Open Geospatial Consortium

Submission Date: <2023-05-19>

Approval Date:<yyyy-mm-dd>

Publication Date:<yyyy-mm-dd>

External identifier of this OGC document: http://www.opengis.net/doc/IS/ogcapi-movingfeatures-1/1.0.draft

Internal reference number of this OGC document:22-003

Version: 1.0.draft

Category: OGC Implementation Specification

Editor: Taehoon Kim, Kyoung-Sook Kim, Mahmoud SAKR, Martin Desruisseaux

OGC API – Moving Features – Part 1: Core

Warning

This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC Implementation Specification
Document stage: Draft
Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Table of Contents

i. Abstract

Moving feature data can represent various phenomena, including vehicles, people, animals, weather patterns, etc. The OGC API Ñ Moving Features (OGC API Ñ MF) is a Draft Standard defines a standard interface for querying and accessing geospatial data that changes over time, such as the location and attributes of moving objects like vehicles, vessels, or pedestrians. The OGC API Ñ MF provides a standard way to manage these data, which can be helpful for applications such as transportation management, disaster response, and environmental monitoring. OGC API Ñ MF also includes operations for filtering, sorting, and aggregating moving feature data based on location, time, and other properties.

The OGC API Ñ Moving Features Ñ Part 1: Core specifies a set of RESTful web service interfaces and data formats for querying and updating moving feature data over the web. OGC API Standards define modular API building blocks to spatially enable Web APIs in a consistent way. OpenAPI is used to define the reusable API building blocks with responses in JSON and HTML.

The OGC API family of standards is organized by resource type.

Table 1. Overview of Resources

Resource	Path	HTTP Method	Document Reference
Collections metadata	`/collections`	GET, POST	Resource Collections
Collection instance metadata	`/collections/{collectionId}`	GET, DELETE, PUT	Resource Collection
MovingFeatures	`/collections/{collectionId}/items`	GET, POST	Resource MovingFeatures
MovingFeature instance	`/collections/{collectionId}/items/{mFeatureId}`	GET, DELETE	Resource MovingFeature
TemporalGeometrySequence	`/collections/{collectionId}/items/{mFeatureId}/tgsequence`	GET, POST	Resource TemporalGeometrySequence
TemporalPrimitiveGeometry instance	`/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}`	DELETE	Resource TemporalPrimitiveGeometry
Queries for TemporalPrimitiveGeometry	`collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}`	GET	TemporalGeometry Query Resources
TemporalProperties instance	`/collections/{collectionId}/items/{mFeatureId}/tproperties`	GET, POST	Resource TemporalProperties
TemporalProperty instance	`/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyId}`	GET, POST	Resource TemporalProperty

ii.Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, OGC Moving Features, OGC Moving Features JSON, Moving Features Access, API, OpenAPI, REST, trajectory

iii. Preface

OGC Declaration

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

iv. Security Considerations

The OGC API Ñ Moving Features Ñ Part 1: Core Draft Standard does not mandate any specific security controls. However, it was constructed to add security controls without impacting conformance, the same as the OGC API Ñ Common Ñ Part 1: Core.

This document applied the Requirement /req/oas30/security for OpenAPI 3.0 Security support.

v. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology
- UniversitŽ libre de Bruxelles
- Geomatys
- Central Research Laboratory, Hitachi Ltd.
- Feng Chia University

vi. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	Organization
Kyoung-Sook KIM	Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology

Taehoon KIM	Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology
Mahmoud SAKR	UniversitŽ libre de Bruxelles
Esteban Zimanyi	UniversitŽ libre de Bruxelles
Martin Desruisseaux	Geomatys
Akinori Asahara	Central Research Laboratory, Hitachi Ltd.
Chen-Yu Hao	Feng Chia University

Chapter 1. Scope

The scope of the OGC API Ñ Moving Features Ñ Part 1:Core is to provide a uniform way to access, communicate, and manage data about moving features across different applications, data providers, and data consumers. The OGC API Ñ MF defines a set of API building blocks that enable clients to discover, retrieve, and update information about moving features, as well as a data model for describing moving features and their trajectories.

The OGC API Ñ Moving Features Ñ Part 1:Core Draft Standard defines an API with two goals. First, to provide access to representations of Moving Features that conform to the OGC Moving Features JSON Encoding Standard. Second, to provide functionality comparable to that of the OGC Moving Features Access Standard. The OGC API Ñ Moving Features Draft Standard is an extension of the OGC API Ñ Common and the OGC API Ñ Features Standards.

Chapter 2. Conformance

This Standard defines two requirements / conformance classes that describe different levels of compliance with the Standard. These requirements / conformance classes help to ensure interoperability between other implementations of the Standard and allow data providers to specify which parts of the Standard they support. The standardization target is "Web APIs".

The conformance classes for OGC API Ñ Moving Features are:

- Collection Catalog
- Moving Features

The conformance class defines the minimum requirements for an API to be compliant with the OGC API Ñ Moving Features Draft Standard. This includes support for querying and retrieving information about moving features using HTTP GET requests. Also, the conformance class enables clients to add, modify, or delete features from the server using HTTP POST, PUT, and DELETE requests. Lastly, the conformance class adds support for querying and retrieving features based on their temporal characteristics, such as their position at a specific time or their velocity over a given time interval.

Implementers of the OGC API Ñ MF can choose which conformance classes they want to support based on the specific needs of their use case and the capabilities of their software. However, to be considered compliant with the Standard, an implementation shall support at least the Core conformance class.

The URIs of the associated conformance classes are:

Table 2. Conformance class URIs

Conformance class	URI
MovingFeatures Collection Catalog	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/mf-collection
MovingFeatures	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/movingfeatures

Conformance with this Standard shall be checked using all the relevant tests specified in Annex A of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing website.

Chapter 3. References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

NOTE

- OGC 08-131r3:¦Ñ¦The Specification Model¦Ñ¦A Standard for Modular Specifications (2009). https://portal.opengeospatial.org/files/?artifact_id=34762
- OGC 16-120r3: OGC Moving Features Access (2017). https://docs.ogc.org/is/16-120r3/16-120r3.html
- OGC 19-045r3: OGC Moving Features Encoding Extension - JSON (2020). https://docs.ogc.org/is/19-045r3/19-045r3.html
- OGC 17-069r4: OGC API Ñ Features Ñ Part 1: Core (2022). https://docs.opengeospatial.org/is/17-069r4/17-069r4.html
- OGC 20-002: OGC API Ñ Features Ñ Part 4: Create, Replace, Update and Delete (Draft). http://docs.ogc.org/DRAFTS/20-002.html
- OGC 19-072: OGC API Ñ Common Ñ Part 1: Core (2023). https://docs.ogc.org/is/19-072/19-072.html
- OGC 20-024: OGC API Ñ Common Ñ Part 2: Geospatial Data (Draft). http://docs.ogc.org/DRAFTS/20-024.html
- IETF RFC 2387: The MIME Multipart/Related Content-type. https://datatracker.ietf.org/doc/html/rfc2387
- IETF RFC 2818: HTTP Over TLS. https://datatracker.ietf.org/doc/html/rfc2818
- IETF RFC 3339: Date and Time on the Internet: Timestamps. https://datatracker.ietf.org/doc/html/rfc3339
- IETF RFC 3986: Uniform Resource Identifier (URI): Generic Syntax. https://datatracker.ietf.org/doc/html/rfc3986
- IETF RFC 7230 to TFC 7235: Hypertext Transfer Protocol (HTTP/1.1). https://datatracker.ietf.org/doc/html/rfc7230, https://datatracker.ietf.org/doc/html/rfc7231, https://datatracker.ietf.org/doc/html/rfc7232, https://datatracker.ietf.org/doc/html/rfc7233, https://datatracker.ietf.org/doc/html/rfc7234, and https://datatracker.ietf.org/doc/html/rfc7235
- IETF RFC 7946: The GeoJSON Format. https://datatracker.ietf.org/doc/html/rfc7946
- IETF RFC 8288: Web Linking. https://datatracker.ietf.org/doc/html/rfc8288
- IETF RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format. https://datatracker.ietf.org/doc/html/rfc8259
- Open API Initiative: OpenAPI Specification, Version 3.0. The latest patch version at the time of publication of this standard was 3.1.0. https://spec.openapis.org/oas/v3.1.0.

Chapter 4. Terms and Definitions

This document used the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word Òshall Ó (not Òmust Ó) is the verb form used to indicate a requirement to be strictly followed to conform to this standard and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the 'ModSpec'. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

application programming interface (API)

An interface that is defined in terms of a set of functions and procedures, and enables a program to gain access to facilities within an application.
[source: Dictionary of Computer Science - Oxford Quick Reference, 2016]

coordinate

one of a sequence of numbers designating the position of a point

Note 1 to entry: In a spatial coordinate reference system, the coordinate values are qualified by units.

[source: ISOÎ19111]

coordinate reference system (CRS)

coordinate system that is related to an object by a datum

Note 1 to entry: Geodetic and vertical datums are referred to as reference frames.

Note 2 to entry: For geodetic and vertical reference frames, the object will be the Earth. In planetary applications, geodetic and vertical reference frames may be applied to other celestial bodies.

[source: ISOÎ19111]

dataset

collection of data, published or curated by a single agent, and available for access or download in one or more formats
[source: DCAT]

datatype

specification of a value domain with operations allowed on values in this domain

Examples: `Integer`, `Real`, `Boolean`, `String` and `Date`.

Note 1 to entry: Data types include primitive predefined types and user definable types.

[source: ISOÎ19103]

distribution

represents an accessible form of a dataset

Note 1 to entry: EXAMPLE: a downloadable file, an RSS feed or a web service that provides the data.

[source: DCAT]

dynamic attribute

characteristic of a feature in which its value varies with time

[source: OGC 19-045r3]

feature

abstraction of a real-world phenomena

Note 1 to entry: A feature can occur as a type or an instance. Feature type or feature instance should be used when only one is meant.

[source: ISOÊ19109]

feature attribute

characteristic of a feature

Note 1 to entry: A feature attribute can occur as a type or an instance. Feature attribute type or feature attribute instance is used when only one is meant.

[source: ISOÊ19109]

feature table

table where the columns represent feature attributes, and the rows represent features

[source: OGC 06-104r4]

geographic feature

representation of real-world phenomenon associated with a location relative to the Earth

[source: ISOÊ19101-2]

geometric object

spatial object representing a geometric set

[source: ISOÊ19107:2003]

leaf

<one parameter set of geometries>

geometry at a particular value of the parameter

[source: ISOÊ19141]

moving feature

feature whose position changes over time

Note 1 to entry: Its base representation uses a local origin and local coordinate vectors of a geometric object at a given reference time. [source: ISOÊ19141]

Note 2 to entry: The local origin and ordinate vectors establish an engineering coordinate reference system (ISO 19111), also called a local frame or a local Euclidean coordinate system.[source: ISOÊ19141]

[source: OGC 19-045r3]

property

facet or attribute of an object referenced by a name

[source: ISOÊ19143]

resource

entity that might be identified

Note 1 to entry: The term Òresource Ó, when used in the context of an OGC Web API standard, should be understood to

mean a web resource unless otherwise indicated.

[source: Dublin Core Metadata Initiative¦Ñ¦DCMI Metadata Terms]

resource type

a type of resource

Note 1 to entry: Resource types are re-usable components that are independent of where the resource resides in the API.

[source: OGC 19-072]

trajectory

path of a moving point described by a one parameter set of points

[source: ISO¦19141]

web API

API using an architectural style that is founded on the technologies of the Web

[source: W3C Data on the Web Best Practices]

web resource

a resource that is identified by a URI.

[source: OGC 17-069r4]

Chapter 5. Conventions

This section provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

5.1. Identifiers

The normative provisions in this Standard are denoted by the URI

http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

5.2. Use of HTTPS

For simplicity, this OGC Standard only refers to the HTTP protocol. This is not meant to exclude the use of HTTPS. This is simply a shorthand notation for ÒHTTP or HTTPSÓ. In fact, most servers are expected to use HTTPS and not HTTP.

Chapter 6. Overview

6.1. General

The OGC API Ñ Features Standard enable access to resources using the HTTP protocol and its associated operations (GET, PUT, POST, DELETE, etc.) The OGC API Ñ Common Standard defines a set of features which are applicable to all OGC APIs. Other OGC standards extend OGC API Ñ Common with features specific to a resource type.

This OGC API Ñ Moving Features Ñ Part1: Core Draft Standard defines an API with the goal to:

- Provide a standard interface for creating (HTTP POST), retrieving (HTTP GET), updating (HTTP PUT), and deleting (HTTP DELETE) *Moving Features*, with conformance to the OGC Moving Features JSON Encoding Standard

Resources exposed through an OGC API may be accessed via a Universal Resource Identifier (URI). The URI representation in this Draft Standard is composed of three sections:

- Dataset distribution API: The endpoint corresponding to a dataset distribution, where the landing page resource as defined in OGC API Ñ Common Ñ Part 1: Core is available (subsequently referred to as Base URI or `{root}`).
- Access Paths: Unique paths to Resources.
- Query Parameters: Parameters to adjust the representation of a Resource or Resources like encoding format or sub-setting.

Access Paths are used to build resource identifiers. This approach is recommended, but not required. Most resources are also accessible through links to previously accessed resources. Unique relation types are used for each resource.

Table 3 summarizes the access paths and relation types defined in this Standard.

Table 3. Moving Features API Paths

Path Template	Relation	Resource
Collections		
{root}/collections	`data`	Metadata describing the Collection Catalog of data available from this API.
{root}/collections/{collectionId}		Metadata describing the Collection Catalog of data which has the unique identifier `{collectionId}`
MovingFeatures		
{root}/collections/{collectionId} /items	`items`	Static information of MovingFeature about available items in the specified Collection
{root}/collections/{collectionId} /items/{mFeatureId}	`item`	Static information describing the MovingFeature of data which has the unique identifier `{mFeatureId}`

Path Template	Relation	Resource
{root}/collections/{collectionId} /items/{mFeatureId}/tgsequence	items	Sequence of TemporalPrimitiveGeometry about available items in the specified MovingFeature
{root}/collections/{collectionId} /items/{mFeatureId}/tgsequence /{tGeometryId}	item	Temporal object describing the TemporalPrimitiveGeometry of data which has the unique identifier {tGeometryId}
{root}/collections/{collectionId} /items/{mFeatureId}/tgsequence /{tGeometryId}/{queryType}		Identifies an Information Resource of type {queryType} associated with the TemporalPrimitiveGeometry instance
{root}/collections/{collectionId} /items/{mFeatureId}/tproperties	items	Temporal object information of TemporalProperties about available items in the specified MovingFeature
{root}/collections/{collectionId} /items/{mFeatureId}/tproperties /{tPropertyName}	item	Temporal object describing the TemporalProperty of data which has the unique identifier {tPropertyName}

Where:

- {root} = Base URI for the API server
- {collectionId} = An identifier for a specific Collection of data
- {mFeatureId} = An identifier for a specific MovingFeature of a specific Collection of data
- {tGeometryId} = An identifier for a specific TemporalPrimitiveGeometry of a specific MovingFeature of data
- {tPropertyName} = An identifier for a specific TemporalProperty of a specific MovingFeatures of data

Figure 1 shows a UML class diagram for OGC API Ñ Moving Features (OGC API Ñ MF) which represents the basic resources of this Standard, such as Collections, Collection, MovingFeatures, MovingFeature, TemporalGeometrySequence, TemporalPrimitiveGeometry, TemporalProperties, and TemporalProperty. In this Standard, a single moving feature can have temporal geometries, such as a set of trajectories. Also, the moving feature can have multiple temporal properties, and each property can have a set of parametric values.

Figure 1. Class diagram for OGC API Ñ Moving Features

6.2. Search

The core search capability is based on OGC API Ñ Common and thus supports:

- bounding box searches,
- time instant or time period searches, and
- equality predicates (i.e. *property=value*).

OGC API Ñ Moving Features extends these core search capabilities to include:

- spatiotemporal queries for accessing TemporalGeometry resources.

