OData Extension for Temporal Data Version 4.0

Committee Specification Draft 04

25 January 2022

This stage:

https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/odata-temporal-ext-v4.0-csd04.md (Authoritative)
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/odata-temporal-ext-v4.0-csd04.html
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/odata-temporal-ext-v4.0-csd04.pdf

Previous stage:

https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/cs01/odata-temporal-ext-v4.0-cs01.docx (Authoritative)
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/cs01/odata-temporal-ext-v4.0-cs01.html
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/cs01/odata-temporal-ext-v4.0-cs01.pdf

Latest stage:

https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/odata-temporal-ext-v4.0.md (Authoritative)
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/odata-temporal-ext-v4.0.html
https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/odata-temporal-ext-v4.0.pdf

Technical Committee:

OASIS Open Data Protocol (OData) TC

Chairs:

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

Editors:

Ralf Handl (ralf.handl@sap.com), SAP SE
Hubert Heijkers (hubert.heijkers@nl.ibm.com), IBM
Gerald Krause (gerald.krause@sap.com), SAP SE
Michael Pizzo (mikep@microsoft.com), Microsoft
Heiko Theißen (heiko.theissen@sap.com), SAP SE
Martin Zurmuehl (martin.zurmuehl@sap.com), SAP SE

Additional artifacts:

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

- ABNF components: *OData Temporal ABNF Construction Rules Version 4.0 and OData Temporal ABNF Test Cases*: https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/abnf/
- OData Temporal Vocabulary:

- https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/vocabularies/Org.OData.Temporal.V1.json
- https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/vocabularies/Org.OData.Temporal.V1.xml

Related work:

This specification is related to:

- *OData Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. A multi-part Work Product that includes:
  - *OData Version 4.02 Part 1: Protocol*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html
  - *OData Version 4.02 Part 2: URL Conventions*. Latest stage. https://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html
  - *ABNF components: OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases*. https://docs.oasis-open.org/odata/odata/v4.0/csd04/abnf/
- *OData Vocabularies Version 4.0*. Edited by Michael Pizzo, Ralf Handl, and Ram Jeyaraman. Latest stage: https://docs.oasis-open.org/odata/odata-vocabularies/v4.0/odata-vocabularies-v4.0.html
- *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-json/v4.0/odata-csdl-json-v4.0.html
- *OData Common Schema Definition Language (CSDL) XML Representation Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-csdl-xml/v4.0/odata-csdl-xml-v4.0.html
- *OData JSON Format Version 4.02*. Edited by Michael Pizzo, Ralf Handl, and Heiko Theißen. Latest stage: https://docs.oasis-open.org/odata/odata-json-format/v4.02/odata-json-format-v4.02.html
- *OData Data Aggregation Extension Version 4.0*. Edited by Ralf Handl, Hubert Heijkers, Gerald Krause, Michael Pizzo, Heiko Theißen, and Martin Zurmuehl. Latest stage: https://docs.oasis-open.org/odata/odata-data-aggregation-ext/v4.0/odata-data-aggregation-ext-v4.0.html

Abstract:

This specification defines how to represent and interact with time-dependent data using the Open Data Protocol (OData).

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.

Key words:

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14 **[RFC2119]** and **[RFC8174]** when, and only when, they appear in all capitals, as shown here.

Citation format:

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

[OData-Temporal-v4.0]

OData Extension for Temporal Data Version 4.0. Edited by Ralf Handl, Hubert Heijkers, Gerald Krause, Michael Pizzo, Heiko Theißen, and Martin Zurmuehl. 25 January 2022. OASIS Committee Specification Draft 04. https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/csd04/odata-temporal-ext-v4.0-csd04.html. Latest stage: https://docs.oasis-open.org/odata/odata-temporal-ext/v4.0/odata-temporal-ext-v4.0.html.

Notices

Table of Contents

1 Introduction

This specification adds the notion of time-dependency to the Open Data Protocol (OData) without changing any of the base principles of OData. It defines semantics and a representation for temporal data, especially:

- Semantics and operations for querying and modifying temporal data,
- Vocabulary terms to annotate which data depends on time, and how.

1.1 Changes from Earlier Versions

1.2 Glossary

1.2.1 Definitions of Terms

1.2.1.1 Application Time

Application time is used to describe data that is known to change over time, for example the budget of a department, or which department an employee works for. Application time may capture planned changes in the future, for example transferring an employee to a new department next month or capturing next year's budget for a department. Both future and past data can be changed.

1.2.1.2 System Time

System time is used to record when data became known by the "system of record". System time does not extend into the future, and record entries are only added and are never changed.

System time is never manipulated directly, and typically not visible in APIs, except for "last changed at" time stamps, and the corresponding properties are read-only.

As system time is typically not visible in APIs, we do not consider this in the remainder of this document.

1.2.1.3 Temporal Object

A temporal object is a set of data whose change over time is tracked by the service as a sequence of time slices.

1.2.1.4 Time Slice

A piece of temporal data with attached time period, documenting that the data did not change during this time period.

Time slices for the same temporal object are non-overlapping, so for any given point in time there is at most one slice whose time period contains that point in time.

Time slices for a temporal object need not cover the complete timeline. There can be points in time for which no time slice exists, indicating that the object's values are not known to the service.

1.2.1.4.1 Closed-Open Semantics

Time slices typically use closed-open semantics, following **[SQL:2011]**. This means the start is part of the period, the end is not part of the period, and for directly adjacent time slices the end of the earlier time slice is identical to the start of the next time slice. The period start must be less than the period end.

1.2.1.4.2 Closed-Closed Semantics

Some software systems predating the availability of temporal databases and with data type *date* for the application-time period start and end use closed-closed semantics. Temporal services on top of these systems can either convert their period end boundaries on-the-fly by adding one day on the way out and subtracting one day on the way in, or alternatively express the used time slice semantics via annotations.

1.2.1.5 Snapshot Entity Set

An entity in a snapshot entity set represents a temporal object at a specified point in time. When the entity is addressed via a resource path (directly or via related resources), the point in time must be specified, see section "Propagation of Temporal Query Options" for details on how to determine this point in time.

The entity's id and its canonical URL are independent of this point in time. Hence, they are sufficient to *reference* the entity but not *address* it. In other words: the entity can be considered a function that maps each point in time to an instance of the entity type. That function can be *referenced* but only its values can be *addressed*.

