OGC® DOCUMENT: 22-003

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

Open Geospatial Consortium

OGC API - MOVING FEATURES - PART 1: CORE

STANDARD
Implementation

DRAFT

Version: 1.0.draft
Submission Date: 2023-05-19
Approval Date: 2029-03-30
Publication Date: 2029-03-30
Editor: Taehoon Kim, Kyoung-Sook Kim, Mahmoud SAKR, Martin Desruisseaux

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.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Suggested additions, changes and comments on this document are welcome and encouraged. Such suggestions may be submitted using the online change request form on OGC web site: http://portal.opengeospatial.org/public_ogc/change_request.php

Copyright notice

Note

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.

CONTENTS

I. ABSTRACT ix

II. KEYWORDS x

III. PREFACE xi

IV. SECURITY CONSIDERATIONS xii

V. SUBMITTING ORGANIZATIONS xiii

VI. SUBMITTERS xiii

1. SCOPE 2

2. CONFORMANCE 4

3. NORMATIVE REFERENCES 6

4. TERMS AND DEFINITIONS 9

5. CONVENTIONS 15
   5.1. Identifiers 15
   5.2. Use of HTTPS 15

6. OVERVIEW 17
   6.1. General 17
   6.2. Search 19
   6.3. Dependencies 20

7. REQUIREMENTS CLASS “MOVING FEATURE COLLECTION CATALOG” 23
   7.1. Overview 23
   7.2. Information Resources 23
   7.3. Resource Collections 24
   7.4. Resource Collection 29

8. REQUIREMENTS CLASS “MOVING FEATURES” 37
   8.1. Overview 37
   8.2. Information Resources 42
   8.3. Resource MovingFeatures 43
   8.4. Resource MovingFeature 53
   8.5. Resource TemporalGeometrySequence 58

8.6. Resource TemporalPrimitiveGeometry 66
8.7. TemporalGeometry Query Resources 69
8.8. Resource TemporalProperties 73
8.9. Resource TemporalProperty 81

9. COMMON REQUIREMENTS 89
9.1. Parameters 89
9.2. HTTP Response 90
9.3. HTTP Status Codes 91

ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE) 95
A.1. Introduction 95
A.2. Conformance Class MovingFeature Collection Catalog 95
A.3. Conformance Class MovingFeatures 101

ANNEX B (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE) 123
B.1. Static geometries, features and accesses 123
B.2. Temporal Geometries and Moving Features 128

ANNEX C (INFORMATIVE) REVISION HISTORY 133

BIBLIOGRAPHY 136

LIST OF TABLES

Table 1 — Overview of Resources ix
Table — Overview of Resources xiii
Table 2 — Conformance class URIs 4
Table 3 — Moving Features API Paths 17
Table 4 — Mapping OGC API — MF Sections to OGC API — Common, OGC API — Features, and OGC MF-JSON Requirements Classes 20
Table 5 — Moving Feature Collection Catalog Resources 24
Table 6 — Table of collection properties 29
Table 7 — MovingFeatures Resources 42
Table 8 — Table of the properties related to the moving feature 53
Table 9 — Table of the properties related to the TemporalPrimitiveGeometry 67
Table 10 — Table of the query resources 70
Table 11 72
Table 12 — Table of the properties related to the temporal property 81
Table 13 — Table of the properties related to the temporal primitive value 82

Table 14 — Typical HTTP status codes 91
Table A.1 — Schema and Tests for MovingFeature Collections content 97
Table A.2 — Schema and Tests for Request Body of +{root}+/collections POST 98
Table A.3 — Schema and Tests for MovingFeature Collection content 99
Table A.4 — Schema and Tests for MovingFeatures content 106
Table A.5 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items POST 107
Table A.6 — Schema and Tests for MovingFeature content 109
Table A.7 — Schema and Tests for TemporalGeometrySequence content 112
Table A.8 — Schema and Tests for Request Body of +{root}+/collections/+{collectionId}+/items/+{mFeatureId}+/tgsequence POST 112
Table A.9 — Schema and Tests for TemporalProperties content 117
Table A.10 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items/{mFeatureId}/tproperties POST 117
Table A.11 — Schema and Tests for TemporalProperty content 119
Table A.12 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName} POST 120
Table B.1 — A non-exhaustive list of interpolation methods listed by ISO 19107 123
Table C.1 133

LIST OF FIGURES

Figure 1 — Class diagram for OGC API — Moving Features 19
Figure 2 — Collection Request Body Schema: 25
Figure 3 — An Example of Creating a New Collection: 26
Figure 4 — Collections GET Response Schema (collections.yaml): 27
Figure 5 — An Example of Collections JSON Payload: 27
Figure 6 — An Example of Replacing an Existing Collection: 31
Figure 7 — An Example of Deleting an Existing Collection: 32
Figure 8 — Collection GET Response Schema (collection.yaml) 33
Figure 9 — An Example of Collection GET Operation: 33
Figure 11 — Example of a response result with subTrajectory parameter 45
Figure 12 — An Example of Creating a New MovingFeature Object: 46
Figure 13 — MovingFeatures GET Response Schema (movingFeatureCollection.yaml): 50
Figure 14 — An Example of a MovingFeatures GET Operation: 51
Figure 15 — MovingFeature GET Response Schema (movingFeature.yaml): 56
Figure 16 — An Example of a MovingFeature JSON Payload: 57
Figure 18 — leaf parameter valid (and invalid) Examples 60
Figure 19 — Example of a response result with leaf parameter 61
Figure 20 — An Example of Creating a New TemporalPrimitiveGeometry Object: 62

Figure 21 — TemporalGeometrySequence GET Response Schema (temporalGeometrySequence.yaml): 64
Figure 22 — An Example of a TemporalGeometrySequence GET operation: 65
Figure 23 — Example of time-to-distance curve [OGC 16-120r3, OGC Moving Features Access] 70
Figure 25 — TemporalProperties Request Body Schema (temporalProperty.yaml): 75
Figure 26 — An Example of Creating a New TemporalProperty Object: 76
Figure 27 — An Example of Creating a New TemporalProperty Object with ParametricValues as a MF-JSON encoding: 77
Figure 28 — TemporalProperties GET Response Schema (TemporalProperties.yaml): 78
Figure 29 — An Example of a TemporalProperties GET Operation: 79
Figure 30 — An Example of a TemporalProperties GET Operation with subTemporalValue: 79
Figure 31 — TemporalProperty Request Body Schema (TemporalPrimitiveValue.yaml): 84
Figure 32 — An Example of Creating a New TemporalPrimitiveValue Object: 84
Figure 33 — An Example of TemporalProperty GET Operation: 86
Figure 34 — An Example of the HTTP Response Schema: 91
Figure B.1 — GM_Object from ISO 19107:2003 figure 6 124
Figure B.2 — General Feature Model from ISO 19109:2009 figure 5 126
Figure B.3 — Spatial operators from ISO 19143 figure 6 127
Figure B.4 — Trajectory type from ISO 19141 figure 3 128
Figure B.5 — Temporal geometry from ISO 19141 figure 6 129
Figure B.6 — Dynamic attribute from OGC 18-075 figure 3 129