6.3. Dependencies

The OGC API Ñ Moving Features (OGC API Ñ MF) Draft Standard is an extension of the OGC API Ñ Common and the OGC API Ñ Features Standards. Therefore, an implementation of OGC API Ñ MF shall first satisfy the appropriate Requirements Classes from OGC API Ñ Common and OGC API Ñ Features. Also, the OGC API Ñ MF Standard is based on the OGC Moving Features Encoding Extension for JSON (OGC MF-JSON) Standards. Therefore, an implementation of OGC API Ñ MF shall satisfy the appropriate Requirements Classes from OGC MF-JSON. Table 4, Identifies the OGC API Ñ Common and OGC API Ñ Features Requirements Classes which are applicable to each section of this Standard. Instructions on when and how to apply these Requirement Classes are provided in each section.

Table 4. Mapping OGC API Ñ MF Sections to OGC API Ñ Common, OGC API Ñ Features, and OGC MF-JSON Requirements Classes

API Ñ MF Section	API Ñ MF Requirements Class	API Ñ Common, API Ñ Features, MF-JSON Requirements Class
Collections	/req/mf-collection	http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections, http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete
MovingFeatures	/req/movingfeatures	http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core, http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete, http://www.opengis.net/spec/movingfeatures/json/1.0/req/trajectory, http://www.opengis.net/spec/movingfeatures/json/1.0/req/prism
HTML	inherit all requirement (no modification)	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/html
JSON	inherit all requirement (no modification)	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json
GeoJSON	inherit all requirement (no modification)	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson
OpenAPI 3.0	inherit all requirement (no modification)	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30

Chapter 7. Requirements Class "Moving Feature Collection Catalog"

7.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections
Dependency	http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete

The `Moving Feature Collection Catalog` requirements class defines the requirements for a moving feature collection. A moving feature collection is an object that provides information about and access to a set of related Moving Features.

7.2. Information Resources

The two resources defined in this Requirement Class are summarized in Table 5.

Table 5. Moving Feature Collection Catalog Resources

Resource	URI	HTTP Method	Description
Collections	`{root}/collections`	GET	Get information which describes the set of available Collections resource
		POST	Add a new resource (Collection) instance to a Collections resource
Collection	`{root}/collections/{collectionId}`	GET	Get information about a specific Collection resource ({collectionId}) of geospatial data
		PUT	Update information about a specific Collection resource ({collectionId})
		DELETE	Delete a specific Collection resource ({collectionId})

7.3. Resource Collections

7.3.1. Overview

The Collections resource supports retrieving and creating operations via GET and POST HTTP methods respectively.

1. A retrieve operation returns a set of metadata which describes the collections available from this API.
2. A create operation posts a new Collection resource instance to the collections with this API.

7.3.2. Operation

Retrieve

The retrieve operation is defined in the Collections conformance class of OGC API Ñ Common. No modifications are needed to support MovingFeature resources.

1. Issue a GET request on {root}/collections path

Support for the HTTP GET method on the {root}/collections path is specified as a requirement in OGC API Ñ Common.

Requirement 1	/req/mf-collection/collections-get
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Common Collections operation requirement /req/collections/rc-md-op.

Create

The create operation is defined in the CREATE conformance class of OGC API Ñ Features. This operation targeted Collection resource.

1. Issue a POST request on {root}/collections path

Support for the HTTP POST method is specified as a requirement in OGC API Ñ Features.

Requirement 2	/req/mf-collection/collections-post
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE request body requirements /req/create-replace-delete/insert-body and /req/create-replace-delete/insert-content-type.
C	The content of the request body SHALL be based upon the Collection request body schema.

Collection Request Body Schema:

```
type: object
required:
Ê - itemType
properties:
Ê title:
```

```
Ê   description: human readable title of the collection
Ê   type: string
Ê updateFrequency:
Ê   description: a time interval of sampling location. The unit is millisecond.
Ê   type: number
Ê description:
Ê   description: any description
Ê   type: string
Ê itemType:
Ê   description: indicator about the type of the items in the moving features
collection (the default value is 'movingfeature').
Ê   type: string
Ê   default: "movingfeature"
```

The following example adds a new feature (collection information object) to the feature collections. The feature is encoded as JSON. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Creating a New Collection:

```
Ê  Client                                                          Server
Ê    |                                                               |
Ê    | POST /collections   HTTP/1.1                                  |
Ê    | Content-Type: application/json                                |
Ê    |                                                               |
Ê    | {                                                             |
Ê    |   "title": "MovingFeatureCollection_1",                       |
Ê    |   "updateFrequency": 1000,                                    |
Ê    |   "description": "a collection of moving features to manage data  |
Ê    |                  in a distinct (physical or logical) space",  |
Ê    |   "itemType": "movingfeature"                                 |
Ê    | }                                                             |
Ê    |-------------------------------------------------------------->|
Ê    |                                                               |
Ê    | HTTP/1.1 201 Created                                          |
Ê    | Location: /collections/mfc_1                                  |
Ê    |<--------------------------------------------------------------|
```

7.3.3. Response

Retrieve

A successful response to the Collections GET operation is a document that contains summary metadata for each collection accessible through an instance of an API implementation. In a typical deployment of the OGF API Ñ MF, the Collections GET response will list collections of all offered resource types. The collections where the value of the `itemType` property is MovingFeature are collections of moving features.

Requirement 3	/req/mf-collection/collections-get-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Common Collections response requirement /req/collections/rc-md-success.
B	The content of that response SHALL be based upon the Collections response schema.
C	The itemType property of the response schema SHALL be MovingFeature.

NOTE The usage of the itemType property is referred from the OGC API Ñ Common item Type section.

Collections GET Response Schema (collections.yaml):

```
type: object
required:
Ê - collections
Ê - links
properties:
Ê collections:
Ê   type: array
Ê   items:
Ê     $ref: 'collection.yaml'
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
```

The following JSON payload is an example of a response to an OGC API Ñ Moving Features Collections GET operation.

An Example of Collections JSON Payload:

```
{
Ê "collections": [
Ê   {
Ê     "id": "mfc-1",
Ê     "title": "MovingFeatureCollection_1",
Ê     "description": "a collection of moving features to manage data in a distinct
(physical or logical) space",
Ê     "itemType": "movingfeature",
Ê     "updateFrequency": 1000,
Ê     "extent": {
Ê       "spatial": {
Ê         "bbox": [
Ê           -180, -90, 190, 90
Ê         ],
```

```
Ê        "crs": "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
Ê      },
Ê      "temporal": {
Ê        "interval": [
Ê          "2011-11-11T12:22:11Z","2012-11-24T12:32:43Z"
Ê        ],
Ê        "trs": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
Ê      }
Ê    },
Ê    "links": [
Ê      {
Ê        "href": "https://data.example.org/collections/mfc-1",
Ê        "rel": "self",
Ê        "type": "application/json"
Ê      }
Ê    ]
Ê  }
Ê ],
Ê "links": [
Ê  {
Ê    "href": "https://data.example.org/collections",
Ê    "rel": "self",
Ê    "type": "application/json"
Ê  }
Ê ]
}
```


Create

A successful response to the Collections POST operation is an HTTP status code.

Requirement 4	/req/mf-collection/collections-post-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

7.3.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

7.4. Resource Collection

7.4.1. Overview

A Collection information object is the set of metadata that describes a single collection. An abbreviated copy of this information is returned for each Collection in the {root}/collections GET

response.

The schema for the collection information object presented in this clause is an extension of the collection schema defined in OGC API Ñ Common and OGC API Ñ Features.

Table 6 defines the set of properties that may be used to describe a collection.

Table 6. Table of collection properties

Property	Requirement	Description
id	M	A unique identifier to the collection.
title	O	A human-readable name given to the collection.
description	O	A free-text description of the collection.
links	M	A list of links for navigating the API (e.g. link to previous or next pages; links to alternative representations, etc.)
extent	O	The spatiotemporal coverage of the collection.
itemType	M	Fixed to the value "movingfeature".
updateFrequency	O	A time interval of sampling location. The time unit of this property is millisecond.

NOTE The *id*, *title*, *description*, *links*, *extent*, and *itemsType* properties were inherited from OGC API Ñ Common and OGC API Ñ Features.

NOTE An update frequency is one of the most important properties of moving feature collection. The update frequency can be used to handle the continuity of the moving feature's trajectory.

Requirement 5	/req/mf-collection/mandatory-collection
A	A collection object SHALL contain all the mandatory properties listed in Table 6.

7.4.2. Operation

Retrieve

The retrieve operation is defined in the OGC API Ñ Common `Collection` conformance class. No modifications are required to support MovingFeature resources.

1. Issue a `GET` request on the `{root}/collections/{collectionId}` path

The `{collectionId}` path parameter is the unique identifier for a single collection offered by an API implementation instance. The list of valid values for `{collectionId}` is provided in the `/collections` response.

Support for the `{root}/collections/{collectionId}` path is required by OGC API Ñ Common.

Requirement 6	/req/mf-collection/collection-get
A	An implementation of OGC API Ñ MF SHALL comply with the OGC API Ñ Common Collection operation requirement http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections/src-md-op.

Replace

The replace operation is defined in the REPLACE conformance class of OGC API Ñ Features. This operation targeted Collection resource.

1. Issue a PUT request on {root}/collections/{collectionId} path

Support for the HTTP PUT method is specified as a requirement in OGC API Ñ Features.

Requirement 7	/req/mf-collection/collection-put
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature PUT operation requirement /req/create-replace-delete/update-put-op.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature PUT request body requirements /req/create-replace-delete/update-put-body and /req/create-replace-delete/update-put-content-type.
C	The content of the request body SHALL be based upon the Collection request body schema, except updateFrequency. If the updateFrequency is included in the request body, the server SHALL ignore it.

NOTE	Once set, the update frequency cannot be changed.

The following example replaces the feature created by the Create Example with a new feature (collection metadata without an update frequency). Once again, the replacement feature is represented as a JSON payload. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Replacing an Existing Collection:

```
Ê  Client                                                          Server
Ê    |                                                               |
Ê    | PUT /collections/mfc_1   HTTP/1.1                             |
Ê    | Content-Type: application/json                                |
Ê    |                                                               |
Ê    | {                                                             |
Ê    |   "title": "MovingFeatureCollection_2",                       |
Ê    |   "description": "Title is changed"                           |
Ê    | }                                                             |
Ê    |-------------------------------------------------------------->|
Ê    |                                                               |
```

```
Ê    | HTTP/1.1 204 OK                                                    |
Ê    |<-------------------------------------------------------------------|
```

Delete

The delete operation is defined in the DELETE conformance class of OGC API Ñ Features.

1. Issue a DELETE request on {root}/collections/{collectionId} path

Support for the HTTP DELETE method is specified as a requirement in OGC API Ñ Features.

Requirement 8	/req/mf-collection/collection-delete
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature DELETE operation requirement /req/create-replace-delete/delete/delete-op.

The following example deletes the feature created by the Create Example and replaced with a new feature in the Replace Example. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Deleting an Existing Collection:

```
Ê  Client                                                          Server
Ê    |                                                               |
Ê    | DELETE /collections/mfc_1   HTTP/1.1                          |
Ê    |-------------------------------------------------------------->|
Ê    |                                                               |
Ê    | HTTP/1.1 204 OK                                               |
Ê    |<--------------------------------------------------------------|
```

7.4.3. Response

Retrieve

A successful response to the Collection GET operation is a set of metadata that describes the collection identified by the {collectionId} parameter.

Requirement 9	/req/mf-collection/collection-get-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Common Collection response requirement /req/collections.
B	The response SHALL only include collection metadata selected by the request.
C	The content of that response SHALL be based upon the Collection response schema.

Requirement 9	/req/mf-collection/collection-get-success
D	The itemType property of the response schema SHALL be 'movingfeature'.

Collection GET Response Schema (collection.yaml)

```
type: object
required:
Ê - id
Ê - links
Ê - itemType
properties:
Ê id:
Ê   description: identifier of the collection used, for example, in URIs
Ê   type: string
Ê   example: 'address'
Ê title:
Ê   description: human readable title of the collection
Ê   type: string
Ê   example: 'address'
Ê description:
Ê   description: a description of the features in the collection
Ê   type: string
Ê   example: 'An address.'
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
Ê   example:
Ê     - href: https://data.example.com/buildings
Ê       rel: item
Ê     - href: https://example.com/concepts/buildings.html
Ê       rel: describedby
Ê       type: text/html
Ê extent:
Ê   $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/extent.yaml'
Ê itemType:
Ê   description: indicator about the type of the items in the collection
Ê   type: string
Ê   default: 'movingfeature'
Ê crs:
Ê   description: the list of coordinate reference systems supported by the service
Ê   type: array
Ê   items:
Ê     type: string
Ê   default:
Ê     - 'https://www.opengis.net/def/crs/OGC/1.3/CRS84'
Ê   example:
```

```
Ê     - 'https://www.opengis.net/def/crs/OGC/1.3/CRS84'
Ê     - 'https://www.opengis.net/def/crs/EPSG/0/4326'
Ê updateFrequency:
Ê   description: a time interval of sampling location. The unit is millisecond.
Ê   type: number
```

The following JSON payload is an example of a response to an OGC API Ñ Moving Features Collection GET operation.

An Example of Collection GET Operation:

```
{
Ê "id": "mfc-1",
Ê "title": "moving_feature_collection_sample",
Ê "itemType": "movingfeature",
Ê "updateFrequency": 1000,
Ê "extent": {
Ê   "spatial": {
Ê     "bbox": [
Ê       -180, -90, 190, 90
Ê     ],
Ê     "crs": [
Ê       "http://www.opengis.net/def/crs/OGC/1.3/CRS84"
Ê     ]
Ê   },
Ê   "temporal": {
Ê     "interval": [
Ê       "2011-11-11T12:22:11Z","2012-11-24T12:32:43Z"
Ê     ],
Ê     "trs": [
Ê       "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
Ê     ]
Ê   }
Ê },
Ê "links": [
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1",
Ê     "rel": "self",
Ê     "type": "application/json"
Ê   }
Ê ]
}
```

Replace

A successful response to the Collection PUT operation is an HTTP status code.

Requirement 10	/req/mf-collection/collection-put-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature PUT response requirement /req/create-replace-delete/update-put-response.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature PUT exception requirement /req/create-replace-delete/update-put-rid-exception.

Delete

A successful response to the Collection DELETE operation is an HTTP status code.

Requirement 11	/req/mf-collection/collection-delete-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature DELETE response requirement /req/create-replace-delete/delete/response.
B	If no resource with the identifier exists in the collection, the server SHALL respond with a not-found exception (404).

7.4.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

Chapter 8. Requirements Class "MovingFeatures"

8.1. Overview

Requirements Class	
http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures	
Target type	Web API
Dependency	http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core
Dependency	http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete
Dependency	http://www.opengis.net/spec/movingfeatures/json/1.0/req/trajectory
Dependency	http://www.opengis.net/spec/movingfeatures/json/1.0/req/prism

The MovingFeatures requirements class defines the requirements for a moving feature. A moving feature is an object that provides information about and access to TemporalGeometry and TemporalProperties.

8.2. Information Resources

The seven resources defined in this Requirement Class are summarized in Table 7.

Table 7. MovingFeatures Resources

Resource	URI	HTTP Method
MovingFeatures	`{root}/collections/{collectionId}/items`	GET, POST
MovingFeature	`{root}/collections/{collectionId}/items/{mfeatureId}`	GET, DELETE
TemporalGeometry	`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence`	GET, POST
TemporalPrimitiveGeometry	`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}`	DELETE
TemporalGeometry Query	`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}`	GET
TemporalProperties	`{root}/collections/{collectionId}/items/{mFeatureId}/tproperties`	GET, POST

Resource	URI	HTTP Method
TemporalProperty	{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertiesName}	GET, POST

8.3. Resource MovingFeatures

8.3.1. Overview

The MovingFeatures resource supports retrieving and creating operations via GET and POST HTTP methods respectively.

1. A retrieve operation returns a set of features which describes the moving feature available from this API.
2. A create operation posts a new MovingFeature resource instance to a specific Collection (specified by {collectionId} with this API.

The OGC API Ñ MF Items query is an OGC API Ñ Features endpoint that may be used to catalog pre-existing moving features. If a {mFeatureID} is not specified, the query will return a list of the available moving features. The list of moving features returned to the response can be limited using the bbox, datetime, limit, and subTrajectory query parameters. This behavior and query parameters for use with the Items query are specified in OGC API Ñ Features and OGC API Ñ Common, except subTrajectory parameter.