Without a point in time, statements about individual structural or navigation properties of the entity or even about its existence cannot be made, and change tracking is not possible.

Snapshot entity sets MUST be annotated with a `Timeline` of type `TimelineSnapshot`.

1.2.1.6 Timeline Entity Set

An entity in a timeline entity set represents one time slice of a temporal object, and all time slices of that temporal object are part of the entity set.

The entity's id and its canonical URL depend on the period of application time of the corresponding time slice. *Referencing* and *addressing* the entity are equivalent concepts, unlike in the snapshot case.

The response to a time-range request need not reflect the time-slice cut of the underlying data model.

Services MAY condense/defragment adjacent time slices that do not differ in represented properties other than the period boundaries. This reduces the payload size, and it also avoids potential information leakage if the service only publishes a projection of the underlying data model.

Services MAY also split time slices to align their boundaries with nested expanded time slices to represent a "coarsest common slicing" across an expand tree.

Timeline entity sets MUST be annotated with a `Timeline` of type `TimelineVisible`.

1.2.2 Document Conventions

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

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

Example 1: text describing an example uses this paragraph style

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

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

All other text is normative unless otherwise labeled.

---

2 Overview

When keeping track of time, the most important questions are:

- When did or will something happen?
- When did we learn that it happened?

This leads to two “directions” or “dimensions” of time (measured as a date or date-plus-time value):

- Application time, also called actual time, business time, effective time, or valid time, and
- System time, also called recording time, audit time, or transaction time.

Keeping track of time is typically done by storing data together with the time period for which that data is deemed valid or effective, using separate periods for application time and system time, and the time periods are part of the logical key for “records”. See **[SQL:2011]** or **[Kulkarni]** on how this is done in the SQL standard.

A consumer’s perspective on this data can be different: even if time is tracked internally, the period of time may or may not be visible in a consumer’s perspective, and even if visible the related properties are often not considered part of an entity’s identity. For example, an employee is still the same person even after switching to another department.

The goals of this extension are:

- Keep the API models as simple as possible by allowing to hide time-dependency,
- Provide easy means for point-in-time queries even if time-dependency is hidden,
- Provide easy means for time-range queries if time-dependency is visible, and
- Provide easy means for modifying time-dependent data.

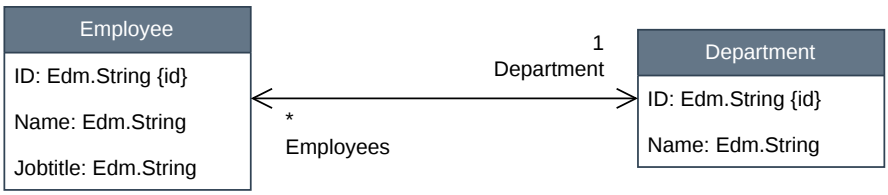
2.1 Example Model

Assume a simple scenario: employees will work in different roles and in different departments during their career, and sometimes they even change their name. At each point in time the employee has exactly one name, one job title, and is assigned to one department.

Employees are assigned an immutable ID when first entering the company, and all employee-related data will be recorded with an application-time period attached. This allows entering planned changes ahead of time, using a future application-time start date: if an employee will get a different job title or switch to a different department, this data can already be entered into the system and stored as a new record.

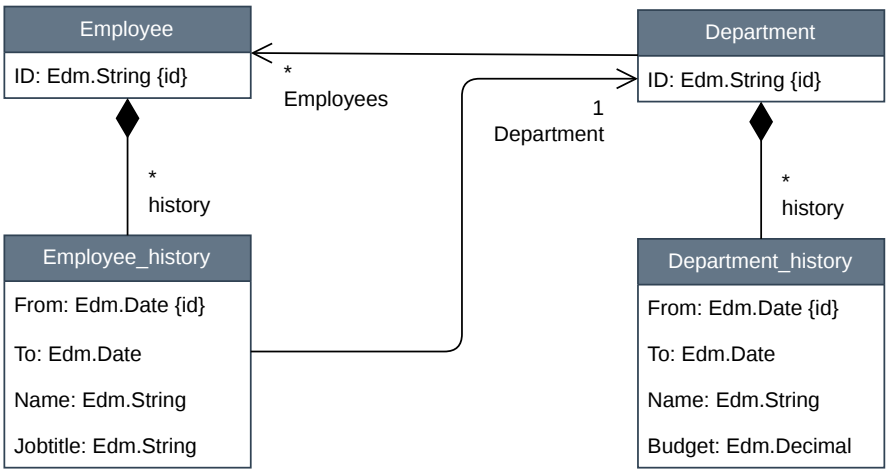
In the sections below we will discuss two different API models that can be backed by the same storage model:

Example 2: model for `api-1` with snapshot entity sets (hidden application time), key properties marked with {id}



and

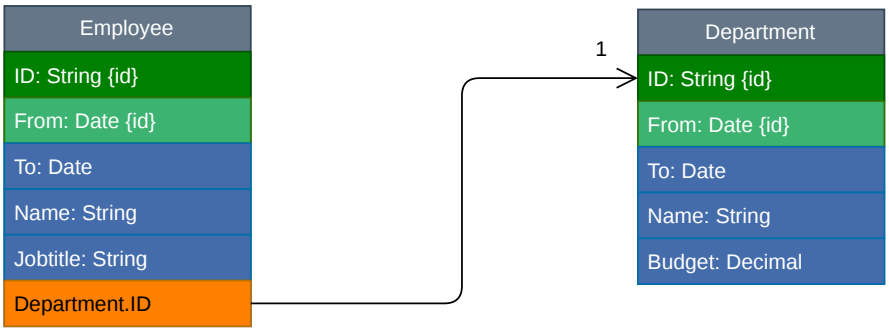
Example 3: model for `api-2` *with timeline entity sets (visible application time), key properties marked with {id}*



2.2 Example Data

Both API models in the previous section are views on the same underlying data. A possible storage model for this data is:

Example 4: simple storage model: object key in dark green, temporal sub-key in light green, foreign keys in orange, non-key fields in blue



The period start date is used as the temporal sub-key for identifying time slices together with the key of the temporal object.

Note: alternatively, the period end date could have been used as temporal sub-key, or the primary key of the time slice tables could have been an artificial key (sequence number, UUID, …) with both the temporal object key and the period boundaries as non-key fields.