LIST OF RECOMMENDATIONS

REQUIREMENTS CLASS 1: REQUIREMENTS CLASS ‘MOVING FEATURE COLLECTION CATALOG’ 23
REQUIREMENTS CLASS 2: REQUIREMENTS CLASS ‘MOVING FEATURES’ 37
REQUIREMENTS CLASS 3: REQUIREMENTS CLASS ‘MOVING FEATURES — COMMON’ 89
REQUIREMENT 1 25
REQUIREMENT 2 25
REQUIREMENT 3 26
REQUIREMENT 4 28
REQUIREMENT 5 29
REQUIREMENT 6 30
REQUIREMENT 7 31
REQUIREMENT 8 32

REQUIREMENT 9 32
REQUIREMENT 10 34
REQUIREMENT 11 35
REQUIREMENT 12 44
REQUIREMENT 13 44
REQUIREMENT 14 46
REQUIREMENT 15 46
REQUIREMENT 16 50
REQUIREMENT 17 52
REQUIREMENT 18 54
REQUIREMENT 19 55
REQUIREMENT 20 55
REQUIREMENT 21 56
REQUIREMENT 22 58
REQUIREMENT 23 59
REQUIREMENT 24 59
REQUIREMENT 25 61
REQUIREMENT 26 62
REQUIREMENT 27 64
REQUIREMENT 28 66
REQUIREMENT 29 67
REQUIREMENT 30 68
REQUIREMENT 31 69
REQUIREMENT 32 72
REQUIREMENT 33 72
REQUIREMENT 34 73
REQUIREMENT 35 74
REQUIREMENT 36 75
REQUIREMENT 37 75
REQUIREMENT 38 78
REQUIREMENT 39 80
REQUIREMENT 40 82
REQUIREMENT 41 83

REQUIREMENT 42 83
REQUIREMENT 43 85
REQUIREMENT 44 87
REQUIREMENT 45 89
REQUIREMENT 46 90
REQUIREMENT 47 90
REQUIREMENT 48 91
CONFORMANCE CLASS A.1 95
CONFORMANCE CLASS A.2 101

I ABSTRACT

Moving feature data can represent various phenomena, including vehicles, people, animals, weather patterns, etc. The OGC API — Moving Features (OGC API — 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 — 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 — MF also includes operations for filtering, sorting, and aggregating moving feature data based on location, time, and other properties.

The OGC API — Moving Features — 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 Moving Features
MovingFeature instance	`/collections/+{collectionId}+/items /+{mFeatureId}+`	GET, DELETE	Resource Moving Feature
TemporalGeometry Sequence	`/collections/+{collectionId}+/items /+{mFeatureId}+/tgsequence`	GET, POST	Resource Temporal GeometrySequence
TemporalPrimitive Geometry instance	`/collections/+{collectionId}+/ items /+{mFeatureId}+/tgsequence/ +{tGeometryId}+`	DELETE	Resource Temporal PrimitiveGeometry
Queries for Temporal PrimitiveGeometry	`collections/+{collectionId}+/ items /+{mFeatureId}+/tgsequence/ +{tGeometryId}+/+{queryType}+`	GET	TemporalGeometry Query Resources
TemporalProperties instance	`/collections/+{collectionId}+/items /+{mFeatureId}+/tproperties`	GET, POST	Resource Temporal Properties

RESOURCE	PATH	HTTP METHOD	DOCUMENT REFERENCE
TemporalProperty instance	`/collections/+{collectionId}+/ items /+{mFeatureId}+/tproperties/ +{tPropertyId}+`	GET, POST	Resource Temporal Property

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

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 — Moving Features — 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 — Common — 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
- Université 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:

Table — Overview of Resources

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	Université libre de Bruxelles
Esteban Zimanyi	Université libre de Bruxelles
Martin Desruisseaux	Geomatys
Akinori Asahara	Central Research Laboratory, Hitachi Ltd.
Chen-Yu Hao	Feng Chia University

1 SCOPE

1 SCOPE

The scope of the OGC API — Moving Features — 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 — 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 — Moving Features — 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 — Moving Features Draft Standard is an extension of the OGC API — Common and the OGC API — Features Standards.

2

CONFORMANCE

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 — Moving Features are:

- Collection Catalog
- Moving Features

The conformance class defines the minimum requirements for an API to be compliant with the OGC API — 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 — 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.

3

NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Policy SWG: OGC 08-131r3, *The Specification Model — Standard for Modular specifications*. Open Geospatial Consortium (2009).

Hideki Hayashi, Akinori Asahara, Kyoung-Sook Kim, Ryosuke Shibasaki, Nobuhiro Ishimaru: OGC 16-120r3, *OGC Moving Features Access*. Open Geospatial Consortium (2017). http://www.opengis.net/doc/is/movingfeatures-access/1.0.

Kyoung-Sook KIM, Nobuhiro ISHIMARU: OGC 19-045r3, *OGC Moving Features Encoding Extension — JSON*. Open Geospatial Consortium (2020). http://www.opengis.net/doc/IS/mf-json/1.0.

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1.

Charles Heazel: OGC 19-072, *OGC API — Common — Part 1: Core*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/is/ogcapi-common-1/1.0.

Charles Heazel: OGC API — Common — Part 2: Geospatial Data (Draft). OGC 20-024, Open Geospatial Consortium, http://docs.ogc.org/DRAFTS/20-024.html

Panagiotis A. Vretanos, Clemens Portele: OGC API — Features — Part 4: Create, Replace, Update and Delete (Draft). http://docs.ogc.org/DRAFTS/20-002.html

E. Levinson: IETF RFC 2387, *The MIME Multipart/Related Content-type*. RFC Publisher (1998). https://www.rfc-editor.org/info/rfc2387.