8.3.2. Operation

Retrieve

The retrieve operation is defined in the Features conformance class of OGC API Ñ Features. Additional support for query parameter subTrajectory is needed to support MovingFeatures resource.

1. Issue a GET request on {root}/collections/{collectionID}/items path

Support for GET on the {root}/collections/{collectionID}/items path is required by OGC API Ñ Features.

Requirement 12	/req/movingfeatures/features-get
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Features Features operation requirement /req/core/fc-op.

Parameter subTrajectory

The subTrajectory query parameter is defined as follows:

Requirement 13	/req/movingfeatures/param-subtrajectory-definition

A	The operation SHALL support a query parameter `subtrajectory` with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	
	`name: subTrajectory`
	`in: query`
	`required: false`
	`schema:`
	`Ê type: array`
	`Ê uniqueItems: true`
	`Ê minItems: 2`
	`Ê maxItems: 2`
	`Ê items:`
	`Ê  type: string`
	`Ê  format: date-time`
	`style: form`
	`explode: false`

B	The `subTrajectory` parameter SHALL be a time interval (i.e., new start time(*subTrajectory[0]*) and new end time(*subTrajectory[1]*)) with `date-time` strings.
C	The new start time(*subTrajectory[0]*) in the `subTrajectory` parameter SHALL be earlier than the new end time(*subTrajectory[1]*), i.e., *subTrajectory[0]* < *subTrajectory[1]*.
D	The syntax of `date-time` is specified by RFC 3339, 5.6.

Requirement 14	/req/movingfeatures/param-subtrajectory-response
A	Only a subset of TemporalGeometry that intersects the time interval in the `subTrajectory` parameter SHALL be part of the result set, if the `subTrajectory` parameter is provided.
B	If the `subTrajectory` parameter is provided, the endpoint SHALL return only a subset of the trajectory derived from temporal primitive geometry by the *subTrajectory* query at time (new start time and new end time) included in the `subTrajectory` parameter, using interpolated trajectory according to the `interpolation` property in TemporalPrimitiveGeometry.
C	If the `subTrajectory` parameter is provided, the `interpolation` property in the response SHALL be the same as the temporal primitive geometry's `interpolation` property value.
D	Apply `subTrajectory` only to moving features that intersect a `bbox` or (and) a `datetime` parameter, if the `subTrajectory` parameter is provided with a `bbox` or (and) a `datetime` parameter.

Requirement 14	/req/movingfeatures/param-subtrajectory-response
E	The subTrajectory parameter SHALL match all temporal geometry objects in the moving feature object in the selected collection, i.e., MovingFeatures resource.

The subTrajectory query parameter is used to select a subset of TemporalGeometrySequence of each MovingFeature in the MovingFeatures, for the specified time interval. The two date-time strings (new start time and new end time) in the time interval adhere to IETF RFC3339. The new start time(*t_s*) is earlier than new end time(*t_e*), i.e., *t_s* < *t_e*. Example 1 shows valid expression examples of the subTrajectory parameter.

Example 1. subTrajectory parameter valid (and invalid) Examples

(O) "2018-02-12T23:20:50Z","2018-02-12T23:30:50Z"

(X) "2018-02-12T23:20:50Z"

(X) "2018-02-12T23:20:50Z","2018-02-12T23:30:50Z","2018-02-12T23:40:50Z"

(X) "2018-02-12T23:20:50Z","2018-02-12T23:20:50Z"

(X) "2018-02-12T23:20:50Z","2018-02-12T22:40:50Z"

If the subTrajectory parameter is provided by the client, the endpoint SHALL return only a subset of the trajectory derived from temporal primitive geometry by the *subTrajectory* operation at time (new start time and new end time) included in the subTrajectory parameter, using interpolated trajectory according to the interpolation property in TemporalPrimitiveGeometry. Also, the interpolation property in the response shall be the same as the original temporal primitive geometry. Note that the *subTrajectory* operation is defined in the OGC Moving Feature Access Standard.

Figure 2. Example of a response result with subTrajectory parameter

Create

The create operation is defined in the CREATE conformance class of OGC API Ñ Features. This operation targeted MovingFeature resource.

1. Issue a `POST` request on `{root}/collections/{collectionID}/items` path

Support for the HTTP POST method is specified as a requirement in OGC API Ñ Features.

Requirement 15	/req/movingfeatures/features-post
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `CREATE` operation requirement `/req/create-replace-delete/insert-post-op`.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `CREATE` request body requirements `/req/create-replace-delete/insert-body` and `/req/create-replace-delete/insert-content-type`.
C	The content of the request body SHALL be based upon the *MovingFeature* object and *MovingFeatureCollection* object in OGC Moving Features JSON encoding standard schema.

The following example adds a new feature (*MovingFeature* object in MF-JSON) to the specific Collection. The feature is represented as *MovingFeature* object (or *MovingFeatureCollection* object) in MF-JSON. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Creating a New MovingFeature Object:

```
Ê  Client                                                                Server
Ê    |                                                                     |
Ê    | POST /collections/mfc_1/items   HTTP/1.1                            |
Ê    | Content-Type: application/geo+json                                  |
Ê    |                                                                     |
Ê    | {                                                                   |
Ê    |   "type": "Feature",                                                |
Ê    |   "id": "mf_1",                                                     |
Ê    |   "properties": {                                                   |
Ê    |      "name": "car1",                                                |
Ê    |      "state": "test1",                                              |
Ê    |      "video": "http://.../example/video.mpeg"                       |
Ê    |   },                                                                |
Ê    |   "crs": {                                                          |
Ê    |      "type": "Name",                                                |
Ê    |      "properties": {                                                |
Ê    |         "name": "urn:ogc:def:crs:OGC:1.3:CRS84"                     |
Ê    |      }                                                              |
Ê    |   },                                                                |
Ê    |   "trs": {                                                          |
Ê    |      "type": "Link",                                                |
Ê    |      "properties": {                                                |
Ê    |         "type": "ogcdef",                                           |
Ê    |         "href": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"  |
Ê    |      }                                                              |
Ê    |   },                                                                |
```

```
Ê    |   "temporalGeometry": {                                                                  |
Ê    |      "type": "MovingPoint",                                                              |
Ê    |      "datetimes": [                                                                      |
Ê    |         "2011-07-14T22:01:01.000Z",                                                      |
Ê    |         "2011-07-14T22:01:02.000Z",                                                      |
Ê    |         "2011-07-14T22:01:03.000Z",                                                      |
Ê    |         "2011-07-14T22:01:04.000Z",                                                      |
Ê    |         "2011-07-14T22:01:05.000Z"                                                       |
Ê    |      ],                                                                                  |
Ê    |      "coordinates": [                                                                    |
Ê    |         [139.757083,35.627701,0.5],                                                      |
Ê    |         [139.757399,35.627701,2.0],                                                      |
Ê    |         [139.757555,35.627688,4.0],                                                      |
Ê    |         [139.757651,35.627596,4.0],                                                      |
Ê    |         [139.757716,35.627483,4.0]                                                       |
Ê    |      ],                                                                                  |
Ê    |      "interpolation": "Linear",                                                          |
Ê    |      "base": {                                                                           |
Ê    |         "type": "glTF",                                                                  |
Ê    |         "href": "http://.../example/car3dmodel.gltf"                                     |
Ê    |      },                                                                                  |
Ê    |      "orientations": [                                                                   |
Ê    |          {"scales": [1,1,1],"angles": [0,0,0]},                                          |
Ê    |          {"scales": [1,1,1],"angles": [0,355,0]},                                        |
Ê    |          {"scales": [1,1,1],"angles": [0,0,330]},                                        |
Ê    |          {"scales": [1,1,1],"angles": [0,0,300]},                                        |
Ê    |          {"scales": [1,1,1],"angles": [0,0,270]},                                        |
Ê    |      ]                                                                                   |
Ê    |   },                                                                                     |
Ê    |   "temporalProperties": [                                                                |
Ê    |      {                                                                                   |
Ê    |         "datetimes": [                                                                   |
Ê    |            "2011-07-14T22:01:01.450Z",                                                   |
Ê    |            "2011-07-14T23:01:01.450Z",                                                   |
Ê    |            "2011-07-15T00:01:01.450Z"                                                    |
Ê    |            ],                                                                            |
Ê    |         "length": {                                                                      |
Ê    |            "type": "Measure",                                                            |
Ê    |            "form": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length",                 |
Ê    |            "values": [1,2.4,1],                                                          |
Ê    |            "interpolation": "Linear",                                                    |
Ê    |            "description": "description1"                                                 |
Ê    |         },                                                                               |
Ê    |         "discharge": {                                                                   |
Ê    |            "type": "Measure",                                                            |
Ê    |            "form": "MQS",                                                                |
Ê    |            "values": [3,4,5],                                                            |
Ê    |            "interpolation": "Step"                                                       |
Ê    |         }                                                                                |
Ê    |      },                                                                                  |
Ê    |      {                                                                                   |
```

```
Ê    |        "datetimes": [                                                  |
Ê    |           "2011-07-15T23:01:01.450Z",                                  |
Ê    |           "2011-07-16T00:01:01.450Z"                                   |
Ê    |        ],                                                              |
Ê    |        "camera": {                                                     |
Ê    |           "type": "Image",                                             |
Ê    |           "values": [                                                  |
Ê    |              "http://.../example/image1",                              |
Ê    |              "VBORw0KGgoAAAANSUhEU......"                              |
Ê    |           ],                                                           |
Ê    |           "interpolation": "Discrete"                                  |
Ê    |        },                                                              |
Ê    |        "labels": {                                                     |
Ê    |           "type": "Text",                                              |
Ê    |           "values": ["car","human"],                                   |
Ê    |           "interpolation": "Discrete"                                  |
Ê    |        }                                                               |
Ê    |     }                                                                  |
Ê    |   ]                                                                    |
Ê    | }                                                                      |
Ê    |------------------------------------------------------------------------>|
Ê    |                                                                        |
Ê    | HTTP/1.1 201 Created                                                   |
Ê    | Location: /collections/mfc_1/items/mf_1                                |
Ê    |<------------------------------------------------------------------------|
```

8.3.3. Response

Retrieve

A successful response to the MovingFeatures GET operation is a document that contains the static data for a set of moving features. If the value of the `subTrajectory` query parameter is provided, the value of the corresponding `temporalGeometry` property of each moving feature is calculated using the `subTrajectory` parameter value and included in the result, i.e., a *MovingFeatureCollection* object of MF-JSON. In a typical deployment of the OGF API Ñ MF, the MovingFeatures GET response will list `MovingFeature` resource.

Requirement 16	/req/movingfeatures/features-get-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Features `Features` response requirement `/req/core/fc-response`.
B	The response SHALL only include moving features selected by the request with parameters.
C	Each moving feature in the response SHALL include the mandatory properties listed in Table 8.

MovingFeatures GET Response Schema (movingFeatureCollection.yaml):

```
type: object
required:
Ê - type
Ê - features
properties:
Ê type:
Ê   type: string
Ê   enum:
Ê     - 'FeatureCollection'
Ê features:
Ê   type: array
Ê   nullable: true
Ê   items:
Ê     $ref: 'movingFeature.yaml'
Ê crs:
Ê   $ref: 'MF-JSON/Prism/crs.yaml'
Ê trs:
Ê   $ref: 'MF-JSON/Prism/trs.yaml'
Ê bbox:
Ê   $ref: 'MF-JSON/Prism/bbox.yaml'
Ê time:
Ê   $ref: 'MF-JSON/Prism/lifeSpan.yaml'
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
Ê timeStamp:
Ê   type: string
Ê   format: date-time
Ê numberMatched:
Ê   type: integer
Ê   minimum: 0
Ê numberReturned:
Ê   type: integer
Ê   minimum: 0
```

The following JSON payload is an example of a response to an OGC API Ñ Moving Features MovingFeatures GET operation.

An Example of a MovingFeatures GET Operation:

```
{
Ê "type": "FeatureCollection",
Ê "features":[
Ê   {
Ê     "id": "mf-1",
Ê     "type": "Feature",
```

```
Ê    "geometry":{
Ê      "type": "LineString",
Ê      "coordinates": [
Ê        [139.757083, 35.627701, 0.5],
Ê        [139.757399, 35.627701, 2.0],
Ê        [139.757555, 35.627688, 4.0],
Ê        [139.757651, 35.627596, 4.0],
Ê        [139.757716, 35.627483, 4.0]
Ê      ]
Ê    },
Ê    "properties":{
Ê      "label": "car",
Ê      "state": "test1",
Ê      "video":
"http://www.opengis.net/spec/movingfeatures/json/1.0/prism/example/video.mpeg"
Ê    },
Ê    "bbox":[
Ê      139.757083, 35.627483, 0.0,
Ê      139.757716, 35.627701, 4.5
Ê    ],
Ê    "time":[
Ê      "2011-07-14T22:01:01Z",
Ê      "2011-07-15T01:11:22Z"
Ê    ],
Ê    "crs": {
Ê      "type": "Name",
Ê      "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
Ê    },
Ê    "trs": {
Ê      "type": "Name",
Ê      "properties": "urn:ogc:data:time:iso8601"
Ê    }
Ê  }
Ê ],
Ê "crs": {
Ê   "type": "Name",
Ê   "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
Ê },
Ê "trs": {
Ê   "type": "Name",
Ê   "properties": "urn:ogc:data:time:iso8601"
Ê },
Ê "links":[
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items",
Ê     "rel": "self",
Ê     "type": "application/geo+json"
Ê   },
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items&offset=1&limit=1",
Ê     "rel": "next",
```

```
Ê     "type": "application/geo+json"
Ê   }
Ê ],
Ê "timeStamp": "2020-01-01T12:00:00Z",
Ê "numberMatched": 100,
Ê "numberReturned": 1
}
```


Create

A successful response to the MovingFeatures POST operation is an HTTP status code.

Requirement 17	/req/movingfeatures/features-post-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.3.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

8.4. Resource MovingFeature

8.4.1. Overview

A MovingFeature object consists of the set of static information that describes a single moving feature and the set of temporal object, such as temporal geometry and temporal properties. An abbreviated copy of this information is returned for each MovingFeature in the {root}/collections/{collectionId}/items GET response.

The schema for the moving feature object presented in this clause is an extension of the GeoJSON Feature Object defined in GeoJSON. Table 8 defines the set of properties that may be used to describe a moving feature.

Table 8. Table of the properties related to the moving feature

Property	Requirement	Description
id	M	A unique identifier to the moving feature.
type	M	The GeoJSON feature type (i.e., one of 'Feature' or 'FeatureCollection').
geometry	M	Projective geometry of the moving feature.
properties	M	A set of properties of GeoJSON.
bbox	O	Bounding box information for the moving feature.

Property	Requirement	Description
time	O	Life span information for the moving feature.
crs	O	Coordinate reference system (CRS) information for the moving feature.
trs	O	Temporal reference system information for the moving feature.
temporalGeometry	O	A sequence of TemporalPrimitiveGeometry for the moving feature.
temporalProperties	O	A set of TemporalProperty of the moving feature.

NOTE The properties *id*, *type*, *geometry*, *properties*, and *bbox* were inherited from GeoJSON. NOTE: The properties

Requirement 18	/req/movingfeatures/mf-mandatory
A	A moving feature object SHALL contain all the mandatory properties listed in Table 8.

8.4.2. Operation

Retrieve

The retrieve operation is defined in the Feature conformance class of OGC API Ñ Features. No modifications are needed to support MovingFeature resources.

1. Issue a GET request on the {root}/collections/{collectionId}/items/{mFeatureId} path

The {mFeatureId} path parameter is the unique identifier for a single moving feature offered by the API. The list of valid values for {mFeatureId} is provided in the {root}/collections/{collectionId}/items GET response.

Support for GET on the {root}/collections/{collectionID}/items/{mFeatureId} path is required by OGC API Ñ Features.