The following example data will be used to further illustrate the capabilities introduced by this extension. It assumes that the example services only support four-digit years.

Example 5: example data: object key in dark green, temporal sub-key in light green, foreign keys in orange, non-key fields in blue

Departments

ID	From	To	Name	Budget
D08	2010-01-01	2012-01-01	Support	1000
D08	2012-01-01	2012-06-01	Support	1250
D08	2012-06-01	2014-01-01	1st Level Support	1250
D08	2014-01-01	max	1st Level Support	1400
D15	2010-01-01	2011-01-01	Services	1100
D15	2011-01-01	max	Services	1170

Employees

ID	From	To	Name	Jobtitle	Department.ID
E314	2011-01-01	2013-10-01	McDevitt	Junior	D08
E314	2013-10-01	2014-01-01	McDevitt	Senior	D08
E314	2014-01-01	max	McDevitt	Senior	D15
E401	2009-11-01	2012-03-01	Norman	Expert	D15
E401	2012-03-01	max	Gibson	Expert	D15

2.3 Example Use Cases

A client might wish to query the example APIs in several ways:

- Snapshot API
  1. Retrieve a current employee, together with the employee's current department.
  2. Retrieve an employee as of a particular point in time, including the employee's department as of that same point in time.
- Timeline API
  1. Retrieve employees that worked for the "Services" department in 2010.
  2. Retrieve all time slices of an employee.
  3. Retrieve all time slices of an employee for a given application-time period, including all time slices of the related departments within the application-time period of each employee time slice.
  4. Change a department's budget in the past, present, or future.
  5. Change an employee's association to a department in the past, present, or future.

---

3 Vocabulary for Temporal Data

The vocabulary for temporal data **[OData-VocTemporal]** defines a structured term for describing temporal query and modification capabilities of temporal services.

The term `ApplicationTimeSupport` describes the temporal capabilities of an entity set. It can be applied to an entity set, or to a containment navigation property to describe its implicit entity sets. It has a structured type with the following properties:

- `UnitOfTime` describes the data type of the application-time period start and end values. Its value is a record of either type `UnitOfTimeDate` or `UnitOfTimeDateTimeOffset`.
  - Records of type `UnitOfTimeDateTimeOffset` MUST specify the property `Precision`.
  - Records of type `UnitOfTimeDate` MAY specify the property `ClosedClosedPeriods`.
- `Timeline` describes the kind of temporal entity set. Its value is a record of either type `TimelineSnapshot` or `TimelineVisible`.
  - Records of type `TimelineSnapshot` MUST NOT specify any properties.
  - Records of type `TimelineVisible` MUST specify the properties `PeriodStart` and `PeriodEnd`.
    - These properties MUST point to properties of the annotated entity set whose values are the period start and end of a time slice.
    - The start is always included in the time slice, the end is included if and only if the sibling property `UnitOfTime` is of type `UnitOfTimeDate` and specifies property `ClosedClosedPeriods` as `true`.
    - If the period end property does not specify a default value, a default value of "ad infinitum" is assumed.
  - Records of type `TimelineVisible` MAY specify the property `ObjectKey`.
    - `ObjectKey` is the "sub-key" or "alternate key" that identifies time slices for a single temporal object. It is only necessary if the annotated entity set can contain time slices for more than one temporal object. The object key is a collection of property paths whose value combination uniquely identifies a temporal object.
- `SupportedActions` is a collection of qualified names of temporal actions that may be bound to the annotated entity set. These can be the standard actions defined in this specification, or service-specific actions.

Example 6: `Employees` entity set from example model `api-1` annotated with temporal terms

```
"Employees": {
  "$Collection": true,
  "$Type": "OrgModel.Employee",
  "@Temporal.ApplicationTimeSupport": {
    "Timeline": {
      "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.json#Temporal.TimelineSnapshot"
    },
    "UnitOfTime": {
```

```
      "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.json#Temporal.UnitOfTimeDate"
    }
  }
},
```

Example 7: `history` *navigation property in entity set* `Employees` *from example model* `api-2` *annotated with temporal terms*

```
"$Annotations": {
  "OrgModel.Default/Employees/history": {
    "@Temporal.ApplicationTimeSupport": {
      "Timeline": {
        "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.json#Temporal.TimelineVisible",
        "TimeSliceStart": "From",
        "TimeSliceEnd": "To",
        "SupportedActions": [
          "Temporal.Update",
          "Temporal.Upsert",
          "Temporal.Delete"
        ]
      },
      "UnitOfTime": {
        "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.json#Temporal.UnitOfTimeDate"
      }
    }
  }
}
```

Example 8: `CostCenters` *entity set containing time slices for multiple temporal objects, the temporal objects identified by combination of* `AreaID` *and* `CostCenterID`

```
"CostCenter": {
  "$Kind": "EntityType",
  "$Key": ["tsid"],
  "tsid": {},
  "AreaID": {},
  "CostCenterID": {},
  "ValidTo": { "$Type": "Edm.Date" },
  "ValidFrom": { "$Type": "Edm.Date" },
  "ProfitCenterID": { "$Nullable": true },
  "DepartmentID": { "$Nullable": true }
},

"Default": {
  "$Kind": "EntityContainer",
  "CostCenters": { "$Collection": true, "$Type": "this.CostCenter" }
},

"$Annotations": {
  "this.Default/CostCenters": {
    "@Temporal.ApplicationTimeSupport": {
      "UnitOfTime": {
        "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.xml#Temporal.UnitOfTimeDate",
        "ClosedClosedPeriods": true
      },
      "Timeline": {
        "@type": "https://oasis-tcs.github.io/odata-
        vocabularies/vocabularies/Org.OData.Temporal.V1.xml#Temporal.TimelineVisible",
        "PeriodStart": "ValidFrom",
        "PeriodEnd": "ValidTo",
        "ObjectKey": ["AreaID", "CostCenterID"]
```

```
      },
      "SupportedActions": [
        "Temporal.Update",
        "Temporal.Upsert",
        "Temporal.Delete"
      ]
    }
  }
}
```

4 Temporal Requests

4.1 Temporal Expressions

A temporal expression is

- A date in the form `dateValue`, see **[OData-ABNF]**
- A timestamp in the form `dateTimeOffsetValueInUrl`, see **[OData-ABNF]**
- One of the literals `min` or `max`
- An expression resulting in a date or timestamp value

The literals `min` and `max` are interpreted depending on the data type of the time period start and end. They represent the minimum and maximum values of the time period that the service will allow. Their values are service specific as they can depend on the capabilities of the underlying data model.

