Repeatable Requests Version 1.0

Committee Specification 01

07 July 2020

This stage:
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/cs01/repeatable-requests-v1.0-cs01.docx (Authoritative)
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/cs01/repeatable-requests-v1.0-cs01.html
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/cs01/repeatable-requests-v1.0-cs01.pdf

Previous stage:
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/csprd01/repeatable-requests-v1.0-csprd01.docx (Authoritative)
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/csprd01/repeatable-requests-v1.0-csprd01.html
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/csprd01/repeatable-requests-v1.0-csprd01.pdf

Latest stage:
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/repeatable-requests-v1.0.docx (Authoritative)
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/repeatable-requests-v1.0.html
https://docs.oasis-open.org/odata/repeatable-requests/v1.0/repeatable-requests-v1.0.pdf

Technical Committee:
OASIS Open Data Protocol (OData) TC

Chairs:
Ralf Handl (ralf.handl@sap.com), SAP SE
Mike Pizzo (mikep@microsoft.com), Microsoft

Editors:
Evan Ireland (evan.ireland@sap.com), SAP SE
Matt Borges (matt.borges@sap.com), SAP SE

Related work:
This specification is related to:

- *OData Version 4.01. Part 1: Protocol*. Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OASIS Standard. Latest stage: https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html.
- *OData JSON Format Version 4.01*. Edited by Michael Pizzo, Ralf Handl, and Mark Biamonte. OASIS Standard. Latest stage: https://docs.oasis-open.org/odata/odata-json-format/v4.01/odata-json-format-v4.01.html.
- *OData Vocabularies Version 4.0*. https://oasis-tcs.github.io/odata-vocabularies/vocabularies/Org.OData.Repeatability.V1.xml.

Abstract:
This document describes a method to provide the ability to retry unsafe (i.e. POST, PUT, PATCH, DELETE) requests without incurring unintended side-effects. This specification can be applied to any HTTP based protocol.

Status:
This document was last revised or approved by the OASIS Open Data Protocol (OData) TC on the above date. The level of approval is also listed above. Check the “Latest stage” location noted above for

possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata#technical.

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

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

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

[RepeatableRequests-v1.0]

Repeatable Requests Version 1.0. Edited by Evan Ireland and Matt Borges. 07 July 2020. OASIS Committee Specification 01. https://docs.oasis-open.org/odata/repeatable-requests/v1.0/cs01/repeatable-requests-v1.0-cs01.html. Latest stage: https://docs.oasis-open.org/odata/repeatable-requests/v1.0/repeatable-requests-v1.0.html.