Requirement 19	/req/movingfeatures/mf-get
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Features Feature operation requirement /req/core/f-op.
B	For every moving feature in a moving feature collection (path {root}/collections/{collectionId}), the server SHALL support the HTTP GET operation at the path {root}/collections/{collectionId}/items/{mFeatureId}

Requirement 19	/req/movingfeatures/mf-get
C	The path parameter `collectionId` is each `id` property in the Collection GET operation response where the value of the `itemType` property is specified as MovingFeature. The path parameter `mFeatureId` is an `id` property of the moving feature.

Delete

The delete operation is defined in the `DELETE` conformance class of OGC API Ñ Features.

1. Issue a `DELETE` request on `{root}/collections/{collectionId}/items/{mFeatureId}` path

Support for the HTTP DELETE method is required by OGC API Ñ Features.

Requirement 20	/req/movingfeatures/mf-delete
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `DELETE` operation requirement `req/create-replace-delete/delete/delete-op`.
B	For every moving feature in a moving feature collection (path `{root}/collections/{collectionId}`), the server SHALL support the HTTP DELETE operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}`
C	The path parameter `collectionId` is each `id` property in the Collection GET operation response where the value of the `itemType` property is specified as MovingFeature. The path parameter `mFeatureId` is an `id` property of the moving feature.

8.4.3. Response

Retrieve

A successful response to the MovingFeature GET operation is a set of metadata that describes the moving feature identified by the `{mFeatureId}` parameter. This response does not include a set of temporal object information. The temporal object information may be accessed using TemporalGeometry and TemporalProperties operations.

Requirement 21	/req/movingfeatures/mf-get-success
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200`.
B	The content of that response SHALL include the set of moving feature's metadata that defined in the response schema.

MovingFeature GET Response Schema (movingFeature.yaml):

```
type: object
required:
Ê - id
```

```
Ê - type
properties:
Ê type:
Ê   type: string
Ê   enum:
Ê     - 'Feature'
Ê temporalGeometry:
Ê   $ref: 'MF-JSON/Prism/temporalGeometry.yaml'
Ê temporalProperties:
Ê   $ref: 'MF-JSON/Prism/temporalProperties.yaml'
Ê crs:
Ê   $ref: 'MF-JSON/Prism/crs.yaml'
Ê trs:
Ê   $ref: 'MF-JSON/Prism/trs.yaml'
Ê bbox:
Ê   $ref: 'MF-JSON/Prism/bbox.yaml'
Ê time:
Ê   $ref: 'MF-JSON/Prism/lifeSpan.yaml'
Ê geometry:
Ê   $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/geometryGeoJSON
.yaml'
Ê properties:
Ê   type: object
Ê   nullable: true
Ê id:
Ê   description: 'An identifier for the feature'
Ê   oneOf:
Ê     - type: string
Ê     - type: integer
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
```

The `time` property of the MovingFeature response represents a particular period of moving feature existence.

The following JSON payload is an example of a response to an OGC API Ñ Moving Features MovingFeature operation.

An Example of a MovingFeature JSON Payload:

```
{
Ê "id": "mf-1",
Ê "type": "Feature",
Ê "geometry":{
Ê   "type": "LineString",
Ê   "coordinates": [
```

```
Ê     [139.757083, 35.627701, 0.5],
Ê     [139.757399, 35.627701, 2.0],
Ê     [139.757555, 35.627688, 4.0],
Ê     [139.757651, 35.627596, 4.0],
Ê     [139.757716, 35.627483, 4.0]
Ê   ]
Ê },
Ê "properties":{
Ê   "name": "car1",
Ê   "state": "test1",
Ê   "video":
"http://www.opengis.net/spec/movingfeatures/json/1.0/prism/example/video.mpeg"
Ê },
Ê "bbox":[
Ê   139.757083, 35.627483, 0.0,
Ê   139.757716, 35.627701, 4.5
Ê ],
Ê "time":[
Ê   "2011-07-14T22:01:01Z",
Ê   "2011-07-15T01:11:22Z"
Ê ],
Ê "crs": {
Ê   "type": "Name",
Ê   "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
Ê },
Ê "trs": {
Ê   "type": "Name",
Ê   "properties": "urn:ogc:data:time:iso8601"
Ê }
}
```


Delete

A successful response to the Collection DELETE operation is an HTTP status code.

Requirement 22	/req/movingfeatures/mf-delete-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Features DELETE response requirement /req/create-replace-delete/delete/response.
B	If no resource with the identifier exists in the collection, the server SHALL respond with a not-found exception (404).

8.4.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

8.5. Resource TemporalGeometrySequence

8.5.1. Overview

The TemporalGeometrySequence resource supports retrieving and creating operations via GET and POST HTTP methods respectively.

1. A retrieve operation returns a set of TemporalPrimitiveGeometry object which is included in the MovingFeature that specified by {mFeatureId}. The set of TemporalPrimitiveGeometry object returned to the response can be limited using the limit, bbox, datetime, and leaf parameters.
2. A create operation posts a new TemporalPrimitiveGeometry resource to the MovingFeature that specified by {mFeatureId}.

8.5.2. Query Parameters

Parameter leaf

The leaf query parameter is defined as follows:

Requirement 23	/req/movingfeatures/param-leaf-definition

A	The operation SHALL support a query parameter leaf with the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: leaf`
	`in: query`
	`required: false`
	`schema:`
	`Ê type: array`
	`Ê uniqueItems: true`
	`Ê minItems: 1`
	`Ê items:`
	`Ê  type: string`
	`Ê  format: date-time`
	`style: form`
	`explode: false`

B	The leaf parameter SHALL be a sequence of monotonic increasing instants with date-time strings.
C	The syntax of date-time is specified by RFC 3339, 5.6.

Requirement 24	/req/movingfeatures/param-leaf-response
A	If the `leaf` parameter is provided by the client and supported by the server, then only resources that have a temporal information (i.e., `datetimes` property) that intersects the temporal information in the `leaf` parameter SHALL be part of the result set.
B	The `leaf` parameter SHALL match all resources in the moving feature that are associated with temporal information.
C	If the `leaf` parameter is provided by the client and supported by the server, the endpoint SHALL return only temporal geometry coordinate (or temporal property value) with the *pointAtTime* query at each time included in the `leaf` parameter, using interpolated trajectory according to the `interpolation` property.
D	If the `leaf` parameter is provided by the client and supported by the server, the `interpolation` property in the response SHALL be 'Discrete'.

The `leaf` parameter is a sequence of monotonic increasing instants represented by date-time strings (ex. "2018-02-12T23:20:50Z") whose structure adheres to IETF RFC3339. The `leaf` parameter consists of a list of the date-time format strings, different from `datetime` parameter. The list does not allow the same element value. Example 2 shows valid expression examples of the `leaf` parameter.

Example 2. leaf parameter valid (and invalid) Examples

(O) "2018-02-12T23:20:50Z"

(O) "2018-02-12T23:20:50Z","2018-02-12T23:30:50Z"

(O) "2018-02-12T23:20:50Z","2018-02-12T23:30:50Z","2018-02-12T23:40:50Z"

(X) "2018-02-12T23:20:50Z","2018-02-12T23:20:50Z"

(X) "2018-02-12T23:20:50Z","2018-02-12T22:40:50Z"

If the `leaf` parameter is provided by the client, the endpoint returns only temporal geometry coordinate (or temporal property value) with the leaf query at each time included in the `leaf` parameter, similar to *pointAtTime* operation in the OGC Moving Feature Access Standard. And `interpolation` property in the response shall be 'Discrete'.

Figure 3. Example of a response result with `leaf` parameter

8.5.3. Operation

Retrieve

1. Issue a `GET` request on the `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` path

Requirement 25	/req/movingfeatures/tgsequence-get
A	For every moving feature identified in the MovingFeatures GET response (path `{root}/collections/{collectionId}/items`), the server SHALL support the HTTP GET operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence`
B	The path parameter `collectionId` is each `id` property in the Collection GET response where the value of the `itemType` property is specified as MovingFeature. The path parameter `mFeatureId` is each `id` property in the MovingFeatures GET response.

Create

The create operation is defined in the `CREATE` conformance class of OGC API Ñ Features. This operation targeted TemporalPrimitiveGeometry resource.

1. Issue a `POST` request on `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` path

Support for the HTTP POST method is specified as a requirement in OGC API Ñ Features.

Requirement 26	/req/movingfeatures/tgsequence-post
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `CREATE` operation requirement `/req/create-replace-delete/insert-post-op`.

Requirement 26	/req/movingfeatures/tgsequence-post
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE request body requirements /req/create-replace-delete/insert-body and /req/create-replace-delete/insert-content-type.
C	The content of the request body SHALL be based upon the *TemporalPrimitiveGeometry* object in OGC Moving Features JSON encoding standard schema.
D	The ending date-time instance (*t_end*) in the temporal geometry object in MovingFeature, determined by mFeatureId, SHALL be earlier than the beginning date-time instance (*t_new*) in the temporal geometry object in the request body, i.e., *t_end* < *t_new*.

The following example adds a new feature (*TemporalPrimitiveGeometry* object in MF-JSON) to the feature created by the Creation a MovingFeature Example. The feature is represented as *TemporalPrimitiveGeometry* object in MF-JSON, which is an extension of the JSON. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Creating a New TemporalPrimitiveGeometry Object:

```
Ê  Client                                                                Server
Ê    |                                                                     |
Ê    | POST /collections/mfc_1/items/mf_1/tgsequence  HTTP/1.1             |
Ê    | Content-Type: application/json                                      |
Ê    |                                                                     |
Ê    | {                                                                   |
Ê    |    "id": "tg_1",                                                    |
Ê    |    "type": "MovingPoint",                                           |
Ê    |    "datetimes": [                                                   |
Ê    |       "2011-07-14T22:01:06.000Z",                                   |
Ê    |       "2011-07-14T22:01:07.000Z",                                   |
Ê    |       "2011-07-14T22:01:08.000Z"                                    |
Ê    |    ],                                                               |
Ê    |    "coordinates": [                                                 |
Ê    |       [139.757716,35.627483,4.0],                                   |
Ê    |       [139.757782,35.627483,4.0],                                   |
Ê    |       [139.757843,35.627483,4.0]                                    |
Ê    |    ],                                                               |
Ê    |    "interpolation": "Linear",                                       |
Ê    |    "base": {                                                        |
Ê    |       "type": "glTF",                                               |
Ê    |       "href": "http://.../example/car3dmodel.gltf"                  |
Ê    |    },                                                               |
Ê    |    "orientations": [                                                |
Ê    |        {"scales": [1,1,1],"angles": [0,0,270]},                     |
Ê    |        {"scales": [1,1,1],"angles": [0,0,270]},                     |
Ê    |        {"scales": [1,1,1],"angles": [0,0,270]}                      |
Ê    |    ]                                                                |
```

```
Ê    | }                                                                          |
Ê    |--------------------------------------------------------------------------->|
Ê    |                                                                            |
Ê    | HTTP/1.1 201 Created                                                       |
Ê    | Location: /collections/mfc_1/items/mf_1/tgsequence/tg_1                    |
Ê    |<---------------------------------------------------------------------------|
```

8.5.4. Response

Retrieve

A successful response to the TemporalGeometrySequence GET operation is a document that contains the set of temporal geometry of the moving feature identified by the {mFeatureId} parameter.

Requirement 27	/req/movingfeatures/tgsequence-get-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature Features response requirement /req/core/fc-response.
B	The response SHALL be based upon the TemporalComplexGeometry object that only includes temporal primitive geometry selected by request with limit, bbox, datetime, and leaf parameters.
C	Each temporal primitive geometry in the response SHALL include the mandatory properties listed in Table 9.

TemporalGeometrySequence GET Response Schema (temporalGeometrySequence.yaml):

```
type: object
required:
Ê - type
Ê - prisms
properties:
Ê type:
Ê   type: string
Ê   default: 'MovingGeometryCollection'
Ê prisms:
Ê   type: array
Ê   items:
Ê     $ref: 'temporalGeometry.yaml'
Ê crs:
Ê   $ref: crs.yaml
Ê trs:
Ê   $ref: trs.yaml
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
```

```
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
Ê timeStamp:
Ê   type: string
Ê   format: date-time
Ê numberMatched:
Ê   type: integer
Ê   minimum: 0
Ê numberReturned:
Ê   type: integer
Ê   minimum: 0
```

TemporalPrimitiveGeometry Schema (temporalPrimitiveGeometry.yaml):

```
type: object
required:
Ê - id
Ê - type
Ê - coordinates
Ê - datetimes
Ê - interpolation
properties:
Ê id:
Ê   type: string
Ê type:
Ê   type: string
Ê   enum:
Ê     - "MovingPoint"
Ê     - "MovingLineString"
Ê     - "MovingPolygon"
Ê     - "MovingPointCloud"
Ê coordinates:
Ê   type: array
Ê   minItems: 2
Ê   items:
Ê     oneOf:
Ê       - $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/pointGeoJSON.ya
ml#coordinates'
Ê       - $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/linestringGeoJS
ON.yaml#coordinates'
Ê       - $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/polygonGeoJSON.
yaml#coordinates'
Ê       - $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/multipointGeoJS
ON.yaml#coordinates'
Ê datetimes:
Ê   type: array
Ê   uniqueItems: true
```

```
Ê   minItems: 2
Ê   items:
Ê     type: string
Ê     format: date-time
Ê interpolation:
Ê   type: string
Ê   enum:
Ê     - 'Discrete'
Ê     - 'Step'
Ê     - 'Linear'
Ê     - 'Quadratic'
Ê     - 'Cube'
Ê base:
Ê   type: object
Ê   nullable: true
Ê   required:
Ê     - href
Ê     - type
Ê   properties:
Ê     href:
Ê       type: string
Ê       format: 'uri'
Ê     type:
Ê       type: string
Ê orientations:
Ê   type: array
Ê   minItems: 2
Ê   items:
Ê     type: object
Ê     required:
Ê       - scales
Ê       - angles
Ê     properties:
Ê       scales:
Ê         type: array
Ê         oneOf:
Ê           - minItems: 2
Ê             maxItems: 2
Ê           - minItems: 3
Ê             maxItems: 3
Ê         items:
Ê           type: number
Ê       angles:
Ê         type: array
Ê         oneOf:
Ê           - minItems: 2
Ê             maxItems: 2
Ê           - minItems: 3
Ê             maxItems: 3
Ê         items:
```

```
Ê          type: number
```

The following JSON payload is an example of a response to an OGC API Ñ Moving Features TemporalGeometrySequence GET operation.

An Example of a TemporalGeometrySequence GET operation:

```
{
Ê "type": "MovingGeometryCollection",
Ê "prisms": [
Ê   {
Ê     "id": "tg-1",
Ê     "type": "MovingPoint",
Ê     "datetimes": [
Ê       "2011-07-14T22:01:02Z",
Ê       "2011-07-14T22:01:03Z",
Ê       "2011-07-14T22:01:04Z"
Ê     ],
Ê     "coordinates": [
Ê       [139.757399, 35.627701, 2.0],
Ê       [139.757555, 35.627688, 4.0],
Ê       [139.757651, 35.627596, 4.0]
Ê     ],
Ê     "interpolation": "Linear",
Ê     "base": {
Ê       "type": "glTF",
Ê       "href":
"https://www.opengis.net/spec/movingfeatures/json/1.0/prism/example/car3dmodel.gltf"
Ê     },
Ê     "orientations":[
Ê       {
Ê         "scales": [1,1,1],
Ê         "angles": [0,355,0]
Ê       },
Ê       {
Ê         "scales": [1,1,1],
Ê         "angles": [0,0,330]
Ê       },
Ê       {
Ê         "scales": [1,1,1],
Ê         "angles": [0,0,300]
Ê       }
Ê     ]
Ê   }
Ê ],
Ê "crs": {
Ê   "type": "Name",
Ê   "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
Ê },
Ê "trs": {
```

```
Ê   "type": "Name",
Ê   "properties": "urn:ogc:data:time:iso8601"
Ê },
Ê "links": [
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-1/tgeometries",
Ê     "rel": "self",
Ê     "type": "application/json"
Ê   },
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-
1/tgeometries&offset=10&limit=1",
Ê     "rel": "next",
Ê     "type": "application/json"
Ê   }
Ê ],
Ê "timeStamp": "2021-09-01T12:00:00Z",
Ê "numberMatched": 100,
Ê "numberReturned": 1
}
```


Create

A successful response to the TemporalGeometrySequence POST operation is an HTTP status code.