Note that there is no literal for “today” as clients and services can be in different time zones and thus can have different notions of which date is “today”. Clients are advised to use concrete temporal values and should avoid using the URL function `now()` with period start and end of type `Edm.Date`.

Note that services may allow service-defined functions for temporal expressions, for example to deal with fiscal years in a particular company.

4.2 Querying Temporal Data

Temporal query options allow point-in-time as well as time-range queries. They take a temporal expression as their argument whose result type MUST match the type of the corresponding time period start and end.

Adding temporal query options to a request does not alter the response shape, it only affects the returned data.

Temporal query options can be used with data modification operations, in which case they do not alter the modification semantics of the request, they only affect the response. That is, they only influence the “implicit `GET`” after successful data modification.

If no temporal query options are specified,

- timeline entity sets return all time slices, and
- snapshot entity sets return the snapshot valid at the time of the request.

4.2.1 Propagation of Temporal Query Options

Temporal query options can be specified for the requested resource, and for expanded navigation properties. They are propagated along navigation paths and `$expand` until overridden on an expanded navigation property, and the overridden query option values are propagated from there to further expanded navigation properties.

If any temporal query option is specified for an expanded navigation property, all temporal query options are overridden, and only the explicitly specified temporal query options take effect and are propagated further.

Temporal query options can be specified even if the directly addressed resource does not track time. In this case the temporal query options do not have any effect other than being propagated along `$expand`.

For entities in a snapshot entity set the point in time for representing data is determined by the first applicable rule:

1. by the query option `$at` nested within `$expand`
2. by a `$at` value propagated along `$expand`
3. by `$at` in the query option part of the request URL, which applies to every segment of the resource path and paths that occur in system query options
4. by the default value “now” — the logic for determining this value is service-specific

For entities in a timeline entity set the time interval for filtering time slices is determined by the first applicable rule:

1. by nested temporal query options within `$expand`
2. by temporal query option values propagated along `$expand`
3. by temporal query options in the query option part of the request URL

4.2.2 Query Option `$at`

The `$at` query option takes a temporal expression as its argument. It retrieves the snapshot whose application time period contains the value of `$at`.

For timeline entity sets and collection-valued navigation to timeline entity sets, `$at=<point-in-time>` is shorthand for

`$from=<point-in-time>&$toInclusive=<point-in-time>`

The query option `$at` can be combined with `$filter` and `$search`. Only entities satisfying all specified criteria are returned.

Example 9: Retrieve current data of an employee

```
GET /api-1/Employees('E314')
```

results in

```
{
  "@context": "$metadata#Employees/$entity",
  "ID": "E314",
  "Name": "McDevitt",
  "Jobtitle": "Senior"
}
```

Example 10: retrieve employee at a specific point in application time

```
GET /api-1/Employees('E314')?$at=2012-01-01
```

results in

```
{
  "@context": "$metadata#Employees/$entity",
  "ID": "E314",
  "Name": "McDevitt",
  "Jobtitle": "Junior"
}
```

Example 11: retrieve multiple employees at a past point in time

```
GET /api-1/Employees?$filter=contains(Name,'i')&$at=2012-01-01
```

results in one time slice for each employee matching the filter at the specified point in time — note that E401 back then does not satisfy this condition

```
{
  "@context": "$metadata#Employees",
  "value": [
    {
      "ID": "E314",
      "Name": "McDevitt",
      "Jobtitle": "Junior"
    }
  ]
}
```

Expanding related entities in combination with `$at` is straight-forward: the response consists of the time slices of related entities that are valid at the requested point in time. The period boundaries of the nested entities reflect the actual validity of the related entities and are independent of the period boundaries of the base entity.

Example 12: retrieve employee in the past, show the past department as of a later point in time

```
GET /api-1/Employees('E314')?$at=2012-01-01&$expand=Department($at=2021-11-23)
```

results in

```
{
  "@context": "$metadata#Employees/$entity",
  "ID": "E314",
  "Name": "McDevitt",
  "Jobtitle": "Junior",
  "Department": {
    "ID": "D08",
    "Name": "1st Level Support"
  }
}
```

Example 13: retrieve department in the future with expanded employees at the same point in time

```
GET /api-1/Departments('D15')?$at=2015-01-01&$expand=Employees
```

results in

```
{
  "@context": "$metadata#Departments/$entity",
  "ID": "D15",
  "Name": "Services",
  "Employees": [
    {
      "ID": "E314",
      "Name": "McDevitt",
      "Jobtitle": "Senior"
    },
    {
      "ID": "E401",
      "Name": "Gibson",
      "Jobtitle": "Expert"
    }
  ]
}
```

4.2.3 Query Options `$from`, `$to`, and `$toInclusive`

The query options `$from`, `$to`, and `$toInclusive` only have an effect on timeline entity sets and collection-valued navigation to timeline entity sets. They each take a temporal expression as their argument.

Allowed combinations are:

- `$from` and `$to` defines a closed-open interval
- `$from` and `$toInclusive` defines a closed-closed interval
- `$from` and neither `$to` nor `$toInclusive` defines a closed-closed interval with right boundary `max`.

It is not allowed to combine `$from` and `$to`/`$toInclusive` with `$at` because `$at=<point-in-time>` is shorthand for

`$from=<point-in-time>&$toInclusive=<point-in-time>`

The result is restricted to time slices whose application-time period overlaps with the interval defined by the query option values, taking the closed-open or closed-closed semantics of the entity set's time slices into account. The benefit of the query options is that the client does not have to inspect the temporal annotations to determine the property names of the period boundaries, the period semantics, and get all comparison operators right.

For timeline entity sets with closed-open semantics `$from=<start>&$to=<end>` is shorthand for

`$filter=<time-slice-start> lt <end> and <time-slice-end> gt <start>`

and for timeline entity sets with closed-closed semantics shorthand for

`$filter=<time-slice-start> lt <end> and <time-slice-end> ge <start>`

For timeline entity sets with closed-open semantics `$from=<start>&$toInclusive=<end>` is shorthand for

`$filter=<time-slice-start> le <end> and <time-slice-end> gt <start>`

and for timeline entity sets with closed-closed semantics shorthand for

`$filter=<time-slice-start> le <end> and <time-slice-end> ge <start>`

The query options `$from` and `$to`/`$toInclusive` can be combined with `$filter` and `$search`. Only entities satisfying all specified criteria are returned.