Notices

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

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

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

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

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

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

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

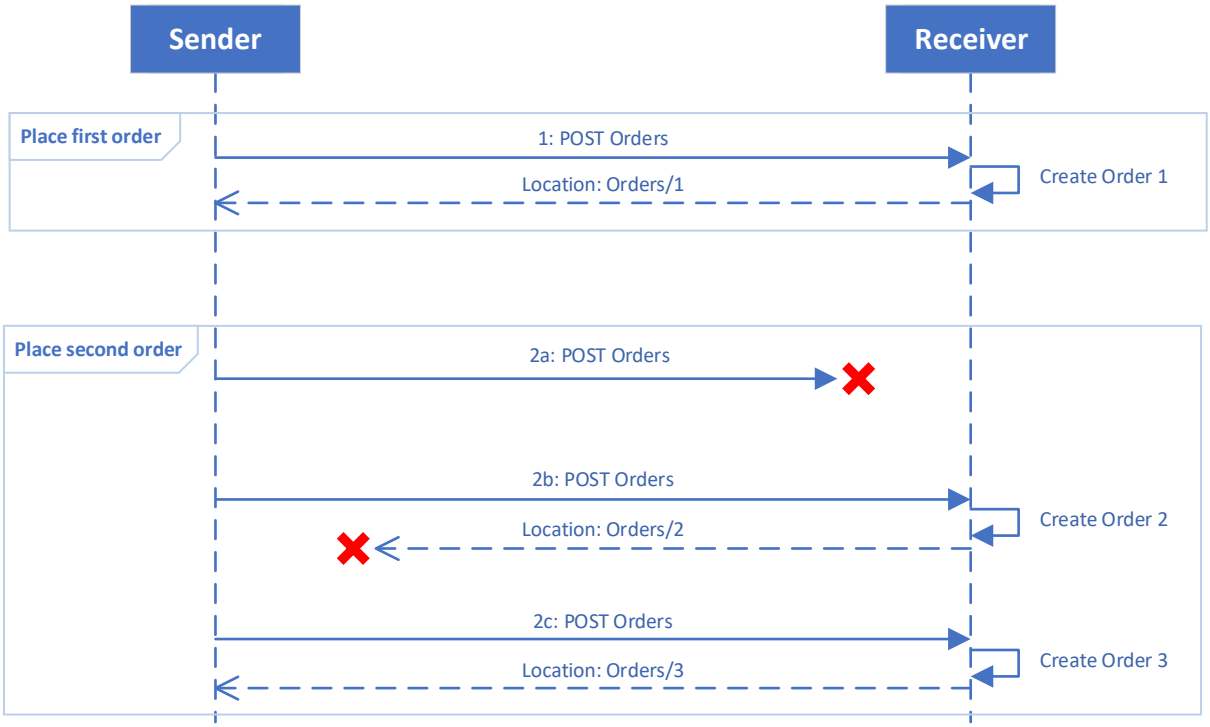
The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading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

Table of Contents

1 Introduction

HTTP is an inherently unreliable protocol. If connection or other issues prevent the client from receiving a response, the client is left in doubt as to whether the request was processed by the server. For safe HTTP requests as defined in **[RFC7231]** section 4.2 (for example, GET) the client can simply re-try the request, but for operations that change state (for example, inserting a new resource or invoking a side-effecting service operation such as `PlaceOrder` or `TransferFunds`) re-issuing the request may result in an undesired state (for example, two orders placed, or double the amount of funds transferred).

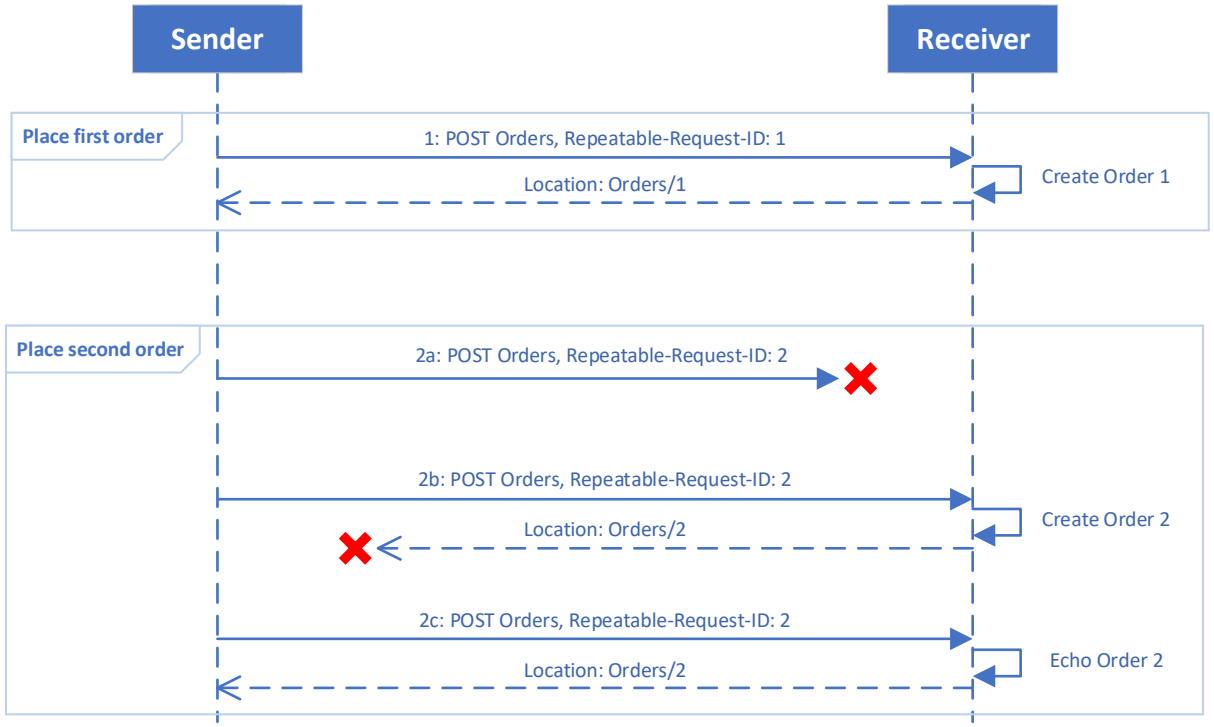
Figure 1: Lost requests and responses without Repeatability