Requirement 28	/req/movingfeatures/tgsequence-post-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.5.5. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

8.6. Resource TemporalPrimitiveGeometry

8.6.1. Overview

A TemporalPrimitiveGeometry object represents the movement of a moving feature with various types of moving geometry, i.e., MovingPoint, MovingLineString, MovingPolygon, and MovingPointCloud. It can also represent the movement of a 3D object with its orientation.

The schema for the TemporalPrimitiveGeometry presented in this clause is an extension of the TemporalPrimitiveGeometry Object defined in MF-JSON standard. Table 9 defines the set of properties that may be used to describe a TemporalPrimitiveGeometry.

Table 9. Table of the properties related to the TemporalPrimitiveGeometry

Property	Requirement	Description
id	M	A unique identifier to the temporal geometry.
type	M	A primitive geometry type of MF-JSON (i.e., one of 'MovingPoint', 'MovingLineString', 'MovingPolygon', or 'MovingPointCloud').
datetimes	M	A sequence of monotonically increasing instants.
coordinates	M	A sequence of leaf geometries of a temporal geometry, having the same number of elements as "datetimes".
interpolation	M	A predefined type of motion curve (i.e., one of 'Discrete', 'Step', 'Linear', 'Quadratic' or 'Cubic').
base	O	*type*: A type of 3D file format, such as 'STL', 'OBJ', 'PLY', and 'glTF'.
		href: A URL to address a 3D model data which represents a base geometry of a 3D shape.
orientations	O	*scales*: An array value of numbers along the x, y, and z axis in order as three scale factors.
		angles: An array value of numbers along the x, y, and z axis in order as Euler angles in degree.

NOTE The detailed information and requirements for each property are described in the OGC Moving Feature JSON encoding standard.

Requirement 29	/req/movingfeatures/tpgeometry-mandatory
A	A TemporalPrimitiveGeometry object SHALL contain all the mandatory properties listed in Table 9.

8.6.2. Operation

Delete

The delete operation is defined in the `DELETE` conformance class of API Ñ Features.

1. Issue a `DELETE` request on `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}` path

The {tGeometryId} parameter is the unique identifier for a TemporalPrimitiveGeometry object offered by the API. The list of valid values for `{tGeometryId}` is provided in the `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` GET response.

Support for the HTTP DELETE method is specified as a requirement in OGC API Ñ Features.

Requirement 30	/req/movingfeatures/tpgeometry-delete
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `DELETE` operation requirement `/req/create-replace-delete/delete/delete-op`.
B	For every temporal geometry in a moving feature (path `{root}/collections/{collectionId}/items/{mFeatureId}`), the server SHALL support the HTTP DELETE operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}`
C	The path parameter `collectionId` is each `id` property in the Collection GET operation response where the value of the `itemType` property is specified as MovingFeature. The path parameter `mFeatureId` is an `id` property of the moving feature. The path parameter `tGeometryId` is an `id` property of the temporal geometry.

8.6.3. Response

Delete

A successful response to the TemporalPrimitiveGeometry DELETE operation is an HTTP status code.

Requirement 31	/req/movingfeatures/tpgeometry-delete-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature `DELETE` response requirement `/req/create-replace-delete/delete/response`.
B	If no resource with the identifier exists in the collection, the server SHALL respond with a not-found exception (404).

8.6.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

8.7. TemporalGeometry Query Resources

8.7.1. Overview

TemporalGeometry Query resources are spatiotemporal queries that support operations for accessing TemporalPrimitiveGeometry resources. The OGC API Ñ MF Standard identifies an initial set of common query types to implement. These are described in this clause. This list may change as the Standard is used and experience is gained.

Query resources related to the TemporalPrimitiveGeometry resource can be exposed using the path templates:

- {root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}

Where:

- {root} = Base URI for the API server
- {collectionId} = An identifier for a specific Collection of data
- {mFeatureId} = An identifier for a specific MovingFeature of a specific Collection of data
- {tGeometryId} = An identifier for a specific TemporalPrimitiveGeometry of a specific MovingFeature of data
- {quertyType} = An identifier for the query pattern performed by an implementation instance of the OGC API Ñ MF.

Table 10 provides a mapping of the initial query types proposed for the OGC API Ñ MF.

Table 10. Table of the query resources

Path Template	Query Type	Description
{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/distance	Distance	Return a graph of the time to distance function as a form of the TemporalProperty.
{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/velocity	Velocity	Return a graph of the time to velocity function as a form of the TemporalProperty.
{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/acceleration	Acceleration	Return a graph of the time to acceleration function as a form of the TemporalProperty.

8.7.2. Shared query parameters

Query parameters are used in URLs to define the resources which are returned on a GET request. The following are defined as standard shared parameters for use.

Parameter datetime

For datetime parameter, see Clause 10.1.3.

8.7.3. Distance Query

The Distance query returns a time to distance curve of the TemporalPrimitiveGeometry object as a form of the TemporalProperty. Figure 4 shows an example of time to distance curve.

Figure 4. Example of time to distance curve [OGC 16-120r3, OGC Moving Features Access]

The filter constraints are defined by the following query parameter:

Parameter datetime

The datetime parameter defines the specified date and time to return the distance value from the time to distance graph. When 'datetime' is not specified, an implementation instance (endpoint) of the API returns data from all available times of the specified TemporalPrimitiveGeometry resource.

8.7.4. Velocity Query

The Velocity query returns a time to velocity curve of the TemporalPrimitiveGeometry object as a form of the TemporalProperty.

The filter constraints are defined by the following query parameter:

Parameter datetime

The datetime parameter defines the specified date and time to return the velocity value from the time to velocity graph. When 'datetime' is not specified, an implementation instance (endpoint) of the API returns data from all available times of the specified TemporalPrimitiveGeometry resource.

8.7.5. Acceleration Query

The Acceleration query returns a time to acceleration curve of the TemporalPrimitiveGeometry object as a form of the TemporalProperty.

The filter constraints are defined by the following query parameter:

Parameter datetime

The datetime parameter defines the specified date and time to return the acceleration value from the time to acceleration graph. When 'datetime' is not specified, an implementation instance

(endpoint) of the API returns data from all available times of the specified TemporalPrimitiveGeometry resource.

8.7.6. Operation Requirements

Requirement 32	/req/movingfeatures/tpgeometry-query
A	For every TemporalPrimitiveGeometry identified in the TemporalGeometrySequence GET response (path `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence`), the server SHALL support the HTTP GET operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}`
B	The path parameter `collectionId` is each `id` property in the Collection GET operation response where the value of the `itemType` property is specified as MovingFeature. The path parameter `mFeatureId` is an `id` property of the moving feature. The path parameter `tGeometryId` is an `id` property of the temporal geometry.
C	The path parameter `queryType` SHALL be one of the predefined query type (distance, velocity, and acceleration)

Permission 1	/per/movingfeatures/tpgeometry-query
A	A distance query GET operation MAY include a `datetime` query parameter.
B	A velocity query GET operation MAY include a `datetime` query parameter.
C	An acceleration query GET operation MAY include a `datetime` query parameter.

8.7.7. Response Requirements

Requirement 33	/req/movingfeatures/tpgeometry-query-success
A	A successful execution of the distance, velocity, and acceleration query GET operation SHALL be reported as a response with an HTTP status code `200`.
B	The content of that response SHALL include the parametric value that is defined in the response schema.
C	The `type` property SHALL be TReal

8.8. Resource TemporalProperties

8.8.1. Overview

A TemporalProperties object consists of the set of TemporalProperty which is included in the MovingFeature that is specified by {mFeatureId}. The TemporalProperties resource supports the retrieve and create operations via the HTTP GET and POST methods respectively.

1. A retrieve operation returns a list of the available abbreviated copy of TemporalProperty object in the specified moving feature.
2. A create operation posts a new TemporalProperty object to the MovingFeature that is specified by {mFeatureId}.

8.8.2. Operation

Retrieve

1. Issue a GET request on the {root}/collections/{collectionId}/items/{mFeatureId}/tproperties path

Requirement 34	/req/movingfeatures/tproperties-get
A	For every moving feature identified in the MovingFeatures GET response (path {root}/collections/{collectionId}/items), the server SHALL support the HTTP GET operation at the path {root}/collections/{collectionId}/items/{mFeatureId}/tproperties
B	The path parameter collectionId is each id property in the Collection GET response where the value of the itemType property is specified as MovingFeature. The path parameter mFeatureId is each id property in the MovingFeatures GET response.

Create

The create operation is defined in the CREATE conformance class in the OGC API Ñ Features. This operation targeted TemporalProperty resource.

1. Issue a POST request on {root}/collections/{collectionId}/items/{mFeatureId}/tproperties path

Support for the HTTP POST method is specified as a requirement in OGC API Ñ Features.

Requirement 35	/req/movingfeatures/tproperties-post
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE request body requirements /req/create-replace-delete/insert-body and /req/create-replace-delete/insert-content-type.

Requirement 35	/req/movingfeatures/tproperties-post
C	The content of the request body SHALL be based upon the TemporalProperties or ParametricValues object schema.

TemporalProperties Request Body Schema (temporalProperty.yaml):

```
type: object
required:
Ê - name
Ê - type
properties:
Ê name:
Ê   type: string
Ê type:
Ê   type: string
Ê   enum:
Ê     - 'TBoolean'
Ê     - 'TText'
Ê     - 'TInteger'
Ê     - 'TReal'
Ê     - 'TImage'
Ê form:
Ê   oneOf:
Ê     - type: string
Ê       format: uri
Ê     - type: string
Ê       minLength: 3
Ê       maxLength: 3
Ê valueSequence:
Ê   type: array
Ê   uniqueItems: true
Ê   items:
Ê     $ref: 'temporalPrimitiveValue.yaml'
Ê description:
Ê   type: string
```

The following example adds a new feature (TemporalProperty Object and *ParametricValues* object in MF-JSON) to the feature created by the Creation a MovingFeature Example. The feature is represented as a JSON payload. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Creating a New TemporalProperty Object:

```
Ê  Client                                                                Server
Ê    |                                                                     |
Ê    | POST /collections/mfc_1/items/mf_1/tproperties   HTTP/1.1           |
Ê    | Content-Type: application/json                                      |
Ê    |                                                                     |
Ê    | {                                                                   |
```

```
Ê    |    "name": "speed",                                                    |
Ê    |    "type": "TReal",                                                    |
Ê    |    "form": "KMH"                                                       |
Ê    | }                                                                      |
Ê    |----------------------------------------------------------------------->|
Ê    |                                                                        |
Ê    | HTTP/1.1 201 Created                                                   |
Ê    | Location: /collections/mfc_1/items/mf_1/tproperties/speed              |
Ê    |<-----------------------------------------------------------------------|
```

An Example of Creating a New TemporalProperty Object with ParametricValues as a MF-JSON Encoding:

```
Ê  Client                                                                Server
Ê    |                                                                     |
Ê    | POST /collections/mfc_1/items/mf_1/tproperties   HTTP/1.1           |
Ê    | Content-Type: application/json                                      |
Ê    |                                                                     |
Ê    | {                                                                   |
Ê    |    "datetimes": [                                                   |
Ê    |       "2011-07-14T22:01:06.000Z",                                   |
Ê    |       "2011-07-14T22:01:07.000Z",                                   |
Ê    |       "2011-07-14T22:01:08.000Z",                                   |
Ê    |    ],                                                               |
Ê    |    "speed": {                                                       |
Ê    |       "type": "Measure",                                             |
Ê    |       "form": "KMH",                                                |
Ê    |       "values": [65.0, 70.0, 80.0],                                 |
Ê    |       "interpolation": "Linear"                                     |
Ê    |    }                                                                |
Ê    | }                                                                   |
Ê    |-------------------------------------------------------------------->|
Ê    |                                                                     |
Ê    | HTTP/1.1 201 Created                                                |
Ê    | Location: /collections/mfc_1/items/mf_1/tproperties/speed           |
Ê    |<--------------------------------------------------------------------|
```

8.8.3. Response

Retrieve

A successful response to the TemporalProperties GET operation is a document that contains the set of TemporalProperty of the moving feature identified by the {mFeatureId} parameter.

Requirement 36	/req/movingfeatures/tproperties-get-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature Features response requirement /req/core/fc-response.

Requirement 36	/req/movingfeatures/tproperties-get-success
B	Each temporal property object in the response SHALL include the mandatory properties listed in Table 11.

TemporalProperties GET Response Schema (TemporalProperties.yaml):

```
type: object
required:
Ê - temporalProperties
properties:
Ê temporalProperties:
Ê   type: array
Ê   items:
Ê     oneOf:
Ê       - $ref: 'temporalProperty.yaml'
Ê       - $ref: 'temporalPrimitiveValue.yaml'
Ê links:
Ê   type: array
Ê   items:
Ê     $ref:
'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/link.yaml'
Ê timeStamp:
Ê   type: string
Ê   format: date-time
Ê numberMatched:
Ê   type: integer
Ê   minimum: 0
Ê numberReturned:
Ê   type: integer
Ê   minimum: 0
```

The following JSON payload is an example of a response to an OGC API Ñ Moving Features TemporalProperties GET operation.

An Example of a TemporalProperties GET Operation:

```
{
Ê "temporalProperties": [
Ê   {
Ê     "name": "length",
Ê     "type": "TReal",
Ê     "form": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length"
Ê   },
Ê   {
Ê     "name": "speed",
Ê     "type": "TReal",
Ê     "form": "KHM"
Ê   }
Ê ],
Ê "links": [
```

```
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-1/tproperties",
Ê     "rel": "self",
Ê     "type": "application/json"
Ê   },
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-
1/tproperties&offset=2&limit=2",
Ê     "rel": "next",
Ê     "type": "application/json"
Ê   }
Ê ],
Ê "timeStamp": "2021-09-01T12:00:00Z",
Ê "numberMatched": 10,
Ê "numberReturned": 2
}
```


Create

A successful response to the TemporalProperties POST operation is an HTTP status code.

Requirement 37	/req/movingfeatures/tproperties-post-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.8.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

8.9. Resource TemporalProperty

8.9.1. Overview

The TemporalProperty resource supports the retrieve and create operations via the HTTP GET and POST methods respectively.

1. A retrieve operation returns a TemporalProperty resource which is included in the TemporalProperties that specified by {tPropertyName}.The TemporalProperty resource returned to the response can be limited using the limit, datetime, and leaf parameters.
2. A create operation posts a new temporal primitive value object to the TemporalProperties that specified by {tPropertyName}.

A temporal property object is a collection of dynamic non-spatial attributes and their temporal values with time. An abbreviated copy of this information is returned for each TemporalProperty in the {root}/collections/{collectionId}/items/{mFeatureId}/tproperties response.

The schema for the temporal property object presented in this clause is an extension of the `ParametricValues Object` defined in MF-JSON standard. Table 11 defines the set of property that may be used to describe a temporal property.

Table 11. Table of the properties related to the temporal property

Property	Requirement	Description
name	M	An identifier for the resource assigned by an external entity.
type	M	A predefined temporal property type (i.e., one of 'TBoolean', 'TText', 'TInteger', 'TReal', and 'TImage').
valueSequence	M	A sequence of temporal primitive value
form	O	A unit of measure.
description	O	A short description.

Table 12. Table of the properties related to the temporal primitive value

Property	Requirement	Description
datetimes	M	A sequence of monotonic increasing instants.
values	M	A sequence of dynamic values having the same number of elements as "datetimes".
interpolation	M	A predefined type for a dynamic value (i.e., one of 'Discrete', 'Step', 'Linear', or 'Regression').

NOTE The detailed information and requirements for each property are described in the OGC Moving Feature JSON Encoding Standard.

Requirement 38	/req/movingfeatures/tproperty-mandatory
A	A temporal property object SHALL contain all the mandatory properties listed in Table 11 and Table 12.

8.9.2. Operation

Retrieve

1. Issue a `GET` request on the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` path

The `{tPropertyName}` parameter is the unique identifier for a single temporal property value offered by an implementation instance (endpoint) of the OGC API Ñ MF. The list of valid values for `{tPropertyName}` is provided in the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` GET response.