E. Rescorla: IETF RFC 2818, *HTTP Over TLS*. RFC Publisher (2000). https://www.rfc-editor.org/info/rfc2818.

G. Klyne, C. Newman: IETF RFC 3339, *Date and Time on the Internet: Timestamps*. RFC Publisher (2002). https://www.rfc-editor.org/info/rfc3339.

T. Berners-Lee, R. Fielding, L. Masinter: IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*. RFC Publisher (2005). https://www.rfc-editor.org/info/rfc3986.

H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: IETF RFC 7946, *The GeoJSON Format*. RFC Publisher (2016). https://www.rfc-editor.org/info/rfc7946.

M. Nottingham: IETF RFC 8288, *Web Linking*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8288.

T. Bray (ed.): IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8259.

R. Fielding, J. Reschke (eds.): IETF RFC 7230, *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7230.

R. Fielding, J. Reschke (eds.): IETF RFC 7231, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7231.

R. Fielding, J. Reschke (eds.): IETF RFC 7232, *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7232.

R. Fielding, Y. Lafon, J. Reschke (eds.): IETF RFC 7233, *Hypertext Transfer Protocol (HTTP/1.1): Range Requests*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7233.

R. Fielding, M. Nottingham, J. Reschke (eds.): IETF RFC 7234, *Hypertext Transfer Protocol (HTTP/1.1): Caching*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7234.

R. Fielding, J. Reschke (eds.): IETF RFC 7235, *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7235.

Open API Initiative: OpenAPI Specification 3.1.0, https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.1.0.md

4

TERMS AND DEFINITIONS

4 TERMS AND DEFINITIONS

This document uses 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 “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document 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.

4.1. 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]

4.2. 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: ISO 19111]

4.3. 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: ISO 19111]

4.4. **dataset**

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

4.5. **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: ISO 19103]

4.6. **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]

4.7. **dynamic attribute**

characteristic of a feature in which its value varies with time
[source: OGC 19-045r3]

4.8. **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: ISO 19109]

4.9. 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: ISO 19109]

4.10. feature table

table where the columns represent feature attributes, and the rows represent features
[source: OGC 06-104r4]

4.11. geographic feature

representation of real-world phenomenon associated with a location relative to the Earth
[source: ISO 19101-2]

4.12. geometric object

spatial object representing a geometric set
[source: ISO 19107:2003]

4.13. leaf

<one parameter set of geometries>
geometry at a particular value of the parameter
[source: ISO 19141]

4.14. 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: ISO 19141]
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: ISO 19141]
[source: OGC 19-045r3]

4.15. property

facet or attribute of an object referenced by a name
[source: ISO 19143]

4.16. resource

entity that might be identified
Note 1 to entry: The term “resource”, 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 — DCMI Metadata Terms]

4.17. 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]

4.18. trajectory

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

[source: ISO 19141]

4.19. web API

API using an architectural style that is founded on the technologies of the Web
[source: W3C Data on the Web Best Practices]

4.20. web resource

a resource that is identified by a URI.
[source: OGC 17-069r4]

5

CONVENTIONS

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 “HTTP or HTTPS”. In fact, most servers are expected to use HTTPS and not HTTP.

6

OVERVIEW

6 OVERVIEW

6.1. General

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

This OGC API — Moving Features — Part 1: 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 (OGC 19-045r3).

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 — Common — 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}`

PATH TEMPLATE	RELATION	RESOURCE
	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}`
{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 **TemporalPrimitive Geometry** 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 **TemporalPrimitive Geometry** 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 **Temporal Property** 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 — Moving Features (OGC API — 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.