As the sender does not receive responses to requests 2a and 2b, it creates three orders instead of the intended two orders.

This document proposes a simple approach that lets the receiver recognize repeated requests, so it can echo a stored response for an already received and processed request without processing the request a second time:

Figure 2: Lost requests and responses with Repeatability



1.1 IPR Policy

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/odata/ipr.php).

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3 Non-Normative References

[OData-Protocol] OData Version 4.01 Part 1: Protocol. See link in "Related work" section on cover page.

[OData-JSON] OData JSON Format Version 4.01. See link in "Related work" section on cover page.

[OData-VocRep] OData Vocabularies Version 4.0: Repeatability Vocabulary. See link in **"**Related work**"** section on cover page.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005. https://www.rfc-editor.org/info/rfc4122**.**

[RFC5789] Dusseault, L., and J. Snell, "Patch Method for HTTP", RFC 5789, March 2010. http://tools.ietf.org/html/rfc5789.

[RFC7231] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014. https://www.rfc-editor.org/info/rfc7231.

1.4 Typographical Conventions

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

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

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

Example 1: text describing an example uses this paragraph style

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

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

All other text is normative unless otherwise labeled.

A *repeatable request* is one in which the client sends the request with the `Repeatability-Request-ID` and `Repeatability-First-Sent` headers.

2 Repeatable Request

A *repeatable request* is one in which the client sends the request with the `Repeatability-Request-ID` and `Repeatability-First-Sent` headers.

An *unsafe request* is a non-idempotent request; that is, a request which has the potential to change the service each time it is executed. For example, a request to create a resource is an unsafe request because executing the request multiple times could result in the same resource being created multiple times.

3 Header Fields

Repeatable Requests defines semantics around the following request and response headers.

3.1 Request Headers

Two request headers are required to facilitate the ability to retry requests without incurring unintended side effects. Clients MUST send the `Repeatability-Request-ID` and `Repeatability-First-Sent` header to specify that a request is repeatable; that is, that the client can make the request multiple times with the same `Repeatability-Request-ID` and `Repeatability-First-Sent` header values and get back an appropriate response without the server executing the request multiple times.

Another optional header, `Repeatability-Client-ID`, can be provided to facilitate the ability for a server to eagerly cleanup tracking information that it may use for the implementation of repeatable requests (rather than keeping such information for a possibly extended retention period).

3.1.1 Repeatability-Request-ID

`Repeatability-Request-ID` MUST be sent by clients to specify that a request is repeatable. The value of the `Repeatability-Request-ID` is an opaque string representing a client-generated, globally unique for all time, identifier for the request. Servers MUST accommodate the 36-character hexadecimal case-insensitive encoding of a UUID (GUID), as defined in **[RFC4122]**. It is recommended for security purposes to use version 4 (random) UUIDs as defined in **[RFC4122]** section 4.1.3. Support for other forms of unique identifiers is optional.

If specified, the client directs that the request MUST be repeatable. Servers aware of repeatability but unable to fulfill this direction for this request type MUST not execute the request and instead return `501 Not Implemented`.