Requirement 39	/req/movingfeatures/tproperty-get
A	For every temporal property in a moving feature (path {root}/collections/{collectionId}/items/{mFeatureId}/tproperties), the server SHALL support the HTTP GET operation at the path {root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertiesName}
B	The path parameter collectionId is each id property in the Collection GET response where the value of the itemType property is specified as MovingFeature. The path parameter mFeatureId is each id property in the MovingFeatures GET response. tPropertiesName is a local identifier of the temporal property.

Create

The create operation is defined in the CREATE conformance class in the OGC API Ñ Features. This operation targeted the new temporal primitive value object defined in Table 12.

1. Issue a POST request on {root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName} path

Support for the HTTP POST method is specified as a requirement in OGC API Ñ Features.

Requirement 40	/req/movingfeatures/tproperty-post
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE request body requirements /req/create-replace-delete/insert-body and /req/create-replace-delete/insert-content-type.
C	The content of the request body SHALL be based upon the TemporalPrimitiveValue schema.
D	The ending date-time instance (*t_end*) in the temporal value object in TemporalProperty, determined by tPropertyName, SHALL be earlier than the beginning date-time instance (*t_new*) in the temporal value object in the request body, i.e., *t_end* < *t_new*.

TemporalProperty Request Body Schema (TemporalPrimitiveValue.yaml):

```
type: object
required:
Ê - datetimes
Ê - values
Ê - interpolation
properties:
```

```
Ê datetimes:
Ê   type: array
Ê   uniqueItems: true
Ê   minItems: 2
Ê   items:
Ê     type: string
Ê     format: date-time
Ê values:
Ê   oneOf:
Ê     - type: number
Ê     - type: string
Ê     - type: boolean
Ê interpolation:
Ê   type: string
Ê   enum:
Ê     - 'Discrete'
Ê     - 'Step'
Ê     - 'Linear'
Ê     - 'Regression'
```

The following example adds a new feature (TemporalPrimitiveValue object) to the feature created by Creating a New TemporalPrimitiveValue Object Example. The feature is represented as a JSON payload. A pseudo-sequence diagram notation is used to illustrate the details of the HTTP communication between the client and the server.

An Example of Creating a New TemporalPrimitiveValue Object:

```
Ê  Client                                                                  Server
Ê    |                                                                       |
Ê    | POST /collections/mfc_1/items/mf_1/tproperties/speed   HTTP/1.1       |
Ê    | Content-Type: application/json                                        |
Ê    |                                                                       |
Ê    | {                                                                     |
Ê    |    "datetimes": [                                                     |
Ê    |       "2011-07-14T22:01:09.000Z",                                     |
Ê    |       "2011-07-14T22:01:010.000Z",                                    |
Ê    |    ],                                                                 |
Ê    |    "values": [                                                        |
Ê    |       90.0,                                                           |
Ê    |       95.0,                                                           |
Ê    |    ],                                                                 |
Ê    |    "interpolation": "Linear"                                          |
Ê    | }                                                                     |
Ê    |---------------------------------------------------------------------->|
Ê    |                                                                       |
Ê    | HTTP/1.1 201 Created                                                  |
Ê    | Location: /collections/mfc_1/items/mf_1/tproperties/speed             |
Ê    |<----------------------------------------------------------------------|
```

8.9.3. Response

Retrieve

A successful response to the TemporalProperty GET operation is a temporal property identified by the `{tPropertyName}` parameter.

Requirement 41	/req/movingfeatures/tproperty-get-success
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code 200.
B	The response SHALL only include temporal properties selected by the request with `limit`, `datetime`, and `leaf` parameters.
C	The content of that response SHALL include the parametric value that is defined in the response schema.

The following JSON payload is an example of a response to an OGC API Ñ Moving Features TemporalProperty GET operation.

An Example of TemporalProperty *GET Operation:*


```
{
Ê "temporalProperties": [
Ê   {
Ê     "datetimes":[
Ê       "2011-07-14T22:01:02Z",
Ê       "2011-07-14T22:01:03Z",
Ê       "2011-07-14T22:01:04Z"
Ê     ],
Ê     "values":[
Ê       65.0,
Ê       70.0,
Ê       80.0
Ê     ],
Ê     "interpolation": "Linear"
Ê   },
Ê   {
Ê     "datetimes":[
Ê       "2011-07-15T08:00:00Z",
Ê       "2011-07-15T08:00:01Z",
Ê       "2011-07-15T08:00:02Z"
Ê     ],
Ê     "values":[
Ê       0.0,
Ê       20.0,
Ê       50.0
Ê     ],
Ê     "interpolation": "Linear"
Ê   }
Ê ],
```

```
Ê "links": [
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-
1/tproperties/speed",
Ê     "rel": "self",
Ê     "type": "application/json"
Ê   },
Ê   {
Ê     "href": "https://data.example.org/collections/mfc-1/items/mf-
1/tproperties/speed&offset=2&limit=2",
Ê     "rel": "next",
Ê     "type": "application/json"
Ê   }
Ê ],
Ê "timeStamp": "2021-09-01T12:00:00Z",
Ê "numberMatched": 20,
Ê "numberReturned": 2
}
```


Create

A successful response to the TemporalProperty POST operation is an HTTP status code.

Requirement 42	/req/movingfeatures/tproperty-post-success
A	An implementation of the OGC API Ñ MF SHALL comply with the OGC API Ñ Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.9.4. Error situations

The requirements for handling unsuccessful requests are provided in the HTTP Response. General guidance on HTTP status codes and how they should be handled is provided in HTTP Status Codes.

Chapter 9. Common Requirements

9.1. Parameters

The query parameters bbox, datetime, and limit are inherited from OGC API Ñ Common. All requirements and recommendations in API Ñ Common regarding these parameters also apply to OGC API Ñ MF. No modifications are required.

9.1.1. Parameter limit

Requirement 43	/req/common/param-limit
A	An implementation instance of the OGC API Ñ MF SHALL support the Limit parameter for the operation.
B	Requests which include the Limit parameter SHALL comply with OGC API Ñ Common requirement /req/collections/rc-limit-definition.
C	Responses to Limit requests SHALL comply with OGC API Ñ Common requirements /req/collections/rc-limit-response

9.1.2. Parameter bbox

Requirement 44	/req/common/param-bbox
A	An implementation instance of the OGC API Ñ MF SHALL support the Bounding Box (bbox) parameter for the operation.
B	Requests which include the Bounding Box parameter SHALL comply with OGC API Ñ Common requirement /req/collections/rc-bbox-definition.
C	Responses to Bounding Box requests SHALL comply with OGC API Ñ Common requirement /req/collections/rc-bbox-response.

9.1.3. Parameter datetime

Requirement 45	/req/common/param-datetime
A	An implementation instance of the OGC API Ñ MF SHALL support the DateTime (datetime) parameter for the operation.
B	Requests which include the DateTime parameter SHALL comply with OGC API Ñ Common requirement /req/collections/rc-time-definition.
C	Responses to DateTime requests SHALL comply with OGC API Ñ Common requirement /req/collections/rc-time-response.

9.2. HTTP Response

Each HTTP request shall result in a response that meets the following requirement.

Requirement 46	/req/common/http-response
A	An HTTP operation SHALL return a response which includes a `status code` and an optional `description` element.
B	If the `status code` is not equal to 200, then the `description` element SHALL be populated.

The YAML schema for these results is provided in HTTP Response Schema.

An Example of the HTTP Response Schema:

```
title: Exception Schema
description: JSON schema for exceptions based on RFC 7807
type: object
required:
Ê - type
properties:
Ê type:
Ê   type: string
Ê title:
Ê   type: string
Ê status:
Ê   type: integer
Ê detail:
Ê   type: string
Ê instance:
Ê   type: string
```

9.3. HTTP Status Codes

Table 13 lists the main HTTP status codes that clients should be prepared to receive. This includes support for specific security schemes or URI redirection. In addition, other error situations may occur in the transport layer outside of the server.

Table 13. Typical HTTP status codes

Status code	Description
200	A successful request.
201	The server has been fulfilled the operation and a new resource has been created.
202	A successful request, but the response is still being generated. The response will include a `Retry-After` header field giving a recommendation in seconds for the client to retry.

Status code	Description
204	A successful request, but the resource has no data resulting from the request. No additional content or message body is provided.
304	An entity tag was provided in the request and the resource has not been changed since the previous request.
308	The server cannot process the data through a synchronous request. The response includes a `Location` header field which contains the URI of the location the result will be available at once the query is complete Asynchronous queries.
400	The server cannot or will not process the request due to an apparent client error. For example, a query parameter had an incorrect value.
401	The request requires user authentication. The response includes a `WWW-Authenticate` header field containing a challenge applicable to the requested resource.
403	The server understood the request, but is refusing to fulfill it. While status code `401` indicates missing or bad authentication, status code `403` indicates that authentication is not the issue, but the client is not authorized to perform the requested operation on the resource.
404	The requested resource does not exist on the server. For example, a path parameter had an incorrect value.
405	The request method is not supported. For example, a POST request was submitted, but the resource only supports GET requests.
406	Content negotiation failed. For example, the `Accept` header submitted in the request did not support any of the media types supported by the server for the requested resource.
413	Request entity too large. For example the query would involve returning more data than the server is capable of processing, the implementation should return a message explaining the query limits imposed by the server implementation.
500	An internal error occurred in the server.

Annex A: Conformance Class Abstract Test Suite (Normative)

A.1. Introduction

The Abstract Test Suite (ATS) is a compendium of test assertions applicable to implementations of the OGC API Ñ MF. An ATS provides a basis for developing an Executable Test Suite to verify that the implementation under test conforms to all the relevant functional specifications.

The abstract test cases (assertions) are organized into test groups that correspond to distinct conformance test classes defined in the OGC API Ñ MF Standard.

OGC APIs are not Web Services in the traditional sense. Rather, they define the behavior and content of a set of Resources exposed through a Web Application Programming Interface (Web API). Therefore, an API may expose resources in addition to those defined by the standard. A test engine shall traverse the API, identify and validate test points, and ignore resource paths which are not to be tested.

The Conformance Classes addressed by this Abstract Test Suite are the:

- MovingFeature Collection Catalog Conformance Class
- MovingFeature Conformance Class

A.2. Conformance Class MovingFeature Collection Catalog

Conformance Class	
http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/mf-collection	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
Dependency	http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html
Dependency	http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json
Dependency	http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections
Dependency	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson
Dependency	http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete

A.2.1. MovingFeature Collections

HTTP GET Operation

Abstract Test 1	/conf/mf-collection/collections-get
Requirement	/req/mf-collection/collections-get /req/mf-collection/collections-get-success
Test purpose	Validate that the MovingFeature Collections can be retrieved from the expected location.
Test method	1. Issue an HTTP GET request to the URL `{root}/collections` 2. Validate that a document was returned with a status code `200` 3. Validate the contents of the returned document using test `/conf/mf-collection/collections-get-success`

Abstract Test 2	/conf/mf-collection/collections-get-success
Requirement	/req/mf-collection/collections-get-success
Test purpose	Validate that the MovingFeature Collections complies with the required structure and contents.
Test method	1. Validate that all response documents comply with OGC API Ñ Common /conf/collections/rc-md-success 2. Validate the Collections resource for all supported media types using the resources and tests identified in Table 14 3. Verify that the response document contains a `itemType` property and its value is 'movingfeature'

The Collections content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 14. Schema and Tests for MovingFeature Collections content

Format	Schema Document	Test ID
HTML	collections.yaml	/conf/html/content
JSON	collections.yaml	/conf/json/content

HTTP POST Operation

Abstract Test 3	/conf/mf-collection/collections-post
Requirement	/req/mf-collection/collections-post /req/mf-collection/collections-post-success
Test purpose	Validate that the MovingFeature Collections can be created at the expected location.

Test method	1. Validate that the server complies with OGC API Ñ Features POST operation requirements 2. Validate that a body of a POST request using for all supported media types using the resources and tests identified in Table 15 3. Validate that the request body complies OGC API Ñ Features POST request body requirements 4. Issue an HTTP POST request to the URL {root}/collections 5. Validate the contents of the response using test /conf/mf-collection/collections-post-success

Table 15. Schema and Tests for Request Body of {root}/collections POST

Format	Schema Document	Test ID
HTML	collection_requestbody.yaml	/conf/html/content
JSON	collection_requestbody.yaml	/conf/json/content

Abstract Test 4	/conf/mf-collection/collections-post-success
Requirement	/req/mf-collection/collections-post-success
Test purpose	Validate that the response of {root}/collections POST request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 201 or 202 2. Validate that all response documents comply with OGC API Ñ Features POST response requirements

A.2.2. MovingFeature Collection

HTTP GET Operation

Abstract Test 5	/conf/mf-collection/collection-get
Requirement	/req/mf-collection/collection-get /req/mf-collection/collection-get-success
Test purpose	Validate that the MovingFeature Collection can be retrieved from the expected location.
Test method	For every Collection described in the Collections content, issue an HTTP GET request to the URL {root}/collections/{collectionId} where {collectionId} is the id property for the collection 1. Validate that a Collection was returned with a status code 200 2. Validate the contents of the returned document using test /conf/mf-collection/collection-get-success

Abstract Test 6	/conf/mf-collection/collection-get-success
Requirement	/req/mf-collection/collection-get-success /req/mf-collection/mandatory-collection

Test purpose	Validate that the MovingFeature Collection complies with the required structure and contents.
Test method	1. Validate that all response documents comply with OGC API Ñ Common /conf/collections/src-md-success 2. Validate the Collection resource for all supported media types using the resources and tests identified in Table 16 and Table 6

Table 16. Schema and Tests for MovingFeature Collection content

Format	Schema Document	Test ID
HTML	collection.yaml	/conf/html/content
JSON	collection.yaml	/conf/json/content

HTTP PUT Operation

Abstract Test 7	/conf/mf-collection/collection-put
Requirement	/req/mf-collection/collection-put /req/mf-collection/collection-put-success
Test purpose	Validate that the MovingFeature Collection can be replaced at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features PUT operation requirements 2. Validate that a body of a PUT request using for all supported media types using the resources and tests identified in Table 15 3. Validate that the request body complies OGC API Ñ Features PUT request body requirements 4. Issue an HTTP PUT request to the URL `{root}/collections/{collectionId}` 5. Validate the contents of the response using test `/conf/mf-collection/collections-put-success`

Abstract Test 8	/conf/mf-collection/collections-put-success
Requirement	/req/mf-collection/collection-put-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}` PUT request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 200, 202, or 204 2. Validate that all response documents comply with OGC API Ñ Features PUT response requirements

HTTP DELETE Operation

Abstract Test 9	/conf/mf-collection/collection-delete
Requirement	/req/mf-collection/collection-delete /req/mf-collection/collection-delete-success

Test purpose	Validate that the MovingFeature Collection can be deleted at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features DELETE operation requirements 2. Issue an HTTP DELETE request to the URL `{root}/collections/{collectionId}` 3. Validate the contents of the response using test `/conf/mf-collection/collections-put-success`

Abstract Test 10	/conf/mf-collection/collections-delete-success
Requirement	/req/mf-collection/collection-delete-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}` DELETE request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 200, 202, or 204 2. Validate that all response documents comply with OGC API Ñ Features DELETE response requirements

A.3. Conformance Class MovingFeatures

Conformance Class	
http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/movingfeatures	
Target type	Web API
Requirements Class	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
Dependency	http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html
Dependency	http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json
Dependency	http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections
Dependency	http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/simple-query
Dependency	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core
Dependency	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson
Dependency	http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete

A.3.1. MovingFeatures

HTTP GET Operation

Abstract Test 11	/conf/movingfeatures/features-get
Requirement	/req/movingfeatures/features-get /ref/movingfeatures/features-get-success

Test purpose	Validate that MovingFeatures can be identified and extracted from a MovingFeature Collection using query parameters.
Test method	For every MovingFeature Collection identified in MovingFeature Collections, issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items` where `{collectionId}` is the id property for a MovingFeature Collection described in the MovingFeature Collections content 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using test `/conf/movingfeatures/features-get-success` Repeat these tests using the following parameter tests that defined in the OGC API Ñ Common: - Bounding Box: Bounding Box Tests - Limit: Limit Tests - Date-Time: Date-Time Tests Execute requests with combinations of the `"bbox"` and `"datetime"` query parameters and verify that only features are returned that match both selection criteria.