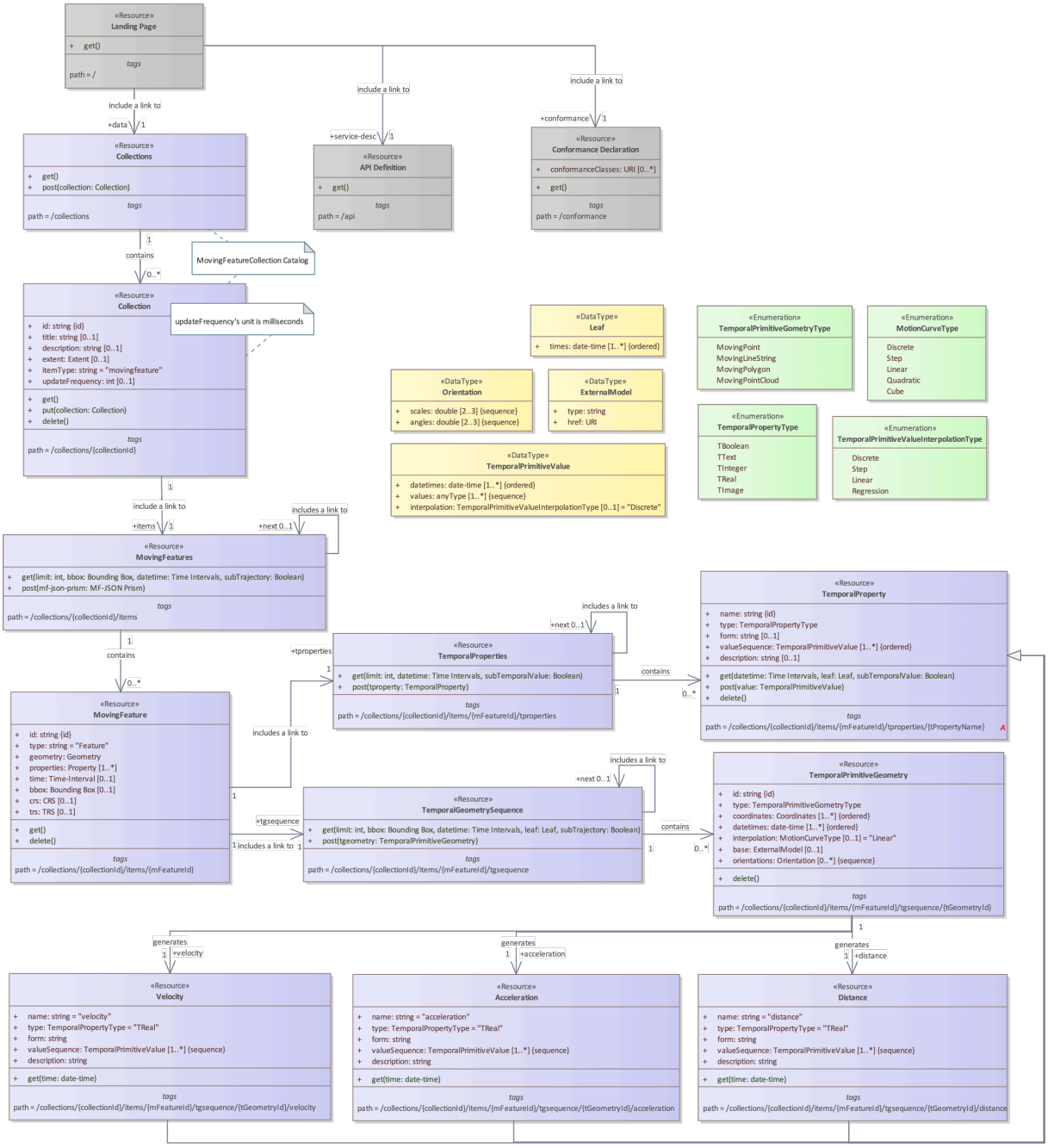

Figure 1 — Class diagram for OGC API — Moving Features

6.2. Search

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

- bounding box searches,

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

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

- spatiotemporal queries for accessing **TemporalGeometry** resources.

6.3. Dependencies

The OGC API — Moving Features (OGC API — MF) Draft Standard is an extension of the OGC API — Common and the OGC API — Features Standards. Therefore, an implementation of OGC API — MF shall first satisfy the appropriate Requirements Classes from OGC API — Common and OGC API — Features. Also, the OGC API — MF Standard is based on the OGC Moving Features Encoding Extension for JSON (OGC MF-JSON) Standards. Therefore, an implementation of OGC API — MF shall satisfy the appropriate Requirements Classes from OGC MF-JSON. Table 4, Identifies the OGC API — Common and OGC API — 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 — MF Sections to OGC API — Common, OGC API — Features, and OGC MF-JSON Requirements Classes

API — MF SECTION	API — MF REQUIREMENTS CLASS	API — COMMON, API — 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
Moving Features	/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

API — MF SECTION	API — MF REQUIREMENTS CLASS	API — COMMON, API — FEATURES, MF-JSON REQUIREMENTS CLASS
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

7

REQUIREMENTS CLASS "MOVING FEATURE COLLECTION CATALOG"

7 REQUIREMENTS CLASS "MOVING FEATURE COLLECTION CATALOG"

7.1. Overview

REQUIREMENTS CLASS 1: REQUIREMENTS CLASS 'MOVING FEATURE COLLECTION CATALOG'	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/mf-collection`
PREREQUISITES	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
NORMATIVE STATEMENTS	Requirement 1: `/req/mf-collection/collections-get` Requirement 2: `/req/mf-collection/collections-post` Requirement 3: `/req/mf-collection/collections-get-success` Requirement 4: `/req/mf-collection/collections-post-success` Requirement 5: `/req/mf-collection/mandatory-collection` Requirement 6: `/req/mf-collection/collection-get` Requirement 7: `/req/mf-collection/collection-put` Requirement 8: `/req/mf-collection/collection-delete` Requirement 9: `/req/mf-collection/collection-get-success` Requirement 10: `/req/mf-collection/collection-put-success` Requirement 11: `/req/mf-collection/collection-delete-success`

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

7.3.2.1. Retrieve

The retrieve operation is defined in the `Collections` conformance class of OGC API — 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 — Common.

REQUIREMENT 1	
IDENTIFIE	/req/mf-collection/collections-get
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Common **Collections** operation requirement /req/collections/rc-md-op.

7.3.2.1.1. Create

The create operation is defined in the CREATE conformance class of OGC API — 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 — Features.

REQUIREMENT 2	
IDENTIFIE	/req/mf-collection/collections-post
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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.

```
type: object
required:
  - itemType
properties:
  title:
    description: human readable title of the collection
    type: string
  updateFrequency:
    description: a time interval of sampling location. The unit is millisecond.
    type: number
  description:
    description: any description
```

```
    type: string
  itemType:
    description: indicator about the type of the items in the moving features
collection (the default value is ‘movingfeature’).
    type: string
    default: “movingfeature”
```

Figure 2 — Collection Request Body Schema:

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.

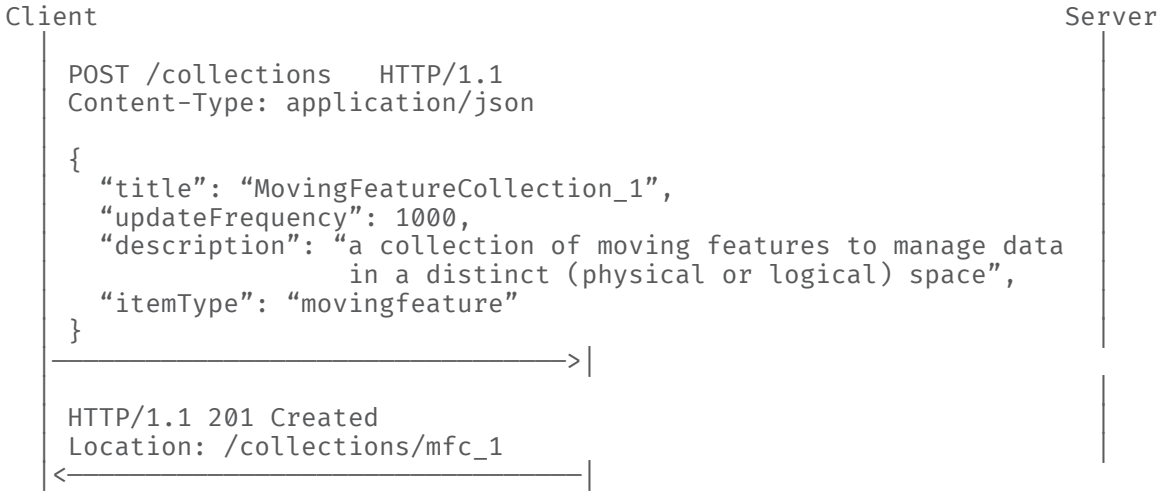

Figure 3 — An Example of Creating a New Collection:

7.3.3. Response

7.3.3.1. 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 — 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	
IDENTIFIE	`/req/mf-collection/collections-get-success`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`

REQUIREMENT 3	
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 — Common item Type section.