If no `$select` is specified, each returned entity SHOULD contain the application-time period boundaries as part of the default selection.

Example 14: retrieve employee history over a period of application time

```
GET /api-2/Employees?$expand=history($select=Name,Jobtitle)
                    &$from=2012-03-01&$to=2025-01-01
```

results in one entity for each employee with time slices that overlap the specified application-time period:

```
{
  "@context": "$metadata#Employees",
  "value": [
    {
      "ID": "E314",
      "history": [
        {
          "Name": "McDevitt",
          "Jobtitle": "Junior",
          "From": "2011-01-01",
          "To": "2013-10-01"
```

```
        },
        {
          "Name": "McDevitt",
          "Jobtitle": "Senior",
          "From": "2013-10-01",
          "To": "2014-01-01"
        },
        {
          "Name": "McDevitt",
          "Jobtitle": "Senior",
          "From": "2014-01-01",
          "To": "9999-12-31"
        }
      ]
    },
    {
      "ID": "E401",
      "history": [
        {
          "Name": "Gibson",
          "Jobtitle": "Expert",
          "From": "2012-03-01",
          "To": "9999-12-31"
        }
      ]
    }
  ]
}
```

The history for the first employee contains two time slices that do not differ in the represented properties, caused by a department change, and the department is not part of the representation.

The service could have combined these two time slices into one.

Example 15: retrieve all employees that ever worked for department D15, with their full history, and the department's data at the start of each employee history time slice

```
GET /api-2/Departments('D15')/Employees?
    $expand=history(
      @emp=$this;
      $expand=Department(
        $expand=history($at=@emp/From)
      )
    )
```

has the following result with department names and budgets as of the beginning of each employee time slice:

```
{
  "@context": "$metadata#Employees",
  "value": [
    {
      "ID": "E314",
      "history": [
        {
          "Name": "McDevitt",
          "Jobtitle": "Junior",
          "From": "2011-01-01",
          "To": "2013-10-01",
          "Department": {
            "ID": "D08",
            "history": [{
              "Name": "Support",
              "Budget": 1000,
              "From": "2010-01-01",
```

```
            "To": "2012-10-01"
          }]
        }
      },
      {
        "Name": "McDevitt",
        "Jobtitle": "Senior",
        "From": "2013-10-01",
        "To": "2014-01-01",
        "Department": {
          "ID": "D08",
          "history": [{
            "Name": "1st Level Support",
            "Budget": 1250,
            "From": "2012-06-01",
            "To": "2014-01-01"
          }]
        }
      },
      {
        "Name": "McDevitt",
        "Jobtitle": "Senior",
        "From": "2014-01-01",
        "To": "9999-12-31",
        "Department": {
          "ID": "D15",
          "history": [{
            "Name": "Services",
            "Budget": 1170,
            "From": "2011-01-01",
            "To": "9999-12-31"
          }]
        }
      }
    ]
  },
  {
    "ID": "E401",
    "history": [
      {
        "Name": "Norman",
        "Jobtitle": "Expert",
        "From": "2009-11-01",
        "To": "2012-03-01",
        "Department": {
          "ID": "D15",
          "history": []
        }
      },
      {
        "Name": "Gibson",
        "Jobtitle": "Expert",
        "From": "2012-03-01",
        "To": "9999-12-31",
        "Department": {
          "ID": "D15",
          "history": [{
            "Name": "Services",
            "Budget": 1170,
            "From": "2011-01-01",
            "To": "9999-12-31"
          }]
        }
      }
```

```
      ]
    }
  ]
}
```

4.2.4 Interaction with Standard System Query Options

For snapshot entity sets the point in time for representing data is determined following the rules in section "Propagation of Temporal Query Options" and evaluated *first*, then all other system query options are evaluated on the data valid at that point in time, including the query option `$apply` defined in **[OData-Aggregation]**.

For timeline entity sets the interval for filtering data is determined following the rules in section "Propagation of Temporal Query Options" and evaluated as an additional criterion for `$filter` in the evaluation sequence defined in **[OData-Protocol, section "System Query Options"]**, which is evaluated *after* the query option `$apply`.

Example 16: retrieve employee history over a period of application time and filter on job title

```
GET /api-2/Employees?$expand=history(
                       $select=Name,Jobtitle;
                       $from=2012-03-01&$to=2025-01-01;
                       $filter=contains(Jobtitle,'e')
                     )
```

results in one entity for each employee with time slices that overlap the specified application-time period and satisfy the filter condition (one less than in example 14):

```
{
  "@context": "$metadata#Employees",
  "value": [
    {
      "ID": "E314",
      "history": [
        {
          "Name": "McDevitt",
          "Jobtitle": "Senior",
          "From": "2013-10-01",
          "To": "2014-01-01"
        },
        {
          "Name": "McDevitt",
          "Jobtitle": "Senior",
          "From": "2014-01-01",
          "To": "9999-12-31"
        }
      ]
    },
    {
      "ID": "E401",
      "history": [
        {
          "Name": "Gibson",
          "Jobtitle": "Expert",
          "From": "2012-03-01",
          "To": "9999-12-31"
        }
      ]
    }
  ]
}
```

The lambda operators `any()` and `all()` are not influenced by temporal query options, they are interpreted for each time slice on the filtered collection, meaning "any related time slice satisfying the lambda expression" and "all related

time slices satisfy the lambda expression". The lambda expressions however can contain sub-expressions working on the period boundaries.

Example 17: filter employees on their name at any point in time

```
GET /api-2/Employees?$expand=history($select=Name,Jobtitle)
                    &$from=2015-01-01
                    &$filter=history/any(h:startswith(h/Name,'N'))
```

results in one employee whose name matches in the past, and the matching time slice is not in the requested time period

```
{
  "@context": "$metadata#Employees",
  "value": [
    {
      "ID": "E401",
      "history": [
        {
          "Name": "Gibson",
          "Jobtitle": "Expert",
          "From": "2012-03-01",
          "To": "9999-12-31"
        }
      ]
    }
  ]
}
```

4.2.5 Requesting Changes to Temporal Data

Change tracking for timeline entity sets works identical to non-temporal entity sets. If the entity set supports change-tracking combined with filtering on application-time period boundaries, the corresponding declared properties SHOULD be listed as `FilterableProperties`, see **[OData-VocCap]**. Clients can then use these properties in `$filter` or use the convenience shortcuts `$at` or `$from` and `$to`/`$toInclusive`.