3.1.2 Repeatability-First-Sent

`Repeatability-First-Sent` MUST be sent by clients to specify that a request is repeatable. `Repeatability-First-Sent` is used to specify the date and time at which the request was first created. `Repeatability-First-Sent` value MUST be expressed using the IMF-fixdate form of HTTP-date as defined in **[RFC7231]**.

`Repeatability-First-Sent` allows the server to determine if the request is within its currently tracked window of time for repeatability. If `Repeatability-First-Sent` is within the server's window and the request has not been seen previously, the server can safely execute it. If it is not in the window of currently tracked requests, the server cannot guarantee the request was not already executed and so MUST return an error. Without using `Repeatability-First-Sent`, if/when the server cleans up tracking information, the server could receive a `Repeatability-Request-ID` that it has already executed but no longer has any tracking data for and so the server would incorrectly execute the request again.

3.1.3 Repeatability-Client-ID

`Repeatability-Client-ID` is an optional header that MAY be provided by the client. `Repeatability-Client-ID` is an opaque string representing a client-generated, globally unique for all time, identifier for the instance of the client application that issued the request. Servers, if they do not ignore this header, MUST accommodate the 36-character hexadecimal case-insensitive encoding of a UUID (GUID), as defined in **[RFC4122]**. It is recommended for security purposes to use version 4 (random) UUIDs as defined in **[RFC4122]** section 4.1.3. Support for other forms of unique identifiers is optional.

`Repeatability-Client-ID`, if provided by the client, MAY be used by the server to support bulk Repeatability Deletion.

3.2 Response Headers

3.2.1 Repeatability-Result

Servers aware of repeatability MUST return the `Repeatability-Result` response header in the result of a repeatable request with one of the case-insensitive values `accepted` or `rejected`.

3.2.1.1 Accepted

The server MUST return the value `accepted` for the `Repeatability-Result` response header if the request was accepted and the server guarantees that the server state reflects a single execution of the operation.

The response returned by the server MUST be the success or failure state of the operation as first executed by the server and reflects either the current state of the system or the state as it existed when the request was first received.

3.2.1.2 Rejected

The server MUST return the value `rejected` for the `Repeatability-Result` response header if the request was rejected because the combination of `Repeatability-First-Sent` and `Repeatability-Request-ID` were invalid or because the `Repeatability-First-Sent` value was outside the range of values held by the server.

The server MUST also return a `4xx` or `5xx` response code without attempting to execute the request when `rejected` is returned.

The server state MUST be the same as if the request was never received.

4 Client Behavior

In order to issue a repeatable request, the client first creates a UUID (GUID) and encodes that as a string. It MUST set that as the string value of the `Repeatability-Request-ID` header and MUST also set the `Repeatability-First-Sent` header to the current date-time value.

The client MAY also include a `Repeatability-Client-ID` header allowing the server to associate with the client any tracking information that it may use in support of repeatable requests.

If the request fails to return, for example, due to connection issues, what the client can do will depend on what it already knows about the server.

If the client has out-of-band knowledge that the server supports repeatability, then the client can safely re-execute the request.

If the client had previously executed a repeatable request for which it got a response, it can also determine whether the server supports repeatability based on whether or not the server responded with a `Repeatability-Result` header. If the server responded with a `Repeatability-Result` header the client knows that the server supports repeatability; re-executing the request in this case is safe. However, if the response did not return a `Repeatability-Result` header, then the client MUST assume that the request did not reach a server that knows about repeatable requests.

If the client does not have out-of-band knowledge that that the server supports repeatability and the client has not previously sent a repeatable request for which it received a `Repeatability-Result` header, it MUST NOT assume the server supports repeatability and therefore cannot safely re-execute the request. In this case, re-issuing the request MAY result in multiple executions.

If the client has determined that the server supports repeatability, it can re-execute a request for which it did not receive a response by sending the request again with the same `Repeatability-Request-ID` and `Repeatability-First-Sent` headers (and `Repeatability-Client-ID`, if it was specified previously).

If the request returns with a `Repeatability-Result` header with a value of `accepted` then the client knows that the request has been executed in a repeatable manner and consumes the results.

