM Error

The VROOM error is sent to the protocol client by the protocol server when there is an error while the protocol server processes the request. It formats as an OData Error Response, as specified in [ODataJSON4.0], section 19. The values of the properties in the VROOM error are defined in the Error responses section of [DOCS-ODRESTAPI].

2.2.7 Simple Types

The following table summarizes the set of simple types used by this specification.

Simple type	Description
VROOM Boolean	A JSON Boolean as described in [RFC4627], conforming to the OData **Edm.Boolean** format as specified in [ODataJSON4.0], section 7.1.
VROOM Number	A JSON number as described in [RFC4627], conforming to any of the OData **Edm.Byte**, **Edm.SByte**, **Edm.Int16**, **Edm.Int32**, **Edm.Int64**, **Edm.Single**, **Edm.Double**, **Edm.Decimal** formats as specified in [ODataJSON4.0], section 7.1.
VROOM String	A JSON string as described in [RFC4627], conforming to the OData **Edm.String** format as specified in [ODataJSON4.0], section 7.1.
VROOM DateTime	A JSON string as described in [RFC4627], conforming to the OData **Edm.DateTimeOffset** format as specified in [ODataJSON4.0], section 7.1, and also conforming to the date and time format specified in [ISO8601].
VROOM Null	A JSON literal null as described in [RFC4627].

2.2.7.1 VROOM Boolean

The VROOM Boolean simple type represents a "true" or "false" value. It is formatted as a JSON literal "true" or "false" as specified in [RFC4627] and conforms to the **Edm.Boolean** format as specified in [ODataJSON4.0], section 7.1.

2.2.7.2 VROOM Number

The VROOM Number simple type represents a numeric value. It is formatted as a JSON number as specified in [RFC4627] and conforms to the OData **Edm.Byte**, **Edm.SByte**, **Edm.Int16**, **Edm.Int32**, **Edm.Int64**, **Edm.Single**, **Edm.Double**, or **Edm.Decimal** formats as specified in [ODataJSON4.0], section 7.1.

2.2.7.3 VROOM String

The VROOM String simple type represents a string value. It is formatted as a JSON string as specified in [RFC4627] and conforms to the **Edm.String** format as specified in [ODataJSON4.0], section 7.1.

2.2.7.4 VROOM DateTime

The VROOM DateTime simple type represents a date and time value. It is formatted as a JSON string as described in [RFC4627], conforming to the **Edm.DateTimeOffset** format as specified in [ODataJSON4.0], section 7.1. The value MUST further conform to the date and time format specified in [ISO8601].

2.2.7.5 VROOM Null

The VROOM Null simple type is a special value that can be used in place of any other value to indicate that no instance is specified. It is represented as a JSON literall null as described in [RFC4627].

3 Protocol Details

3.1 Server Details

This protocol allows protocol servers to perform implementation-specific authorization checks and to notify protocol clients of authorization faults by using either HTTP status codes or VROOM Error responses. Except where specified otherwise, protocol clients SHOULD interpret HTTP status codes as specified in [RFC2616] section 10 and VROOM Error responses as specified in the Error responses section of [DOCS-ODRESTAPI].

3.1.1 Abstract Data Model

This section describes the relationship between the data types specified in this protocol and the abstract data model specified in [OData-Protocol], section 3, which uses the **Entity Data Model (EDM)** as its data modeling vocabulary, which is specified in [MC-CSDL], section 2. The described relationship is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as longa s their external behavior is consistent with that described in this document.

The following states specify the data model that a protocol server maintains to participate in this protocol:

- **VROOM object type:** A VROOM Object type is mapped to an **EDM entity type** ([MC-CSDL], section 2.1.2) or an **EDM complex type** ([MC-CSDL], section 2.1.7). If the VROOM object has a unique identity, an independent existence, and forms the operational unit of consistency, then it corresponds to an **EDM entity type**. Otherwise, it corresponds to an **EDM complex type**. For each VROOM property in the VROOM Object type, if the type of the VROOM property can be mapped to **EDM complex type**, **EDM simple type** ([MC-CSDL], section 2.2.1), collection of **EDM complex type**, or collection of **EDM simple type**, then the VROOM property is mapped to an **EDM property** ([MC-CSDL], section 2.1.3) in the corresponding **EDM entity type** or **EDM complex type**. If the type of the VROOM property can be mapped to **EDM entity type** or **EDM entity set** ([MC-CSDL], section 2.1.18), then the VROOM property is mapped to an **EDM navigation property** ([MC-CSDL], section 2.1.4) in the corresponding **EDM entity type** or **EDM complex type**.
- **VROOM Collection:** A VROOM Collection is mapped to a collection of **EDM simple type** values, a collection of **EDM complex type** values, or an **EDM entity set**. If the elements in the VROOM Collection correspond to instances of **EDM simple type**, then the VROOM Collection corresponds to a Collection of **EDM simple type**. If the elements in the VROOM Collection correspond to instances of **EDM complex type**, then the VROOM Collection corresponds to a Collection of **EDM complex type** values. If the elements in the VROOM Collection correspond to instances of **EDM entity** type, then the VROOM Collection corresponds to an **EDM entity set**.
- **VROOM Null:** A VROOM null is mapped to the **EDM simple type null** ([MC-CSDL], section 2.2.1.1.1).
- **VROOM Property:** A VROOM property is mapped to an **EDM property** or **EDM navigation property.**
- **VROOM Method:** A VROOM method is mapped to an **EDM function import** ([MC-CSDL], section 2.1.15).