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

Figure 4 — Collections GET Response Schema (collections.yaml):

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

```
{
  “collections”: [
    {
      “id”: “mfc-1”,
      “title”: “MovingFeatureCollection_1”,
      “description”: “a collection of moving features to manage data in a
distinct (physical or logical) space”,
      “itemType”: “movingfeature”,
      “updateFrequency”: 1000,
      “extent”: {
        “spatial”: {
          “bbox”: [
            -180, -90, 190, 90
          ],
          “crs”: “http://www.opengis.net/def/crs/OGC/1.3/CRS84”
        },
        “temporal”: {
          “interval”: [
            “2011-11-11T12:22:11Z”,”2012-11-24T12:32:43Z”
          ],
          “trs”: “http://www.opengis.net/def/uom/ISO-8601/0/Gregorian”
        }
```

```
      },
      “links”: [
        {
          “href”: “https://data.example.org/collections/mfc-1”,
          “rel”: “self”,
          “type”: “application/json”
        }
      ]
    }
  ],
  “links”: [
    {
      “href”: “https://data.example.org/collections”,
      “rel”: “self”,
      “type”: “application/json”
    }
  ]
}
```

Figure 5 — An Example of Collections JSON Payload:

7.3.3.1.1. Create

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

REQUIREMENT 4	
IDENTIFIE	`/req/mf-collection/collections-post-success`
INCLUDEI IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Features 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 Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

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 — Common and OGC API — Features.

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

Table 6 — Table of collection properties

PROPERTY	REQUIREM	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 1: The *id*, *title*, *description*, *links*, *extent*, and *itemsType* properties were inherited from OGC API — Common and OGC API — Features.

NOTE 2: 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	
IDENTIFIEI	`/req/mf-collection/mandatory-collection`

REQUIREMENT 5	
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	A collection object SHALL contain all the mandatory properties listed in Table 6.

7.4.2. Operation

7.4.2.1. Retrieve

The retrieve operation is defined in the OGC API — 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 — Common.

REQUIREMENT 6	
IDENTIFIE	`/req/mf-collection/collection-get`
INCLUDEI IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	An implementation of OGC API — MF SHALL comply with the OGC API — Common **Collection** operation requirement `http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections/src-md-op`.

7.4.2.2. Replace

The replace operation is defined in the `REPLACE` conformance class of OGC API — 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 — Features.

REQUIREMENT 7	
IDENTIFIE	/req/mf-collection/collection-put
INCLUDEI IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature PUT operation requirement /req/create-replace-delete/update-put-op.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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.

Figure 6 — An Example of Replacing an Existing Collection:

7.4.2.3. Delete

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

REQUIREMENT 8	
IDENTIFIE	/req/mf-collection/collection-delete
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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.

Figure 7 — An Example of Deleting an Existing Collection:

7.4.3. Response

7.4.3.1. 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	
IDENTIFIE	/req/mf-collection/collection-get-success
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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.
D	The itemType property of the response schema SHALL be 'movingfeature'.

```
type: object
required:
  - id
  - links
  - itemType
properties:
  id:
    description: identifier of the collection used, for example, in URIs
    type: string
    example: 'address'
  title:
    description: human readable title of the collection
    type: string
    example: 'address'
  description:
    description: a description of the features in the collection
    type: string
    example: 'An address.'
  links:
    type: array
    items:
      $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/link.yaml'
    example:
      - href: https://data.example.com/buildings
        rel: item
      - href: https://example.com/concepts/buildings.html
        rel: describedby
        type: text/html
  extent:
    $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/extent.yaml'
  itemType:
    description: indicator about the type of the items in the collection
    type: string
    default: 'movingfeature'
  crs:
    description: the list of coordinate reference systems supported by the
service
    type: array
    items:
      type: string
    default:
      - 'https://www.opengis.net/def/crs/OGC/1.3/CRS84'
    example:
      - 'https://www.opengis.net/def/crs/OGC/1.3/CRS84'
      - 'https://www.opengis.net/def/crs/EPSG/0/4326'
  updateFrequency:
    description: a time interval of sampling location. The unit is millisecond.
    type: number
```

Figure 8 — Collection GET Response Schema (collection.yaml)

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

```
{
  "id": "mfc-1",
  "title": "moving_feature_collection_sample",
```

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


Figure 9 — An Example of Collection GET Operation:

7.4.3.2. Replace

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

REQUIREMENT 10	
IDENTIFIE	/req/mf-collection/collection-put-success
INCLUDEI IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature PUT response requirement /req/create-replace-delete/update-put-response.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature PUT exception requirement /req/create-replace-delete/update-put-rid-exception.

7.4.3.3. Delete

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

REQUIREMENT 11	
IDENTIFIE	/req/mf-collection/collection-delete-success
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

8

REQUIREMENTS CLASS "MOVING FEATURES"

8 REQUIREMENTS CLASS "MOVING FEATURES"

8.1. Overview

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'MOVING FEATURES'	
IDENTIFIER	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/movingfeatures
PREREQUISITES	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

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'MOVING FEATURES'	
	http://www.opengis.net/spec/movingfeatures/json/1.0/req/prism
NORMATIVE STATEMENTS	Requirement 12: /req/movingfeatures/param-subtrajectory-definition Requirement 13: /req/movingfeatures/param-subtrajectory-response Requirement 14: /req/movingfeatures/features-get Requirement 15: /req/movingfeatures/features-post Requirement 16: /req/movingfeatures/features-get-success Requirement 17: /req/movingfeatures/features-post-success Requirement 18: /req/movingfeatures/mf-mandatory Requirement 19: /req/movingfeatures/mf-get

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'MOVING FEATURES'
Requirement 20: /req/movingfeatures/mf-delete
Requirement 21: /req/movingfeatures/mf-get-success
Requirement 22: /req/movingfeatures/mf-delete-success
Requirement 23: /req/movingfeatures/param-leaf-definition
Requirement 24: /req/movingfeatures/param-leaf-response
Requirement 25: /req/movingfeatures/tgsequence-get
Requirement 26: /req/movingfeatures/tgsequence-post
Requirement 27: /req/movingfeatures/tgsequence-get-success
Requirement 28: /req/movingfeatures/tgsequence-post-success

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'MOVING FEATURES'
Requirement 29: /req/movingfeatures/tpgeometry-mandatory
Requirement 30: /req/movingfeatures/tpgeometry-delete
Requirement 31: /req/movingfeatures/tpgeometry-delete-success
Requirement 32: /req/movingfeatures/tpgeometry-query
Requirement 33: /req/movingfeatures/tpgeometry-query-success
Requirement 34: /req/movingfeatures/param-subtemporalvalue-definition
Requirement 35: /req/movingfeatures/param-subtemporalvalue-response
Requirement 36: /req/movingfeatures/tproperties-get
Requirement 37: /req/