Change tracking for snapshot entity sets only reports changes to time slices that contain the point in time specified via `$at`, or the point in time at which the defining query was received if no `$at` is specified. Mere passage of time does not lead to reported changes.

4.3 Modifying Temporal Data

This section and its subsections describe modifications in application time, both for

- *snapshot entity sets* with hidden application time, where time slices exist in the data persistency but are not directly visible in the entity set, each OData entity corresponds to a temporal object and only represents data at one point in time, and for
- *timeline entity sets* with visible application time, where each OData entity corresponds to an application-time slice and all application-time slices of a temporal object are part of the entity set.

Modification operations fall into two categories:

- Direct modification of time slices, and
- Changes to a temporal object over a period of application time that can affect multiple time slices without explicitly addressing each single affected time slice.

4.3.1 Direct Modification of Time Slices

The temporal query options `$at`, `$from` and `$to`/`$toInclusive` can be used with data modification operations, in which case they do not alter the modification semantics of the request, they only affect the response of the "implicit `GET`" after the data modification.

Modification of time slices SHOULD be done with the temporal actions defined in the next section and its subsections.

This specification does not define the behavior of standard insert, update, upsert, or delete requests on temporal entity sets, and potential side-effects of direct modification requests to period boundaries and adjacent time slices are beyond the scope of this specification as the underlying business logic will vary from service to service.

4.3.2 Operations on Temporal Objects

Changes to a temporal object over a period of application time can affect multiple time slices. For timeline entity sets this can in principle be achieved by constructing batch requests with multiple operations on individual time slices. This puts burden on the client, and it also makes it harder for services to check whether the request represents a consistent change of the temporal data.

Both can be avoided by providing convenience operations expressing the intent of the modification and leaving the actual time slice manipulation to the service. These operations can then also be used with snapshot entity sets that do not allow direct manipulation of time slices.

These convenience operations are modeled as bound actions and defined in the vocabulary for temporal data **[OData-VocTemporal]**. Implementations SHOULD consider the preferences `return=representation` and `return=minimal` as specified inv**[OData-Protocol]**. The convenience operations are atomic (all or nothing): they either succeed and produce the result described below, or they fail and do not change the temporal objects.

All actions define a collection of entities as their binding parameter. This collection can be

- A snapshot entity set
- A timeline entity set containing time slices for a single temporal object, or
- A timeline entity set containing time slices for multiple temporal objects.

Services MAY support any or all of these convenience actions. Services SHOULD advertise the supported actions with property `SupportedActions` of term `ApplicationTimeSupport`.

Services may define specialized convenience operations for use cases not covered by this specification.

All actions return a collection of triples that consist of period start and end and a time slice, which has the same entity set as the binding parameter and advertises this through `odata.context` control information.

4.3.2.1 Update during a Period

The `Update` action updates existing time slices with values from delta time slices whose temporal object keys match and whose periods overlap.

Its non-binding parameter `deltaTimeslices` is a collection of a structure containing the period boundaries, the properties to update, and optionally temporal object key values. The period boundaries are interpreted according to the `UnitOfTime` of the collection. In particular, `ClosedClosedPeriods` governs whether the period end of type `Edm.Date` is the last day in the period or the first day after the period. The upper period boundary for items in `deltaTimeslices` need not be supplied; if it has no declared default value, it defaults to `max`.

This works identical to the SQL statement UPDATE FOR PORTION OF:

1. The “delta time slices” in `deltaTimeslices` are processed in the order of the collection.
2. For each delta time slice all time slices from the bound collection are selected whose temporal object key values are identical to the values of corresponding properties present in the delta time slice, and whose application-time period overlaps with the period of the delta time slice.
3. Selected time slices whose period is not fully included in the period of the delta time slice are split into two or three consecutive time slices, one with fully included period, and one or two with a non-overlapping period immediately before or after the delta time slice’s period.

4. Then all fully included time slices (including ones created in the previous step) are updated following the rules for updating entities specified in **[OData-Protocol]**.
5. Gaps between selected time slices in the period to update are not affected.

On success it returns the created or updated time slices.

Example 18: Change a department's budget during a period of application time with `api-2` (visible timeline)

```
POST /api-2/Departments('D08')/history/Temporal.Update
Content-Type: application/json

{
  "deltaTimeslices": [
    {
      "Timeslice": {
        "From": "2012-04-01",
        "To": "2014-07-01",
        "Budget": 1320
      }
    }
  ]
}
```

Given the example departments data the operation will split the time slice starting at 2012-01-01 creating a new time slice starting at 2012-04-01 which is then updated with the desired budget. It will then update the time slice starting at 2012-06-01 with the desired budget, and finally it will split the time slice starting at 2014-01-01 creating a new time slice starting at 2014-07-01, then update the original with the desired budget, leaving the new time slice starting at 2014-07-01 untouched:

Departments (after)

ID	From	To	Name	Budget
D08	2010-01-01	2012-01-01	Support	1000
D08	2012-01-01	***2012-04-01***	Support	1250
D08	***2012-04-01***	***2012-06-01***	***Support***	***1320***
D08	2012-06-01	2014-01-01	1st Level Support	***1320***
D08	2014-01-01	***2014-07-01***	1st Level Support	***1320***
D08	***2014-07-01***	***max***	***1st Level Support***	***1400***
D15	2010-01-01	2011-01-01	Services	1100
D15	2011-01-01	max	Services	1170

It returns the resulting created or updated time slices

```
{
  "@context": "../../$metadata#Collection(Temporal.TimesliceWithPeriod)",
  "value": [
    {
      "Timeslice": {
        "@context": "#Departments('D08')/history/$entity",
        "From": "2012-01-01",
        "To": "2012-04-01",
        "Name": "Support",
        "Budget": 1250
      }
    },
    {
      "Timeslice": {
        "@context": "#Departments('D08')/history/$entity",
        "From": "2012-04-01",
        "To": "2012-06-01",
        "Name": "Support",
        "Budget": 1320
      }
```

```
    },
    {
      "Timeslice": {
        "@context": "#Departments('D08')/history/$entity",
        "From": "2012-06-01",
        "To": "2014-01-01",
        "Name": "1st Level Support",
        "Budget": 1320
      }
    },
    {
      "Timeslice": {
        "@context": "#Departments('D08')/history/$entity",
        "From": "2014-01-01",
        "To": "2014-07-01",
        "Name": "1st Level Support",
        "Budget": 1320
      }
    },
    {
      "Timeslice": {
        "@context": "#Departments('D08')/history/$entity",
        "From": "2014-07-01",
        "To": "9999-12-31",
        "Name": "1st Level Support",
        "Budget": 1400
      }
    }
  ]
}
```

The service could have joined the third and fourth time slice because they do not significantly differ.