Abstract Test 12	/conf/movingfeatures/features-get-success
Requirement	/ref/movingfeatures/features-get-success
Test purpose	Validate that the MovingFeatures complies with the required structure and contents.
Test method	1. Validate that all response documents comply with OGC API Ñ Features /conf/core/fc-response 2. Validate the Collections resource for all supported media types using the resources and tests identified in Table 17

The MovingFeatures content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 17. Schema and Tests for MovingFeatures *content*

Format	Schema Document	Test ID
HTML	movingFeatureCollection.yaml	/conf/html/content
GeoJSON	movingFeatureCollection.yaml	/conf/geojson/content

HTTP POST Operation

Abstract Test 13	/conf/movingfeatures/features-post

Requirement	/req/movingfeatures/mf-mandatory /req/movingfeatures/features-post /req/movingfeatures/features-post-success
Test purpose	Validate that the MovingFeature can be created at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features POST operation requirements 2. Validate that a body of a POST request using for all supported media types using the resources and tests identified in Table 18 and Table 8 3. Validate that the request body complies OGC API Ñ Features POST request body requirements 4. Issue an HTTP POST request to the URL `{root}/collections/{collectionId}/items` 5. Validate the contents of the response using test `/conf/movingfeatures/features-post-success`

Table 18. Schema and Tests for Request Body of `{root}/collections/{collectionId}/items` POST

Format	Schema Document	Test ID
JSON	MF-JSON_Prism.schema.json	/conf/json/content

Abstract Test 14	/conf/movingfeatures/features-post-success
Requirement	/req/movingfeatures/features-post-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}/items` POST request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 201 or 202 2. Validate that all response documents comply with OGC API Ñ Features POST response requirements

A.3.2. MovingFeature

HTTP GET Operation

Abstract Test 15	/conf/movingfeatures/mf-get
Requirement	/req/movingfeatures/mf-get /ref/movingfeatures/mf-get-success
Test purpose	Validate that the MovingFeature can be retrieved from the expected location.

Test method	For every MovingFeature identified in MovingFeature Collection, issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeaturesId}` where `{collectionId}` is the id property for a MovingFeature Collection described in the MovingFeature Collections content and `{mFeatureId}` is the id property for the MovingFeature 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using test `/conf/movingfeatures/features-get-success`

Abstract Test 16	/conf/movingfeatures/mf-get-success
Requirement	/ref/movingfeatures/mf-get-success
Test purpose	Validate that the MovingFeature complies with the required structure and contents.
Test method	1. Validate that all response documents comply with OGC API Ñ Features /conf/core/f-success 2. Validate the Collections resource for all supported media types using the resources and tests identified in Table 19

The MovingFeature content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 19. Schema and Tests for MovingFeature content

Format	Schema Document	Test ID
HTML	movingFeature.yaml	/conf/html/content
GeoJSON	movingFeature.yaml	/conf/geojson/content

HTTP DELETE Operation

Abstract Test 17	/conf/movingfeatures/mf-delete
Requirement	/req/movingfeatures/mf-delete /req/movingfeatures/mf-delete-success
Test purpose	Validate that the MovingFeature can be deleted at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features DELETE operation requirements 2. Issue an HTTP DELETE request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}` 3. Validate the contents of the response using test `/conf/mf-collection/collections-put-success`

Abstract Test 18	/conf/movingfeatures/mf-delete-success
Requirement	/req/movingfeatures/mf-delete-success

Test purpose	Validate that the response of `{root}/collections/{collectionId}/items/{mFeatureId}` DELETE request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 200, 202, or 204 2. Validate that all response documents comply with OGC API Ñ Features DELETE response requirements

A.3.3. Parameter Leaf

Abstract Test 19	/conf/movingfeatures/param-leaf-definition
Requirement	/req/movingfeatures/param-leaf-definition
Test purpose	Validate that the `leaf` query parameter is constructed correctly.
Test method	Verify that the `leaf` query parameter complies with the definition (using an OpenAPI Specification 3.0 fragment)

Abstract Test 20	/conf/movingfeatures/param-leaf-response
Requirement	/req/movingfeatures/param-leaf-definition /req/movingfeatures/param-leaf-response
Test purpose	Validate that the `leaf` query parameter is processed correctly.
Test method	DO FOR each Resource which have `datetimes` property: 1. Calculate a temporal geometry coordinate (or temporal property value) with the *pointAtTime* query at each time included in the `leaf` parameter, using interpolated trajectory according to the `interpolation` property 2. Verify that the temporal geometry coordinate (or temporal property value) intersects the interpolated trajectory according to the `interpolation` property, using datetime value defined by the `leaf` parameter

A.3.4. TemporalGeometrySequence

HTTP GET Operation

Abstract Test 21	/conf/movingfeatures/tgsequence-get
Requirement	/req/movingfeatures/tgsequence-get /req/movingfeatures/tgsequence-get-success
Test purpose	Validate that the TemporalGeometrySequence can be identified and extracted from a MovingFeature object using query parameters.

Test method	For every TemporalGeometrySequence identified in MovingFeature, issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` where `{collectionId}` is the id property for a MovingFeature Collection described in the MovingFeature Collection content and `{mFeatureId}` is the id property for the MovingFeature 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using test `/conf/movingfeatures/tgsequence-get-success` Repeat these tests using the following parameter tests that defined in the OGC API Ñ Common and OGC API Ñ MF: - Bounding Box: Bounding Box Tests - Limit: Limit Tests - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test Execute requests with combinations of the `"bbox"`, `"datetime"`, and `"leaf"` query parameters and verify that only features are returned that match both selection criteria.

Abstract Test 22	/conf/movingfeatures/tgsequence-get-success
Requirement	/req/movingfeatures/tgsequence-get-success
Test purpose	Validate that the TemporalGeometrySequence complies with the required structure and contents.
Test method	1. Validate that the `type` property is present and has a value of `MovingGeometryCollection` 2. Validate the `prism` property is present and that it is populated with an array of TemporalPrimitiveGeometry items 3. Validate that only TemporalPrimitiveGeometry which match the selection criteria are included in the MovingFeature 4. If the `links` property is present, validate that all entries comply with OGC API Ñ Features /conf/core/fc-links 5. If the `timeStamp` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-timeStamp 6. If the `numberMatched` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberMatched 7. If the `numberReturned` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberReturned 8. Validate the TemporalGeometry resource for all supported media types using the resources and tests identified in Table 20

The TemporalPrimitiveGeometry content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 20. Schema and Tests for TemporalGeometrySequence *content*

Format	Schema Document	Test ID
HTML	temporalGeometrySequence.yaml	/conf/html/content
JSON	temporalGeometrySequence.yaml	/conf/json/content

HTTP POST Operation

Abstract Test 23	/conf/movingfeatures/tgsequence-post
Requirement	/req/movingfeatures/tpgeometry-mandatory /req/movingfeatures/tgsequence-post /req/movingfeatures/tgsequence-post-success
Test purpose	Validate that the TemporalPrimitiveGeometry can be created at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features POST operation requirements 2. Validate that a body of a POST request using for all supported media types using the resources and tests identified in Table 21 and Table 9 3. Validate that the request body complies OGC API Ñ Features POST request body requirements 4. Issue an HTTP POST request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` 5. Validate the contents of the response using test `/conf/movingfeatures/tgeometry-post-success`

Table 21. Schema and Tests for Request Body of `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` *POST*

Format	Schema Document	Test ID
JSON	MF-JSON_Prism.schema.json	/conf/json/content

Abstract Test 24	/conf/movingfeatures/tgsequence-post-success
Requirement	/req/movingfeatures/tgsequence-post-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence` POST request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 201 or 202 2. Validate that all response documents comply with OGC API Ñ Features POST response requirements

A.3.5. TemporalPrimitiveGeometry

HTTP DELETE Operation

Abstract Test 25	/conf/movingfeatures/tpgeometry-delete
Requirement	/req/movingfeatures/tpgeometry-delete /req/movingfeatures/tpgeometry-delete-success
Test purpose	Validate that the TemporalPrimitiveGeometry can be deleted at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features DELETE operation requirements 2. Issue an HTTP DELETE request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tgeometryId}` 3. Validate the contents of the response using test `/conf/mf-collection/collections-put-success`

Abstract Test 26	/conf/mf-collection/tpgeometry-delete-success
Requirement	/req/movingfeatures/tpgeometry-delete-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}` DELETE request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 200, 202, or 204 2. Validate that all response documents comply with OGC API Ñ Features DELETE response requirements

A.3.6. TemporalGeometry Query

HTTP GET Operation

Abstract Test 27	/conf/movingfeatures/tpgeometry-query-distance
Requirement	/req/movingfeatures/tpgeometry-query /req/movingfeatures/tpgeometry-query-success
Test purpose	Validate that resources can be identified and extracted from a TemporalPrimitiveGeometry with a `Distance` query using query parameters.

Test method	IF a query parameter `datetime` is not empty, validate that the query parameter `datetime` with the following parameter tests that defined in the OGC API Ñ Common: - Date-Time: Date-Time Tests 1. Issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/distance` 2. Validate that a document was returned with a status code `200` 3. Verify the `type` is `"TReal"` IF a query parameter `datetime` is not empty: Execute requests with `datetime` query parameter and verify the correctly calculated value is returned. IF a query parameter `datetime` is empty: Verify that a time-to-distance curve is correctly returned according to the specified TemporalPrimitiveGeometry resource by {tGeometryId}.

Abstract Test 28	/conf/movingfeatures/tpgeometry-query-velocity
Requirement	/req/movingfeatures/tpgeometry-query /req/movingfeatures/tpgeometry-query-success
Test purpose	Validate that resources can be identified and extracted from a TemporalPrimitiveGeometry with a `Velocity` query using query parameters.
Test method	IF a query parameter `datetime` is not empty, validate that the query parameter `datetime` with the following parameter tests that defined in the OGC API Ñ Common: - Date-Time: Date-Time Tests 1. Issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/velocity` 2. Validate that a document was returned with a status code `200` 3. Verify the `type` is `"TReal"` IF a query parameter `datetime` is not empty: Execute requests with `datetime` query parameter and verify the correctly calculated value is returned. IF a query parameter `datetime` is empty: Verify that a time-to-velocity curve is correctly returned according to the specified TemporalPrimitiveGeometry resource by {tGeometryId}.

Abstract Test 29	/conf/movingfeatures/tpgeometry-query-acceleration

Requirement	/req/movingfeatures/tpgeometry-query /req/movingfeatures/tpgeometry-query-success
Test purpose	Validate that resources can be identified and extracted from a TemporalPrimitiveGeometry with a `Acceleration` query using query parameters.
Test method	IF a query parameter `datetime` is not empty, validate that the query parameter `datetime` with the following parameter tests that defined in the OGC API Ñ Common: - Date-Time: Date-Time Tests 1. Issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/acceleration` 2. Validate that a document was returned with a status code `200` 3. Verify the `type` is `"TReal"` IF a query parameter `datetime` is not empty: Execute requests with `datetime` query parameter and verify the correctly calculated value is returned. IF a query parameter `datetime` is empty: Verify that a time-to-acceleration curve is correctly returned according to the specified TemporalPrimitiveGeometry resource by {tGeometryId}.

A.3.7. TemporalProperties

HTTP GET Operation

Abstract Test 30	/conf/movingfeatures/tproperties-get
Requirement	/req/movingfeatures/tproperties-get /req/movingfeatures/tproperties-get-success
Test purpose	Validate that the TemporalProperties can be identified and extracted from a MovingFeature object using query parameters.

Test method	For every TemporalProperty identified in MovingFeature, issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` where `{collectionId}` is the id property for a MovingFeature Collection described in the MovingFeature Collection content and `{mFeatureId}` is the id property for the MovingFeature 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using test `/conf/movingfeatures/tproperties-get-success` Repeat these tests using the following parameter tests that are defined in the OGC API Ñ Common: - Limit: Limit Tests - Date-Time: Date-Time Tests Execute requests with combinations of the `"datetime"` query parameters and verify that only features are returned that match both selection criteria.

Abstract Test 31	/conf/movingfeatures/tproperties-get-success
Requirement	/req/movingfeatures/tproperties-get-success
Test purpose	Validate that the TemporalProperties complies with the required structure and contents.
Test method	1. Validate the TemporalProperties property is present and that it is populated with an array of TemporalProperty items 2. Validate that the `name` and `type` property is present 3. Validate the `type` property is present and its value is one of the predefined values (i.e., one of 'TBoolean', 'TText', 'TInteger', 'TReal', and 'TImage') 4. If the `links` property is present, validate that all entries comply with OGC API Ñ Features /conf/core/fc-links 5. If the `timeStamp` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-timeStamp 6. If the `numberMatched` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberMatched 7. If the `numberReturned` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberReturned 8. Validate the TemporalProperties resource for all supported media types using the resources and tests identified in Table 22

The TemporalProperties content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 22. Schema and Tests for TemporalProperties *content*

Format	Schema Document	Test ID
HTML	temporalPropertyCollection.yaml	/conf/html/content
JSON	temporalPropertyCollection.yaml	/conf/json/content

HTTP POST Operation

Abstract Test 32	/conf/movingfeatures/tproperties-post
Requirement	/req/movingfeatures/tproperty-mandatory /req/movingfeatures/tproperties-post /req/movingfeatures/tproperties-post-success
Test purpose	Validate that the TemporalProperty can be created at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features POST operation requirements 2. Validate that a body of a POST request using for all supported media types using the resources and tests identified in Table 23 and Table 11 3. Validate that the request body complies OGC API Ñ Features POST request body requirements 4. Issue an HTTP POST request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` 5. Validate the contents of the response using test `/conf/movingfeatures/tproperties-post-success`

Table 23. Schema and Tests for Request Body of `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` *POST*

Format	Schema Document	Test ID
HTML	tproperty_requestbody.yaml	/conf/html/content
JSON	tproperty_requestbody.yaml	/conf/json/content
JSON	MF-JSON_Prism.schema.json	/conf/json/content

Abstract Test 33	/conf/movingfeatures/tproperties-post-success
Requirement	/req/movingfeatures/tproperties-post-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` POST request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 201 or 202 2. Validate that all response documents comply with OGC API Ñ Features POST response requirements

A.3.8. TemporalProperty

HTTP GET Operation

Abstract Test 34	/conf/movingfeatures/tproperty-get
Requirement	/req/movingfeatures/tproperty-get /req/movingfeatures/tproperty-get-success
Test purpose	Validate that the TemporalProperty can be identified and extracted from a TemporalProperties using query parameters.
Test method	For every TemporalProperty identified in MovingFeature, issue an HTTP GET request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tpropertyName}` where `{collectionId}` is the id property for a MovingFeature Collection described in the MovingFeature Collections content, `{mFeatureId}` is the id property for the MovingFeature, `{tpropertyName}` is the name property for the TemporalProperty 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using test `/conf/movingfeatures/tproperty-get-success` Repeat these tests using the following parameter tests that defined in the OGC API Ñ Common and OGC API Ñ MF: - Limit: Limit Tests - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test Execute requests with combinations of the `"datetime"` and `"leaf"` query parameters and verify that only features are returned that match both selection criteria.

Abstract Test 35	/conf/movingfeatures/tproperty-get-success
Requirement	/req/movingfeatures/tproperty-get-success
Test purpose	Validate that the TemporalProperty complies with the required structure and contents.
Test method	1. Validate the TemporalProperties property is present and that it is populated with an array of TemporalPrimitiveValue items 2. If the `links` property is present, validate that all entries comply with OGC API Ñ Features /conf/core/fc-links 3. If the `timeStamp` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-timeStamp 4. If the `numberMatched` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberMatched 5. If the `numberReturned` property is present, validate that it complies with OGC API Ñ Features /conf/core/fc-numberReturned 6. Validate the TemporalProperty resource for all supported media types using the resources and tests identified in Table 24

The TemporalProperty content may be retrieved in a number of different formats. The following table identifies the applicable schema document for each format and the test to be used to validate the against that schema. All supported formats should be exercised.