If the request returns with a `Repeatability-Result` header with a value of `rejected` then the client knows that the creation time is beyond the window of requests that the server has stored and it cannot safely retry the operation, or some other error has occurred (for example, the `Repeatability-Request-ID` and `Repeatability-First-Sent` values were inconsistent with each other or with a previous request).

If the request returns with HTTP response code `501 Not Implemented` with a `Repeatability-Result` value of `accepted`, it implies the service knows about repeatability but there is something wrong with the request.

If the request returns with HTTP response code `501 Not Implemented` with a `Repeatability-Result` value of `rejected`, it implies that the service does know about repeatability but the server does not support repeatability of the request.

5 Server Behavior

This section applies to servers that are aware of this specification.

If the server receives the `Repeatability-Request-ID` header but the `Repeatability-First-Sent` value is missing, it MUST return `400 Bad Request` with a `Repeatability-Result` of `rejected`.

If the server receives the `Repeatability-First-Sent` header but the `Repeatability-Request-ID` value is missing, it MUST return `400 Bad Request` with a `Repeatability-Result` of `rejected`.

If the server receives the `Repeatability-First-Sent` header but the value is not a valid IMF-fixdate form of HTTP-date as defined in **[RFC7231]**, it MUST return `400 Bad Request` with a `Repeatability-Result` of `rejected`.

If the server receives a request with valid, non-null, `Repeatability-Request-ID` and `Repeatability-First-Sent` values and

- does not support repeatable execution of the request it MUST return `501 Not Implemented` with a `Repeatability-Result` of `rejected`, or
- the `Repeatability-First-Sent` value is before the earliest remembered `Repeatability-Request-ID`, or this request cannot be reliably executed for some other implementation-specific reason, the server MUST return `412 Precondition Failed` with a `Repeatability-Result` value of `rejected`, or
- has not seen the `Repeatability-Request-ID` since its earliest remembered `Repeatability-Request-ID` (if any), and the `Repeatability-First-Sent` value is within its window of remembered `Repeatability-Request-ID` values, then it MUST execute the request and return the result with `Repeatability-Result` header value of `accepted` and record the `Repeatability-Request-ID` and any additional information to ensure the request is not executed more than once, and any additional information to respond should the client send the request again.

The server SHOULD return an error `400 Bad Request` along with a `Repeatability-Result` value of `rejected` if `Repeatability-Request-ID` is non-null and the request verb, URI, or header fields other than `Date` are different from that of the original request.

If the server has seen the `Repeatability-Request-ID`, it MAY return an error `400 Bad Request` along with a `Repeatability-Result` header value of `rejected` if the request body was different from that of the original `Repeatability-Request-ID`.

If the server has seen the `Repeatability-Request-ID` and the request matches the previous request to the extent validated by the server, the server MUST return a response with a `Repeatability-Result` value of `accepted` that is either:

- the same response code and body as was generated (if any) when the original request with that `Repeatability-Request-ID` was processed, or
- the response code and response body resulting from re-executing the request if the original response code was `4xx` or `5xx`, i.e. a client error or an internal server error.

In order to permit the server to optimize the storage of response bodies, the client and server may wish to negotiate the amount of content that will be returned in an initial response and any subsequent repeated response. The mechanism for such response content negotiation may depend on the protocol used.

Whether a server is considered to have *seen* a previous request should be transactionally consistent with the mutating effects of the request. For example, a server is not required to remember a previous request whose effects were rolled back due to a failure, since the client could reissue such a request without any possibility for duplication of the effects.

Servers MAY support repeatability on `POST`, `PUT`, `PATCH` and `DELETE`:

- Repeatability on `POST` ensures that the operation is executed, or the insert is performed no more than once.
- Repeatability on `PUT` or `PATCH` is different from the use of an ETag in that repeated `PUT` or `PATCH` operations to the same resource will return success (or fail), possibly including a payload, versus a concurrency violation. It is important for a repeated request to return with success if the original request actually succeeded, rather than a failure due to a conflict detected on the repeated execution.
- Repeatability on `DELETE` is different from use of an ETag in that repeated `DELETE` operations to the same resource will return success (or fail) rather than `404 Not Found`.