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

The following table summarizes the operations of this protocol.

Operation	Description
ProcessVroomRequest	Process the **OData** request message that contains the resource path, **OData** query options, and **OData** payload, and respond with an **OData** response message.

3.1.5.1 ProcessVroomRequest

During this operation, the protocol server receives an OData request that contains the resource path, OData query options (as specified in [OData-Protocol] section 11.2), and OData payload (as specified in [OData-Protocol] section 11.4); processes the request; and then responds with an OData response ([OData-Protocol] section 9).

The protocol server endpoint for this operation is formed by appending "/_api/v2.0" to the URL of the **Web site**, for example "https://api.example.com/repository/_api/v2.0".

The resource path and OData query options are appended to the endpoint; for example "https://api.example.com/repository/_api/v2.0/<resourcePath>?<ODataQueryOptions>".

The resource path contained in the request message MUST conform to the following format:

```
resourcePath            = resourcePathAlias "/" resourceMembers [addressByPath] /
                          resourceRoot "/" resourceMembers [addressByPath]
resourcePathAliasValue  = resourceRoot /
                          resourceRoot "/" resourceMembers
addressByPath           = ":/" domainSpecificPath [ ":/" resourceMembers ]
resourceRoot            = entitySetName /
                          singletonEntity
entitySetName           = vroomIdentifier
singletonEntity         = vroomIdentifier
methodParameters        = parameterValue /
                          namedParameterValueList
parameterNameValuePair  = parameterName "=" parameterValue
namedParameterValueList = parameterNameValuePair /
                          parameterNameValuePair "," namedParameterValueList
resourceMembers         = resourceMember /
                          resourceMember "/" resourceMembers
resourceMember          = instancePropertyGet /
                          instanceMethodInvoke
instancePropertyGet      = propertyName
instanceMethodInvoke    = namespace "." methodName [ "(" methodParameters ")" ]
methodName              = vroomIdentifier ; VROOM method name
propertyName            = vroomIdentifier ; VROOM property name
parameterName           = vroomIdentifier ; VROOM parameter name
resourcePathAlias       = *PCHAR ; resource path alias
parameterValue          = PrimitiveLiteral ; OData-Protocol OData ABNF Construction Rules
```

```
namespace               = namespacePart *( "." namespacePart )
namespacePart           = vroomIdentifier
vroomIdentifier         = identifierLeadCharacter *127identifierCharacter
identifierLeadCharacter = ALPHA
identifierCharacter     = ALPHA / "_" / DIGIT
domainSpecificPath      = 1*pcharNoColon *("/" 1*pcharNoColon)
pcharNoColon            = UCHAR / "@" / "&" / "=" / "+"
```

where the **resourcePathAlias** is a string whose value is **resourcePathAliasValue**.

The protocol client sends a **VroomRequest** message, and the protocol server responds with a **VroomResponse** message, as follows.

1. The protocol client sends a **VroomRequest** message that contains the resource path, OData query options, and OData payload.

2. The protocol server receives the **VroomRequest** message, processes the resource path to locate the VROOM object, performs actions against the VROOM object, and responds to the protocol client with a **VroomResponse** response message. If an error occurs while the protocol server processes the request, the response contains an error message that provides details about the error.

3.1.5.1.1 Message

The following message definitions are specific to this operation.

The **VroomRequest** message is the request message for the **ProcessVroomRequest** Web service method. The message contains the resource path, OData query options, and OData payload.

The **VroomResponse** message is the OData response ([OData-Protocol] section 9) message for the **ProcessVroomRequest** Web service method.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Client Details

The client side of this protocol is simply a pass-through. That is, no additional timers or other state is required on the client side of this protocol. Calls made by the higher-layer protocol or applications are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.2.1 Abstract Data Model

None.

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

None.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

The following examples illustrate how a protocol client retrieves and updates data on a protocol server by using this protocol. The example scenario is a book store that has a collection of books. Each book within that collection has an identifier, title, author, and content.

In this scenario, the following VROOM types are defined on the protocol server:

VROOM type	VROOM type name	Description	Properties	Methods
BookCollection	SampleCode.BookCollection	A VROOM Object type that defines a collection of books within a catalog.	None.	The **BookCollection** type supports **Requesting Data** as specified in [OData-Protocol] section 11.2 to retrieve a book from the collection. The **BookCollection** type supports the **OData Create an Entity** request as specified in [OData-Protocol] section 11.4.2 to add a book to the collection.
Book	SampleCode.Book	A VROOM Object type that defines a specific book in a collection of books.	**author** – A VROOM String that contains the name of the author of the book. **id** – A VROOM String value that contains the identifier for the book. **publishDate** – a VROOM DateTime value that contains the date when the book was published. **title** – A VROOM String that contains the title of the book. **content** – A VROOM Stream that contains the raw content of the book.	**checkOut** – Checks out the book for a user. This method has only one parameter, which is the user's name. The return value of the **CheckOut** method is the date that the book is due. The **Book** type supports the **Update an Entity** request as specified in [OData-Protocol] section 11.4.3 to update information about the book.