REQUIREMENTS CLASS 2: REQUIREMENTS CLASS 'MOVING FEATURES'

movingfeatures/tproperties-post

Requirement 38: /req/movingfeatures/tproperties-get-success

Requirement 39: /req/movingfeatures/tproperties-post-success

Requirement 40: /req/movingfeatures/tproperty-mandatory

Requirement 41: /req/movingfeatures/tproperty-get

Requirement 42: /req/movingfeatures/tproperty-post

Requirement 43: /req/movingfeatures/tproperty-get-success

Requirement 44: /req/movingfeatures/tproperty-post-success

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	+{collectionId}+/ items/ +{mfeatureId} {root}+/collections/ +{collectionId}+/items/ +{mfeatureId}	GET, DELETE
TemporalGeometry	+{root}+/collections/ +{collectionId}+/items/ +{mFeatureId}+/tgsequence	GET, POST
TemporalPrimitiveGeometry	+{collectionId}+/ items/ +{mFeatureId}+/ tgsequence/ +{tGeometryId} {root}+/collections/ +{collectionId}+/ items/ +{mFeatureId}+/ tgsequence/ +{tGeometryId}	DELETE
TemporalGeometry Query	+{collectionId}+/ items/ +{mFeatureId}+/ tgsequence/ +{tGeometryId}+/ +{queryType}{root}+/ collections/ +{collectionId}+/ items/ +{mFeatureId}+/ tgsequence/ +{tGeometryId}+/ +{queryType}	GET
TemporalProperties	+{root}+/collections/ +{collectionId}+/ items/ +{mFeatureId}+/ tproperties	GET, POST

RESOURCE	URI	HTTP METHOD
TemporalProperty	+{collectionId}+/ items/ +{mFeatureId}+/ tproperties/ +{tPropertiesName} {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 — MF `Items` query is an OGC API — 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 — Features and OGC API — Common, except `subTrajectory` parameter.

8.3.2. Query Parameters

Query parameters are used in URLs to define the resources which are returned on a GET request.

The query parameters `bbox`, `datetime`, and `limit` are inherited from OGC API — Common.

8.3.2.1. Parameter subTrajectory

The `subTrajectory` query parameter is defined as follows:

REQUIREMENT 12	
IDENTIFIE	`/req/movingfeatures/param-subtrajectory-definition`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

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:`
	`  type: boolean`
	`style: form`
	`explode: false`

B	The `subTrajectory` parameter SHALL be used with `datetime` parameter.
C	If the `subTrajecotry` parameter is **"true"**, the `datetime` parameter SHALL be a bounded interval, not half-bounded intervals or a date-time.

REQUIREMENT 13	
IDENTIFIE	`/req/movingfeatures/param-subtrajectory-response`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The endpoint SHALL return only a subsequence of the trajectory derived from temporal primitive geometry by the ***subTrajectory*** operation at a time interval (new start time and new end time) included in the `datetime` parameter, using interpolated trajectory according to the `interpolation` property in **TemporalPrimitiveGeometry**, if the `subTrajectory` parameter is "true".
B	If the `subTrajectory` parameter is "true", the `datetime` parameter SHALL match all temporal primitive geometry objects in the moving feature or moving feature collection.
C	If the `subTrajectory` parameter is "true", the `interpolation` property in the response SHALL be the same as the temporal primitive geometry's `interpolation` property value.
D	Apply `subTrajectory` only to resources that intersect a `bbox` parameter, if the `subTrajectory` parameter is provided with a `bbox` parameter.
E	The `subTrajectory` parameter SHALL not be used with the `leaf` parameter.

The `subTrajectory` query parameter is used to select a subsequence of **TemporalGeometrySequence** for the specified time interval. Each **MovingFeature** in the **MovingFeatures** has **TemporalGeometrySequence**. The `subTrajectory` parameter is used to implement the ***subTrajectory*** operation, which is defined in the OGC Moving Feature Access Standard. This operation required two timestamps (*newStartTime* and *newEndTime*) to represent a specified time interval. The time interval for the subTrajectory operation is taken from `datetime` parameter.

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 operation at time (*newStartTime* and *newEndTime*) included in the `subTrajectory` parameter, using interpolated trajectory according to the `interpolation` property in **TemporalPrimitiveGeometry**. The `interpolation` property in the response shall be the same as the original temporal primitive geometry.

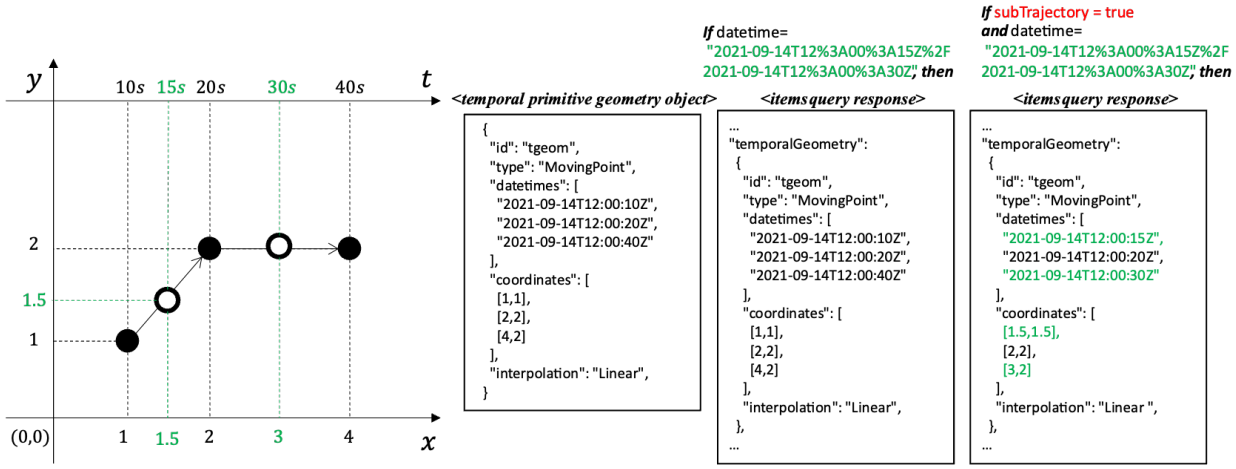

Figure 11 — Example of a response result with `subTrajectory` parameter

8.3.3. Operation

8.3.3.1. Retrieve

The retrieve operation is defined in the `Features` conformance class of OGC API — 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 — Features.