Example 19: Change an employee's job title during a period of application time with `api-1` (snapshot). Note that the period boundaries are not visible in `api-1` and are provided as `PeriodStart` and `PeriodEnd` next to the `Timeslice` data. The `PeriodEnd` is omitted, meaning the end of application time.

```
POST /api-1/Employees/Temporal.Update
Content-Type: application/json

{
  "deltaTimeslices": [
    {
      "PeriodStart": "2021-10-01",
      "Timeslice": {
        "ID": "E401",
        "Jobtitle": "Ultimate Expert"
      }
    }
  ]
}
```

Given the example employee data the operation will split the time slice for employee `E401` starting at 2012-03-01 creating a new time slice starting at 2021-10-01 which is then updated with the desired job title.

Employees (after)

ID	From	To	Name	Jobtitle	Department.ID
E314	2011-01-01	2013-10-01	McDevitt	Junior	D08
E314	2013-10-01	2014-01-01	McDevitt	Senior	D08
E314	2014-01-01	max	McDevitt	Senior	D15
E401	2009-11-01	2012-03-01	Norman	Expert	D15
E401	2012-03-01	***2021-10-01***	Gibson	Expert	D15

ID	From	To	Name	Jobtitle	Department.ID
E401	*2021-10-01*	*max*	*Gibson*	*Ultimate Expert*	*D15*

It returns the resulting created or updated time slices

```
{
  "@context": "../$metadata#Collection(Temporal.TimesliceWithPeriod)",
  "value": [
    {
      "PeriodStart": "2012-03-01",
      "PeriodEnd": "2021-10-01",
      "Timeslice": {
        "@context": "#Employees/$entity",
        "ID": "E401",
        "Name": "Gibson",
        "Jobtitle": "Expert"
      }
    },
    {
      "PeriodStart": "2021-10-01",
      "PeriodEnd": "9999-12-31",
      "Timeslice": {
        "@context": "#Employees/$entity",
        "ID": "E401",
        "Name": "Gibson",
        "Jobtitle": "Ultimate Expert"
      }
    }
  ]
}
```

4.3.2.2 Upsert during a Period

The `Upsert` action upserts existing time slices with values from delta time slices whose temporal object keys match and whose periods overlap, and in addition creates new time slices to close gaps in the matching time period.

It has the same signature as `Update`, and steps 1 to 4 work identical to `Update`. Step 5 is

5. Gaps between selected time slices in the period to update are closed by creating new adjacent time slices: for each gap that has an immediately preceding time slice, a new time slice is created by

    1. copying the property values of the preceding time slice (except for computed properties),
    2. setting the period boundaries to close the gap, and then
    3. updating the new time slice following the rules for updating entities specified in **[OData-Protocol]**.

    If no preceding time slice exists, the time slice is created following the rules for creating entities specified in **[OData-Protocol]**.

On success it returns the created or updated time slices.

Example 20: Upsert two cost centers during a period of application time using the model from example 8

```
POST /api-3/CostCenters/Temporal.Upsert
Content-Type: application/json

{
  "deltaTimeslices": [
    {
      "Timeslice": {
        "AreaID": "51",
        "CostCenterID": "C1",
        "ValidTo": "2001-03-31",
```

```
        "ValidFrom": "1984-04-01",
        "ProfitCenterID": "P2"
      }
    },
    {
      "Timeslice": {
        "AreaID": "51",
        "CostCenterID": "C2",
        "ValidFrom": "2012-04-01",
        "DepartmentID": "D04"
      }
    }
  ]
}
```


Given this example data (primary key in olive)

CostCenters (before)

tsid	AreaID	CostCenterID	ValidTo	ValidFrom	ProfitCenterID	DepartmentID
n	51	C1	max	1955-04-01	P1	D02

the operation will split the time slice `n` for cost center `C1` starting at 1955-04-01 creating two new time slices, the one starting at 1984-04-01 is then updated with the desired profit center. It will also insert a new time slice for the not yet existing cost center `C2` with the desired values.

Note that this example uses closed-closed periods, so each time slice ends the day before the start of the next adjacent time slice.

CostCenters (after)

tsid	AreaID	CostCenterID	ValidTo	ValidFrom	ProfitCenterID	DepartmentID
n	51	C1	***1984-03-31***	1955-04-01	P1	D02
o	***51***	***C1***	***2001-03-31***	***1984-04-01***	***P2***	***D02***
p	***51***	***C1***	***max***	***2001-04-01***	***P1***	***D02***
q	***51***	***C2***	***max***	***2012-04-01***		***D04***

It returns the resulting created or updated time slices per affected temporal object


```
{
  "@context": "../$metadata#Collection(Temporal.TimesliceWithPeriod)",
  "value": [
    {
      "Timeslice": {
        "@context": "#CostCenters/$entity",
        "tsid": "n",
        "AreaID": "51",
        "CostCenterID": "C1",
        "ValidTo": "1984-03-31",
        "ValidFrom": "1955-04-01",
        "ProfitCenterID": "P1",
        "DepartmentID": "D02"
      }
    },
    {
      "Timeslice": {
        "@context": "#CostCenters/$entity",
        "tsid": "o",
        "AreaID": "51",
        "CostCenterID": "C1",
        "ValidTo": "2001-03-31",
        "ValidFrom": "1984-04-01",
        "ProfitCenterID": "P2",
        "DepartmentID": "D02"
      }
    },
```

```
    {
      "Timeslice": {
        "@context": "#CostCenters/$entity",
        "tsid": "p",
        "AreaID": "51",
        "CostCenterID": "C1",
        "ValidTo": "9999-12-31",
        "ValidFrom": "2001-04-01",
        "ProfitCenterID": "P1",
        "DepartmentID": "D02"
      }
    },
    {
      "Timeslice": {
        "@context": "#CostCenters/$entity",
        "tsid": "q",
        "AreaID": "51",
        "CostCenterID": "C2",
        "ValidTo": "9999-12-31",
        "ValidFrom": "2012-04-01",
        "ProfitCenterID": null,
        "DepartmentID": "D04"
      }
    }
  ]
}
```

4.3.2.3 Delete during a Period

The `Delete` action deletes (sub-periods of) time slices.