Servers MUST ignore `Repeatability-Request-ID` and `Repeatability-First-Sent` for `GET` and `HEAD` requests.

If the server knows about repeatability and it does not support it for a particular repeatable request, it MUST NOT execute the request and MUST return a `4xx` or `5xx` response with a `Repeatability-Result` header with a value of `rejected`.

6 Example Scenarios

Example 2: Create a new Order. Adding a new item to a collection is a `POST` request to the collection. To safeguard against a lost response the client adds repeatability headers:

```
POST /service/Orders
Content-Type: application/json
Repeatability-Request-ID: 112a3a3e-f94c-4f56-b49b-5aab3d97e5b7
Repeatability-First-Sent: Tue, 26 Mar 2019 16:06:51 GMT

{
  "CustomerID": "ALFKI",
  "OrderLines": [
    {
      "ProductID": "tomatoes-red-cherry",
      "Quantity": 5,
      "Unit": "kg",
    },
    {
      "ProductID": "grapejuice-merlot",
      "Quantity": 2,
      "Unit": "l",
    }
  ]
}
```

The client does not receive a response, so it simply sends the request again:

```
POST /service/Orders
Content-Type: application/json
Repeatability-Request-ID: 112a3a3e-f94c-4f56-b49b-5aab3d97e5b7
Repeatability-First-Sent: Tue, 26 Mar 2019 16:06:51 GMT

{
  "CustomerID": "ALFKI",
  "OrderLines": [
    {
      "ProductID": "tomatoes-red-cherry",
      "Quantity": 5,
      "Unit": "kg",
    },
    {
      "ProductID": "grapejuice-merlot",
      "Quantity": 2,
      "Unit": "l",
    }
  ]
}
```

This time the client receives a response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://host/service/Orders/4711
Repeatability-Result: accepted

{
  "OrderID": 4711,
  "CustomerID": "ALFKI",
  "OrderLines": [
    {
      "ProductID": "tomatoes-red-cherry",
      "Quantity": 5,
```

```
      "Unit": "kg",
    },
    {
      "ProductID": "grapejuice-merlot",
      "Quantity": 2,
      "Unit": "l",
    }
  ]
}
```

The `Repeatability-Result` response header tells the client that it need not worry: the new order was created exactly once.

Example 2: Invoke an Action. The client wants to place an exact clone of a recent order using the Clone action:

```
POST /service/Orders/4711/Clone
Content-Type: application/json
Repeatability-Request-ID: a47a83d9-be50-46aa-ab2a-55f18f4fbc64
Repeatability-First-Sent: Mon, 01 Apr 2019 06:22:03 GMT

{}
```

The client does not receive a response, so it simply sends the request again. This time the client receives a response:

```
HTTP/1.1 204 No Content
Location: http://host/service/Orders/4712
Repeatability-Result: accepted
```

The `Repeatability-Result` response header tells the client that it need not worry: the new order was cloned exactly once.

7 Repeatable Request Cleanup

In some situations, such as when using occasionally-connected mobile devices, clients may expect the server to offer a significant retention period (e.g. 50 days) for remembered repeatable requests. In such situations, the server's storage system may be burdened by the retention requirements, so it is valuable to offer clients a way to signal that certain remembered repeatable requests may be forgotten (deleted) by the server even before the retention period has expired. Some clients may be able to acknowledge that they have received all responses to all outstanding requests. Bulk deletion of all the tracking information for repeatable requests from a particular `Repeatability-Client-ID` may enable a significant performance boost for the server.

A server MAY support the deletion of remembered requests by `Repeatability-Request-ID`. If supported, the server SHOULD use the `DELETE` method with the URL pattern "`$RepeatableRequestWithRequestID/<Repeatability-Request-ID>`" for this purpose. The HTTP response status SHOULD be `204 No Content`, even if no such request was found.

A server MAY support the deletion of remembered requests by `Repeatability-Client-ID`. If supported, the server SHOULD use the `DELETE` method with the URL pattern "`$RepeatableRequestsWithClientID/<Repeatability-Client-ID>`" for this purpose. The HTTP response status SHOULD be `204 No Content`, even if no such requests were found.

Note that supporting cleanup by `Repeatability-Client-ID` does not mean that the server needs to record information about client instances separately from its set of remembered repeatable requests. For example, it might be achieved simply with an extra (indexed) storage column in the storage table used to track repeatable requests.

8 Incorporation into OData

This specification is valid outside the context of OData but this section details the specifics of using it with OData, see **[OData-Protocol]**.

8.1 Support

OData services are not required to support Repeatability. Clients MUST rely on external means (e.g. capabilities) in order to know whether the server supports repeatability.

8.2 Discovery

Services supporting repeatability SHOULD annotate the entity container, entity sets, singletons, action imports, or actions in the service metadata with the term `Repeatability.Supported` defined in the `Repeatability` vocabulary, see **[OData-VocRep]**.

Services supporting repeatable requests cleanup by `Repeatability-Request-ID` and/or `Repeatability-Client-ID` SHOULD annotate the entity container with the terms `Repeatability.DeleteWithRequestIDSupported` and/or `Repeatability.DeleteWithClientIDSupported`.

If lower-level elements such as individual entity sets do not support repeatability, then they can opt out of repeatability using a lower-level override of the `Repeatability.Supported` term.

Services MAY support repeatability without the use of annotations in the service metadata.

8.3 Response Payload

The client can optionally use system query options `$select` and/or `$expand` in the request URL to force the service to return a payload containing the minimal information required by the client, as compared to what it would ordinarily return without the use of system query options. Note that `@Core.ContentID` is always returned in the response payload if it was specified in the request body.

If the client sends a repeatable request containing a data modification operation for an entity, and the client does not include `$select` or `$expand` in the request URL, the server MAY choose to return `204 No Content` even if it would ordinarily return status code `200` or `201` for a non-repeatable request.

The above paragraphs allow the service to minimize the tracking information that it stores in support of repeatable requests.

8.4 Batch Requests

Services MAY support repeatability for individual requests within a batch request, as well as for individual requests within a change set or atomicity group within a batch request.

Individual requests within a batch MAY have a mix of `Repeatability-Request-ID` and `Repeatability-First-Sent` values. In this case, each individual response within the batch response has the appropriate `Repeatability-Result` (or not) according to the corresponding request.

Repeatable request headers MUST NOT be applied to change sets or atomicity groups directly because there is no way to specify headers for an atomicity group in JSON batch requests, see **[OData-JSON]**. To make a change set or atomicity group repeatable, a client MUST specify the same `Repeatability-Request-ID` and `Repeatability-First-Sent` values for all requests in the change set or atomicity group. The client MUST retry the entire change set or atomicity group as a unit if it is repeatable; individual operations within the change set or atomicity group MUST NOT be retried.