REQUIREMENT 14	
IDENTIFIE	/req/movingfeatures/features-get
INCLUDED IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Features Features operation requirement /req/core/fc-op.

8.3.3.1.1. Create

The create operation is defined in the CREATE conformance class of OGC API — 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 — Features.

REQUIREMENT 15	
IDENTIFIE	/req/movingfeatures/features-post
INCLUDED IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 ***MovingFeature Collection*** object in OGC Moving Features JSON Encoding Standard.

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.

```
   Client
Server
     |
 |
     | POST /collections/mfc_1/items   HTTP/1.1
 |
```

```
| Content-Type: application/geo+json
|
|
| {
|   "type": "Feature",
|   "id": "mf_1",
|   "properties": {
|     "name": "car1",
|     "state": "test1",
|     "video": "http://.../example/video.mpeg"
|   },
|   "crs": {
|     "type": "Name",
|     "properties": {
|       "name": "urn:ogc:def:crs:OGC:1.3:CRS84"
|     }
|   },
|   "trs": {
|     "type": "Link",
|     "properties": {
|       "type": "ogcdef",
|       "href": "http://www.opengis.net/def/uom/ISO-8601/0/Gregorian"
|     }
|   },
|   "temporalGeometry": {
|     "type": "MovingPoint",
|     "datetimes": [
|       "2011-07-14T22:01:01.000Z",
|       "2011-07-14T22:01:02.000Z",
|       "2011-07-14T22:01:03.000Z",
|       "2011-07-14T22:01:04.000Z",
|       "2011-07-14T22:01:05.000Z"
|
```

```
      ],
      "coordinates": [
         [139.757083,35.627701,0.5],
         [139.757399,35.627701,2.0],
         [139.757555,35.627688,4.0],
         [139.757651,35.627596,4.0],
         [139.757716,35.627483,4.0]
      ],
      "interpolation": "Linear",
      "base": {
         "type": "glTF",
         "href": "http://.../example/car3dmodel.gltf"
      },
      "orientations": [
          {"scales": [1,1,1],"angles": [0,0,0]},
          {"scales": [1,1,1],"angles": [0,355,0]},
          {"scales": [1,1,1],"angles": [0,0,330]},
          {"scales": [1,1,1],"angles": [0,0,300]},
          {"scales": [1,1,1],"angles": [0,0,270]},
      ]
   },
   "temporalProperties": [
      {
         "datetimes": [
            "2011-07-14T22:01:01.450Z",
            "2011-07-14T23:01:01.450Z",
            "2011-07-15T00:01:01.450Z"
            ],
         "length": {
            "type": "Measure",
            "form": "http://www.qudt.org/qudt/owl/1.0.0/quantity/
Length",
```

```
            "values": [1,2.4,1],
            "interpolation": "Linear",
            "description": "description1"
         },
         "discharge": {
            "type": "Measure",
            "form": "MQS",
            "values": [3,4,5],
            "interpolation": "Step"
         }
      },
      {
         "datetimes": [
            "2011-07-15T23:01:01.450Z",
            "2011-07-16T00:01:01.450Z"
         ],
         "camera": {
            "type": "Image",
            "values": [
               "http://.../example/image1",
               "VBORw0KGgoAAAANSUhEU......"
            ],
            "interpolation": "Discrete"
         },
         "labels": {
            "type": "Text",
            "values": ["car","human"],
            "interpolation": "Discrete"
         }
      }
   ]
```

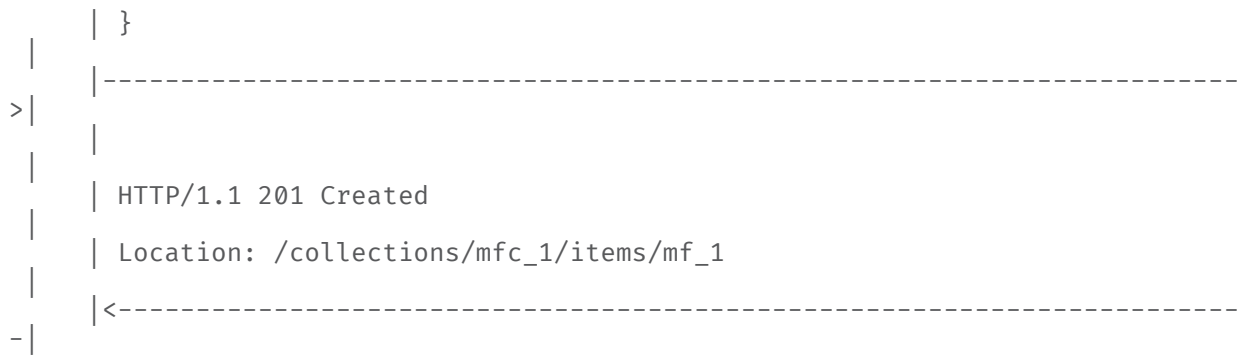

Figure 12 — An Example of Creating a New MovingFeature Object:

8.3.4. Response

8.3.4.1. 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 — MF, the **MovingFeatures** GET response will list **MovingFeature** resource.

REQUIREMENT 16	
IDENTIFIE	`/req/movingfeatures/features-get-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Features `Features` response requirement `/req/core/fc-response`.
B	The response SHALL only include moving features selected by the request with `limit`, `bbox`, `datetime`, and `subTrajectory` parameters.
C	Each moving feature in the response SHALL include the mandatory properties listed in Table 8.

```
type: object
required:
  - type
  - features
properties:
  type:
```

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

Figure 13 — MovingFeatures GET Response Schema (movingFeatureCollection.yaml):