Its non-binding parameter `deltaTimeslices` is a collection of a structure containing the period boundaries and optionally temporal object key values. The period boundaries are interpreted according to the `UnitOfTime` of the collection. In particular, `ClosedClosedPeriods` governs whether the period end of type `Edm.Date` is the last day in the period or the first day after the period.

This works identical to the SQL statement DELETE FOR PORTION OF:

1. The “delta time slices” in `deltaTimeslices` are processed in the order of the collection.
2. For each delta time slice all time slices from the bound collection are selected whose temporal object key values are identical to the values of corresponding properties present in the delta time slice, and whose application-time period overlaps with the period of the delta time slice.
3. Selected time slices whose period is not fully included in the period of the delta time slice are split into two consecutive time slices, one with non-overlapping, and one with fully included period.
4. Then all fully included time slices (including ones created in the previous step) are deleted following the rules for deleting entities specified in **[OData-Protocol]**.

On success it returns the deleted time slices.

4.3.3 ETags

Timeline entity sets need no special consideration as each time slice is an OData entity.

This specification does not define the behavior of standard insert, update, upsert, or delete requests for snapshot entity sets and consequently defines no rules for ETags of entities in snapshot entity sets. Services supporting these requests are advised to associate ETags with time slices rather than with temporal objects.

4.3.4 Read, Edit, Navigation, and Association URLs

Timeline entity sets need no special consideration as each time slice is an OData entity.

For snapshot entity sets read, edit, navigation, and association URLs are not specific to time slices, they are specific to the time-independent OData entity.

---

5 Conformance

Conforming services MUST follow all rules of this specification for the temporal system query options they support.

Conforming clients MUST be prepared to consume a model that uses any or all constructs defined in this specification.

Appendix A. References

This appendix contains the normative and informative references that are used in this document.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[OData-ABNF]

ABNF components: OData ABNF Construction Rules Version 4.02 and OData ABNF Test Cases.
See link in “Related work” section on cover page.

[OData-Aggregation]

OData Extension for Data Aggregation Version 4.0.
See link in “Related work” section on cover page.

[OData-CSDL]

OData Common Schema Definition Language (CSDL) JSON Representation Version 4.02.
See link in “Related work” section on cover page.

OData Common Schema Definition Language (CSDL) XML Representation Version 4.02.
See link in “Related work” section on cover page.

[OData-Protocol]

OData Version 4.02. Part 1: Protocol.
See link in “Related work” section on cover page.

[OData-URL]

OData Version 4.02. Part 2: URL Conventions.
See link in “Related work” section on cover page.

[OData-VocCap]

OData Vocabularies Version 4.0: Capabilities Vocabulary.
See link in “Related work” section on cover page.

[OData-VocCore]

OData Vocabularies Version 4.0: Core Vocabulary.
See link in “Related work” section on cover page.

[OData-VocTemporal]

OData Temporal Vocabulary.
See link in “Additional artifacts” section on cover page.

[RFC2119]

Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997
https://www.rfc-editor.org/info/rfc2119.

[RFC8174]

Leiba, B., “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words”, BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017
https://www.rfc-editor.org/info/rfc8174.

A.2 Informative References

[Fowler]

Martin Fowler, “Temporal Patterns”, 16 February 2005
http://martinfowler.com/eaaDev/timeNarrative.html.

[Kulkarni]

Krishna Kulkarni, “Temporal Features in SQL standard”, September 2012
https://dbs.uni-leipzig.de/file/Temporal%20features%20in%20SQL2011.pdf.

[Snodgrass]

Richard T. Snodgrass, “Developing Time-Oriented Database Applications in SQL”, Morgan Kaufmann Publishers, Inc., San Francisco, July, 1999, ISBN 1-55860-436-7
http://www2.cs.arizona.edu/people/rts/tdbbook.pdf and http://www2.cs.arizona.edu/people/rts/pp30-31.pdf.

[SQL:2011]

ISO/IEC 9075-2:2011 Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation).

Appendix B. Acknowledgments

B.1 Special Thanks

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

Special thanks to Andrew Eisenberg, whose contributions in the early stages of the OData TC were invaluable to getting this extension specification on track.

B.2 Participants

OData TC Members:

First Name	Last Name	Company
George	Ericson	Dell
Hubert	Heijkers	IBM
Ling	Jin	IBM
Stefan	Hagen	Individual
Michael	Pizzo	Microsoft
Christof	Sprenger	Microsoft
Ralf	Handl	SAP SE
Gerald	Krause	SAP SE
Heiko	Theißen	SAP SE
Mark	Biamonte	Progress Software
Martin	Zurmuehl	SAP SE

Appendix C. Revision History

Revision	Date	Editor	Changes Made
Working Draft 01	2013-09-05	Ralf Handl	Initial working draft
Committee Specification Draft 01	2021-07-22	Ralf Handl Hubert Heijkers Gerald Krause Michael Pizzo Heiko Theißen Martin Zurmuehl	Timeline entity sets for manipulating time slices Snapshot entity sets for hiding application-time dependency Modification during a period via actions Closed-closed application time
Committee Specification Draft 02	2021-10-15	Ralf Handl	Example model with object key Example for Update action on snapshot entity set Example for Upsert action with object key Semantics of omitting upper period boundaries in action parameters
Committee Specification Draft 03	2021-12-02	Ralf Handl	Removed the “stop on change” actions Minor clarifications
Committee Specification 01	2022-01-07	Ralf Handl	Non-material changes

Appendix D. 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.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, Candidate OASIS Standard, OASIS Standard, or Approved Errata).

[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 Standards Final Deliverable, 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 deliverable.]

[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 OASIS Standards Final Deliverable 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 OASIS Standards Final Deliverable. 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 OASIS Standards Final Deliverable 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 Standards Final Deliverable, 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.