In the examples, the catalog contains the following books:

Identifier	Title	Author	Publish date
M!3387ac63	How to Cook Chinese Food	Soha Kamal	March 1, 2008
F6a265,ab	How to Cook Japanese Food	Soha Kamal	May 5, 2007
Z2e80eb25	Best Recipes	Lisa Andrews	January 3, 2009
35.704655a3	Family Recipes	Patrick Hines	December 1, 2005

4.1 Retrieve Book Information

In this example, a protocol client requests information about a book by using the book identifier "M!3387ac63".

The protocol client sends the following message including some HTTP headers:

```
GET /_api/v2.0/books/M!3387ac63 HTTP/1.1
Host: api.example.com
Accept: application/json
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "M!3387ac63",
  "author": "Soha Kamal",
  "publishDate": "2008-03-01T00:00:000.000Z",
  "title": "How to Cook Chinese Food"
}
```

4.2 Retrieve Books by a Specific Author

In this example, a protocol client requests information about all of the books that are authored by Soha Kamal.

The protocol client sends the following message including some HTTP headers:

```
GET /_api/v2.0/books?$filter=author%20eq%20'Soha%20Kamal' HTTP/1.1
Host: api.example.com
Accept: application/json
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "value": [
    {
      "id": "M!3387ac63",
      "author": "Soha Kamal",
```

```
      "publishDate": "2008-03-01T00:00:000.000Z",
      "title": "How to Cook Chinese Food"
    },
    {
      "id": "F6a265,ab",
      "author": "Soha Kamal",
      "publishDate": "2007-05-05T00:00:000.000Z",
      "title": "How to Cook Japanese Food"
    }
  ]
}
```

4.3 Update Book Information

In this example, a protocol client sends a request to update author information for the book that is associated with the identifier "Z2e80eb25".

The protocol client sends the following message including some HTTP headers:

```
PATCH /_api/v2.0/books/Z2e80eb25 HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "author": "Lisa Andrews"
}
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 200 OK
Content-Type: application/json


{
  "id": "Z2e80eb25",
  "author": "Lisa Andrews",
  "publishDate": "2009-01-03T00:00:000.000Z",
  "title": "Best Recipes"
}
```

4.4 Add a Book to a Catalog

In this example, the protocol client submits a request to add a book to the catalog.

The protocol client sends the following message including some HTTP headers:

```
POST /_api/v2.0/books HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "author": "Neil Black",
  "title": "Simple Cookbook",
  "publishDate": "2009-08-01T00:00:00.000Z"
}
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": "fdda44df",
  "author": "Neil Black",
  "title": "Simple Cookbook",
  "publishDate": "2009-08-01T00:00:00.000Z"
}
```

4.5 Unsuccessfully Add a Book to a Catalog

In this example, the protocol client submits a request to add a book to the catalog. The protocol server returns an error because the book already exists int eh catalog.

The protocol client sends the following message including some HTTP headers:

```
POST /_api/v2.0/books HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "author": "Lisa Andrews",
  "title": "Best Recipes",
  "publishDate": "2009-01-03T00:00:00.000Z"
}
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 409 Conflict
Content-Type: application/json

{
  "error": {
    "code": "nameAlreadyExists",
    "message": "A book with the same title already exists in the catalog.",
  }
}
```

4.6 Retrieve Book Content

In this example, the protocol client submits a request to get the content of the book associated with the identifier "M!3387ac63".

The protocol client sends the following message including some HTTP headers:

```
GET / api/v2.0/books/M!3387ac63/content HTTP/1.1
Host: api.example.com
Accept: */*
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream


Sample content of book How to Cook Chinese Food.
```

4.7 Update Book Content

In this example, the protocol client submits a request to update the content of the book associated with the identifier "M!3387ac63".

The protocol client sends the following message including some HTTP headers:

```
PUT /_api/v2.0/books/M!3387ac63/content HTTP/1.1
Host: api.example.com
Content-Type: text/plain

New Sample Content of book
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 204 No Content
```

4.8 Check Out a Book

In this example, the protocol client submits a request to check out the book that is associated with the identifier "M!3387ac63".

The protocol client sends the following message including some HTTP headers:

```
POST / api/v2.0/books/M!3387ac63/SampleCode.checkOut HTTP/1.1
Host: api.example.com
Content-Type: application/json

{
  "user": "Sam Bruce"
}
```

The protocol server responds with the following message including some HTTP headers.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "dueDate": "2019-01-01T00:00:00.000Z"
}
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Microsoft SharePoint Server 2019
- Microsoft SharePoint Server Subscription Edition Preview

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index

V