Table 24. Schema and Tests for TemporalProperty *content*

Format	Schema Document	Test ID
HTML	temporalProperty.yaml	/conf/html/content
JSON	temporalProperty.yaml	/conf/json/content

HTTP POST Operation

Abstract Test 36	/conf/movingfeatures/tproperty-post
Requirement	/req/movingfeatures/tproperty-mandatory /req/movingfeatures/tproperty-post /req/movingfeatures/tproperty-post-success
Test purpose	Validate that the TemporalPrimitiveValue can be created at the expected location.
Test method	1. Validate that the server complies with OGC API Ñ Features POST operation requirements 2. Validate that a body of a POST request using for all supported media types using the resources and tests identified in Table 25 and Table 12 3. Validate that the request body complies OGC API Ñ Features POST request body requirements 4. Issue an HTTP POST request to the URL `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` 5. Validate the contents of the response using test `/conf/movingfeatures/tproperty-post-success`

Table 25. Schema and Tests for Request Body of `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` *POST*

Format	Schema Document	Test ID
HTML	tvalue_requestbody.yaml	/conf/html/content
JSON	tvalue_requestbody.yaml	/conf/json/content

Abstract Test 37	/conf/movingfeatures/tproperty-post-success
Requirement	/req/movingfeatures/tproperty-post-success
Test purpose	Validate that the response of `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` POST request complies with the required structure and contents.
Test method	1. Validate that a document was returned with a status code 201 or 202 2. Validate that all response documents comply with OGC API Ñ Features POST response requirements

Annex B: Relationship with other OGC/ISO Standards (Informative)

This specification is built upon the following OGC/ISO standards. The geometry concept is presented first, followed by the feature concept. Note that a feature is *not* a geometry. However, a feature often contains a geometry as one of its attributes. However, it is legal to build features without a geometry attribute, or with more than one geometry attributes.

B.1. Static geometries, features and accesses

The following standards define static objects, without time-varying properties.

B.1.1. Geometry (ISO 19107)

The ISO 19107, *Geographic information Ñ Spatial schema* standard defines a `GM_Object` base type which is the root of all geometric objects. Some examples of `GM_Object` subtypes are `GM_Point`, `GM_Curve`, `GM_Surface` and `GM_Solid`. A `GM_Object` instance can be regarded as an infinite set of points in a particular coordinate reference system. The standard provides a `GM_CurveInterpolation` code list to identify how those points are computed from a finite set of points. Some interpolation methods listed by ISO 19107 are (non-exhaustive list):

linear

Positions on a straight line between each consecutive pair of control points.

geodesic

Positions on a geodesic curve between each consecutive pair of control points. A geodesic curve is a curve of shortest length. The geodesic shall be determined in the coordinate reference system of the curve.

circularArc3Points

For each set of three consecutive control points, a circular arc passing from the first point through the middle point to the third point. Note: if the three points are co-linear, the circular arc becomes a straight line.

elliptical

For each set of four consecutive control points, an elliptical arc passing from the first point through the middle points in order to the fourth point. Note: if the four points are co-linear, the arc becomes a straight line. If the four points are on the same circle, the arc becomes a circular one.

cubicSpline

The control points are interpolated using initial tangents and cubic polynomials, a form of degree 3 polynomial spline.

The UML below shows the `GM_Object` base type with its operations (e.g. `distance(É)` for computing the distance between two geometries). `GM_Curve` (not shown in this UML) is a subtype of

GM_Primitive. All operations assume static objects, without time-varying coordinates or attributes.

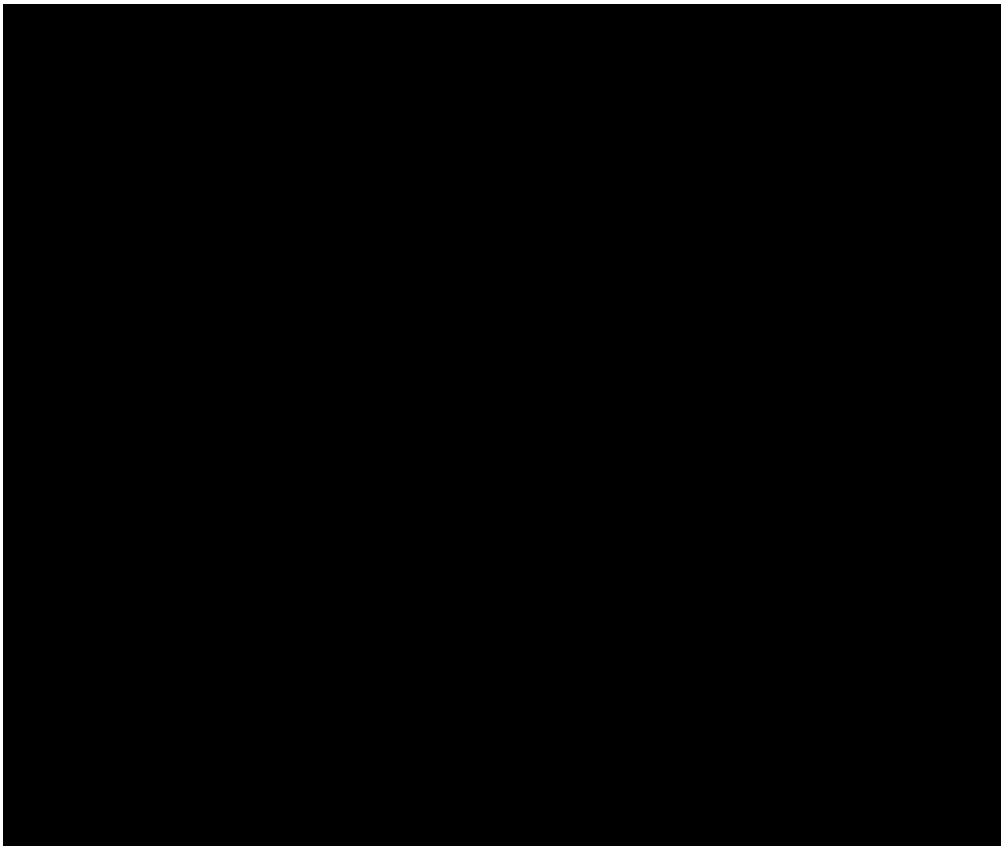

Figure 5. GM_Object from ISO 19107:2003 figure 6

Geometry, topology and temporal-objects (GM_Object, TP_Object, TM_Object) are not abstractions of real-world phenomena. These types can provide types for feature properties as described in the next section but cannot be specialized to features.

B.1.2. Features (ISO 19109)

The ISO 19109, *Geographic information Ñ Rules for application schema* standard defines types for the definition of features. A feature is an abstraction of a real-world phenomena. The terms Òfeature typeÓ and Òfeature instanceÓ are used to separate the following concepts of ÒfeatureÓ:

Feature type

> The whole collection of real-world phenomena classified in a concept. For example the ÒbridgeÓ feature type is the abstraction of the collection of all real-world phenomena that is classified into the concept behind the term ÒbridgeÓ.

Feature instance

> A certain occurrence of a feature type. For example ÒTower BridgeÓ feature instance is the abstraction of a certain real-world bridge in London.

In object-oriented modelling, feature types are equivalent to classes and feature instances are equivalent to objects,

The UML below shows the General Feature Model. FeatureType is a metaclass that is instantiated as classes that represent individual feature types. A FeatureType instance contains the list of properties (attributes, associations and operations) that feature instances of that type can contain. Geometries

are properties like any other, without any special treatment. All properties are static, without time-varying values.

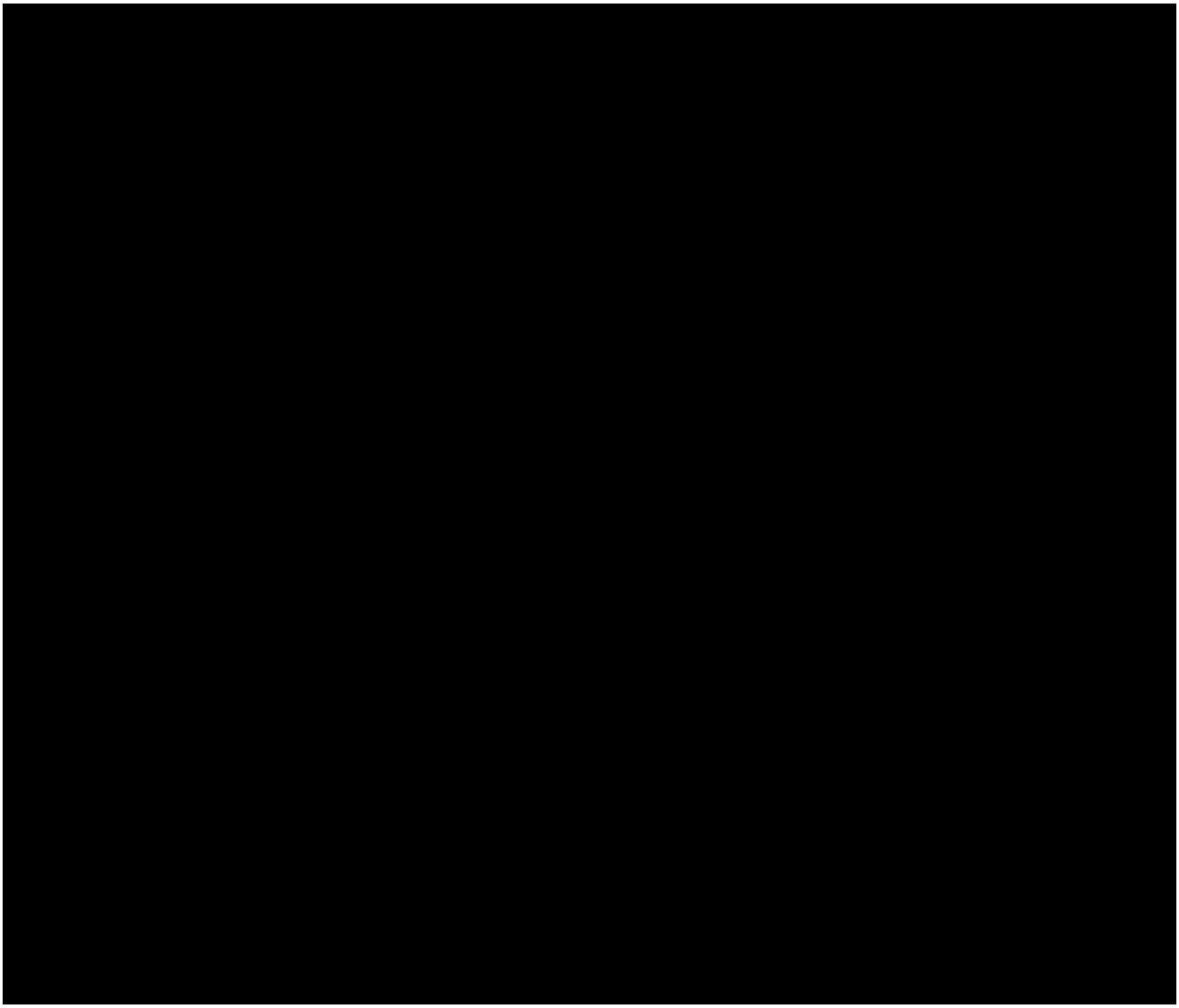

Figure 6. General Feature Model from ISO 19109:2009 figure 5

B.1.3. Simple Features SQL

The Simple Feature Access Ñ Part 2: SQL Option Standard describes a feature access implementation in SQL based on a profile of ISO 19107. This standard defines *feature table* as a table where the columns represent feature attributes, and the rows represent feature instances. The geometry of a feature is one of its feature attributes.

B.1.4. Filter Encoding (ISO 19143)

The ISO 19143, *Geographic information Ñ Filter encoding* standard (also OGC Standard) provides types for constructing queries. These objects can be transformed into a SQL ÒSELECT É FROM É WHERE É ORDER BY ÉÓ statement to fetch data stored in a SQL-based relational database. Similarly, the same objects can be transformed into a XQuery expression in order to retrieve data from XML document. The UML below shows the objects used for querying a subset based on spatial operations such as Òcontains Ó or Òintersects Ó.

Figure 7. Spatial operators from ISO 19143 figure 6

B.1.5. Features web API

The OGC 17-069, *Features Ñ Part 1: Core* Standard specifies the fundamental building blocks for interacting with features using a Web API pattern. This Draft Standards defines how to get all features available on a server, or to get feature instances by their identifier.

B.1.6. Features Filtering web API

The draft OGC TBD, *Features Ñ Part 3: Filtering and the Common Query Language (CQL)* standard extends the Feature web API with capabilities to encode more sophisticated queries. The conceptual model is close to ISO 19143.

B.2. Temporal Geometries and Moving Features

B.2.1. Moving Features (ISO 19141)

The ISO 19141, *Geographic information Ñ Schema for moving features* standard extends the ISO 19107 spatial schema for addressing features whose locations change over time. Despite the ÒMoving FeaturesÓ name, that standard is more about ÒMoving geometriesÓ. The UML below shows how the `MF_Trajectory` type extends the ÒstaticÓ types from ISO 19107.

Figure 8. Trajectory type from ISO 19141 figure 3

Trajectory inherits operations shown below. Those operations are in addition to the operations inherited from `GM_Object`. For example the `distance(É)` operation from ISO 19107 is now completed by a `nearestApproach(É)` operation.

Figure 9. Temporal geometry from ISO 19141 figure 6

B.2.2. Moving Features XML encoding (OGC 18-075)

The OGC 18-075 *Moving Features Encoding Part I: XML Core* Standard takes a subset of the ISO 19141 Standard and defines an XML encoding. That standard also completes ISO 19141 by allowing to specify attributes whose value change over time. This extension to the above *General Feature Model* is shown below:

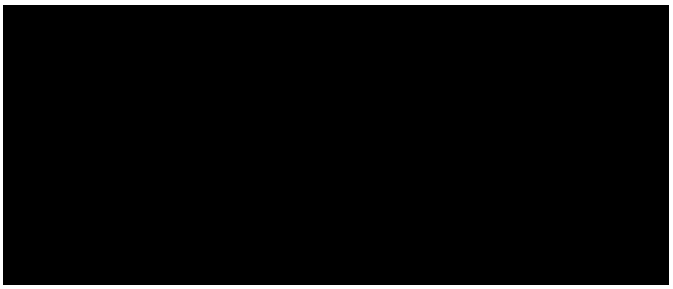

Figure 10. Dynamic attribute from OGC 18-075 figure 3

B.2.3. Moving Features JSON encoding (OGC 19-045)

The OGC 19-045 *Moving Features Encoding Extension Ñ JSON* Standard takes a subset of the ISO 19141 Standard and defines a JSON encoding. The Standard provides various UML diagrams summarizing ISO 19141.

Annex C: Revision History

Date	Release	Editor	Primary clauses modified	Description
2021-09-14	0.1	Taehoon Kim, Kyoung-Sook Kim, and Martin Desruisseaux	all	first draft version
2022-03-01	0.2	Taehoon Kim, Kyoung-Sook Kim	all	revised sections related to resources to add CRUD operations
2022-10-09	0.3	Taehoon Kim, Kyoung-Sook Kim	all	added TemporalGeometry Query resources
2023-02-21	0.9	Taehoon Kim, Kyoung-Sook Kim, Mahmoud, and Esteban	all	finalize draft version
2023-05-19	0.9.9	Taehoon Kim, Kyoung-Sook Kim, Mahmoud, and Esteban	all	finalize draft version
2023-07-10	1.0.draft	Taehoon Kim, Kyoung-Sook Kim, Mahmoud, and Esteban	all	finalize draft version

Annex D: Bibliography

[1] OGC: OGC Moving Features Encoding Extension Ñ JSON. (2020).

[2] OGC: OGC Moving Features Access. (2017).

[3] OGC: OGC API Ñ Features Ñ Part 1: Core. (2019).

[4] OGC: OGC API Ñ Features Ñ Part 2: Coordinate Reference Systems by Reference. (2020).

[5] OGC: OGC API Ñ Features Ñ Part 4: Create, Replace, Update and Delete. (2020).

[6] OGC: OGC API Ñ Features, https://ogcapi.ogc.org/features/

[7] OGC: OGC API Ñ Common, https://ogcapi.ogc.org/common/

[8] OGC: OGC API, https://ogcapi.ogc.org/

[9] OpenAPI, https://www.openapis.org/