There is no correlation between the repeatability of a request and the repeatability of any of its dependent requests. That is, a repeatable request may be retried without retrying any of its dependent requests.

Repeatability cannot be applied to batch requests themselves because a single `Repeatability-Request-ID` on the batch request is not sufficient for uniquely identifying the individual requests within

the batch request, and because repeatability implies transactional atomicity which cannot be guaranteed for a batch request containing multiple change sets, some of which may succeed (commit) and some of which may fail (rollback). Therefore, if a server receives a batch request with either a `Repeatability-Request-ID` or a `Repeatability-First-Sent` value, it MUST NOT execute any requests within the batch and MUST respond with `4xx` or `5xx` response and with a `Repeatability-Result` value of `rejected`.

9 Security Considerations

For HTTP relevant security implications please cf. the relevant sections of **[RFC7231]** (9. Security Considerations) and for the HTTP `PATCH` method **[RFC5789]** (5. Security Considerations) as starting points.

Servers SHOULD authenticate the client before further evaluating the repeatability of any requests.

As mentioned in section 3.1.1, it is recommended for security purposes that clients use randomly generated `Repeatability-Request-ID` values such as version 4 UUIDs as defined in **[RFC4122]** section 4.1.3.

10 Conformance

10.1 Service Conformance

In order to conform to this specification, a service:

1. MUST conform to the semantics of the following headers, or fail the request:
   1.1. `Repeatability-Request-ID` (section 3.1.1)
   1.2. `Repeatability-First-Sent` (section 3.1.2)
2. MUST return the `Repeatability-Result` response header in response to repeatable requests (section 2) with one of the values `accepted` (section 3.2.1.1) or `rejected` (section 3.2.1.2)
3. MUST follow the guidelines in Server Behavior (section 5)

10.2 Client Conformance

If a client wants to safely repeat a request, a client:

1. MUST specify the `Repeatability-Request-ID` header in an unsafe request (section 3.1.1)
2. MUST specify the `Repeatability-First-Sent` header in a unsafe request (section 3.1.2)
3. MUST only repeat a request if it has previously determined (through whatever means) that the server supports repeatability and MUST specify the repeatability headers (section 4).

Appendix A. Acknowledgments

The following individuals were members of the OASIS OData Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

- Johannes Alberti (SAP SE)
- Jay Balunas (Red Hat)
- Mark Biamonte (Progress Software)
- Michael Bolz (SAP SE)
- Matthew Borges (SAP SE)
- Pablo Castro (Microsoft)
- Renze de Vries (SDL)
- Yi Ding (Microsoft)
- Rob Dolin (Microsoft)
- Chet Ensign (OASIS)
- George Ericson (Dell)
- Juliet Espinosa (GS1 Colombia/LOGYCA)
- Colleen Evans (Microsoft)
- Ashwani Goyal (SAP SE)
- chong gu (Huawei Technologies Co., Ltd.)
- Stefan Hagen (Individual)
- Ralf Handl (SAP SE)
- Hongxia Hao (Huawei Technologies Co., Ltd.)
- Hubert Heijkers (IBM)
- Oliver Heinrich (SAP SE)
- Serge Huber (Jahia Solutions Group SA)
- Jens Huesken (SAP SE)
- Evan Ireland (SAP SE)
- Gershon Janssen (Individual)
- Ling Jin (IBM)
- Gareth Jones (Microsoft)
- Ted Jones (Red Hat)
- Stephan Klevenz (SAP SE)
- Konstantin Kosinsky (Microsoft)
- Gerald Krause (SAP SE)
- Steven Legg (ViewDS Identity Solutions)
- shitao li (Huawei Technologies Co., Ltd.)
- Nuno Linhares (SDL)
- Saurabh Madan (Microsoft)
- Susan Malaika (IBM)
- Web Master (OASIS)
- Peggy Moloney (Microsoft)
- Joel Myhre (Pacific Disaster Center)
- Bob Natale (Mitre Corporation)
- Kenneth Peeples (Red Hat)
- Michael Pizzo (Microsoft)
- Jovan Popovic (Microsoft)
- Zheng Qian (Huawei Technologies Co., Ltd.)
- Ramesh Reddy (Red Hat)
- Uday Singh (SAP SE)
- Dan Solero (AT&T)
- Christof Sprenger (Microsoft)
- Divyansh Srivastava (Microsoft)
- Congyong Su (Microsoft)
- Wojciech Trocki (Red Hat)
- Mathias Uhlmann (SAP SE)
- Steve Veum (Progress Software)
- Aaron Wang (Microsoft)
- qingwei zhao (Huawei Technologies Co., Ltd.)
- Martin Zurmuehl (SAP SE)

Appendix B. Revision History

Revision	Date	Editor	Changes Made
Initial Draft 01	2013-06-25	Mike Pizzo, Ralf Handl	Initial version
Committee Note Draft 01	2019-10-17	Matt Borges, Evan Ireland	Aligned header names Added Repeatability Deletion Clarified client and server behavior with regard to errors Clarified what servers are required to store and return for repeated requests and how the client and server can negotiate this for OData
Committee Specification Draft 01	2020-04-07	Matt Borges, Evan Ireland	Changed the committee note draft into a committee specification draft