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

```
{
  "type": "FeatureCollection",
  "features":[
    {
      "id": "mf-1",
      "type": "Feature",
      "geometry":{
        "type": "LineString",
        "coordinates": [
          [139.757083, 35.627701, 0.5],
          [139.757399, 35.627701, 2.0],
          [139.757555, 35.627688, 4.0],
          [139.757651, 35.627596, 4.0],
          [139.757716, 35.627483, 4.0]
        ]
      },
      "properties":{
        "label": "car",
        "state": "test1",
        "video": "http://www.opengis.net/spec/movingfeatures/json/1.0/prism/
example/video.mpeg"
      },
      "bbox":[
        139.757083, 35.627483, 0.0,
        139.757716, 35.627701, 4.5
```

```
      ],
      "time":[
        "2011-07-14T22:01:01Z",
        "2011-07-15T01:11:22Z"
      ],
      "crs": {
        "type": "Name",
        "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
      },
      "trs": {
        "type": "Name",
        "properties": "urn:ogc:data:time:iso8601"
      }
    }
  ],
  "crs": {
    "type": "Name",
    "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
  },
  "trs": {
    "type": "Name",
    "properties": "urn:ogc:data:time:iso8601"
  },
  "links":[
    {
      "href": "https://data.example.org/collections/mfc-1/items",
      "rel": "self",
      "type": "application/geo+json"
    },
    {
      "href": "https://data.example.org/collections/mfc-1/items&offset=1&limit=
1",
      "rel": "next",
      "type": "application/geo+json"
    }
  ],
  "timeStamp": "2020-01-01T12:00:00Z",
  "numberMatched": 100,
  "numberReturned": 1
}
```


Figure 14 — An Example of a MovingFeatures GET Operation:

8.3.4.1.1. Create

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

REQUIREMENT 17	
IDENTIFIE	`/req/movingfeatures/features-post-success`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 17	
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.3.5. Error situations

The requirements for handling unsuccessful requests are provided in the Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

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.

Table 8 defines the set of properties that may be used to describe a moving feature. The schema for the moving feature object presented in this clause is an extension of the ***GeoJSON Feature Object*** defined in GeoJSON. By default, the properties defined in Table 8 are the same as the ***MovingFeature Object*** in OGC MF-JSON. The semantics of each property are also the same as those defined in MF-JSON.

However, depending on where this schema is used (i.e., depending on which resource produces results with which query parameter), there are differences in requirements from the schema defined in MF-JSON; as sometimes it only needs to have static information, and sometimes it also has a **temporalGeometry**. For example, in MF-JSON, `type` and `temporalGeometry` are mandatory, but in this API, `id` and `type` are mandatory. This is why the defined schema is represented in GeoJSON, not MF-JSON.

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

PROPERTY	REQUIREM	DESCRIPTION
id	M	A unique identifier to the moving feature.
type	M	The GeoJSON feature type (i.e., one of 'Feature' or 'FeatureCollection').
geometry	O	Projective geometry of the moving feature.

PROPERTY	REQUIREM	DESCRIPTION
properties	O	A set of properties of GeoJSON.
bbox	O	Bounding box information for the moving feature.
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 1: The properties *id*, *type*, *geometry*, *properties*, and *bbox* were inherited from GeoJSON.

NOTE 2: The properties **time**, **crs**, **trs**, **temporalGeometry**, and **temporalProperties** were inherited from OGC MF-JSON

REQUIREMENT 18	
IDENTIFIEI	`/req/movingfeatures/mf-mandatory`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	A moving feature object SHALL contain all the mandatory properties listed in Table 8.

8.4.2. Operation

8.4.2.1. Retrieve

The retrieve operation is defined in the `Feature` conformance class of OGC API — 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 — Features.

REQUIREMENT 19	
IDENTIFIE	/req/movingfeatures/mf-get
INCLUDEI IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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}
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.

8.4.2.2. Delete

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

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

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

REQUIREMENT 20	
IDENTIFIE	/req/movingfeatures/mf-delete
INCLUDEI IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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}

REQUIREMENT 20	
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

8.4.3.1. 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	
IDENTIFIEI	`/req/movingfeatures/mf-get-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
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.

```
type: object
required:
  - id
  - type
properties:
  type:
    type: string
    enum:
      - 'Feature'
  temporalGeometry:
    $ref: "MF-JSON/Prism/temporalGeometry.yaml"
  temporalProperties:
    $ref: "MF-JSON/Prism/temporalProperties.yaml"
  crs:
    $ref: "MF-JSON/Prism/crs.yaml"
  trs:
    $ref: "MF-JSON/Prism/trs.yaml"
  bbox:
    $ref: "MF-JSON/Prism/bbox.yaml"
  time:
    $ref: "MF-JSON/Prism/lifeSpan.yaml"
```

```
  geometry:
    $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schema
s/geometryGeoJSON.yaml'
  properties:
    type: object
    nullable: true
  id:
    description: 'An identifier for the feature'
    oneOf:
      - type: string
      - type: integer
  links:
    type: array
    items:
      $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/link.yaml'
```

Figure 15 — MovingFeature GET Response Schema (movingFeature.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 — Moving Features **MovingFeature** operation.

```
{
  "id": "mf-1",
  "type": "Feature",
  "geometry":{
    "type": "LineString",
    "coordinates": [
      [139.757083, 35.627701, 0.5],
      [139.757399, 35.627701, 2.0],
      [139.757555, 35.627688, 4.0],
      [139.757651, 35.627596, 4.0],
      [139.757716, 35.627483, 4.0]
    ]
  },
  "properties":{
    "name": "car1",
    "state": "test1",
    "video": "http://www.opengis.net/spec/movingfeatures/json/1.0/prism/
example/video.mpeg"
  },
  "bbox":[
    139.757083, 35.627483, 0.0,
    139.757716, 35.627701, 4.5
  ],
  "time":[
    "2011-07-14T22:01:01Z",
    "2011-07-15T01:11:22Z"
  ],
  "crs": {
    "type": "Name",
    "properties": "urn:ogc:def:crs:OGC:1.3:CRS84"
  },
  "trs": {
    "type": "Name",
    "properties": "urn:ogc:data:time:iso8601"
  }
```

}

Figure 16 — An Example of a MovingFeature JSON Payload:

8.4.3.2. Delete

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

REQUIREMENT 22	
IDENTIFIE	`/req/movingfeatures/mf-delete-success`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

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 sequence of **TemporalPrimitiveGeometry** object which is included in the **MovingFeature** that specified by `{mFeatureId}`. The sequence of **TemporalPrimitiveGeometry** object returned to the response can be limited using the query parameters `limit`, `bbox`, `datetime`, and `leaf` (or `subTrajectory`).
2. A create operation posts a new **TemporalPrimitiveGeometry** resource to the **MovingFeature** that specified by {mFeatureId}.

8.5.2. Query Parameters

Query parameters are used in URLs to define the resources which are returned on a GET request.

The query parameters `bbox`, `datetime`, and `limit` are inherited from OGC API — Common.

The `subTrajectory` query parameter is defined in the **MovingFeatures** clause.

8.5.2.1. Parameter leaf

The `leaf` query parameter is defined as follows:

REQUIREMENT 23	
IDENTIFIER	`/req/movingfeatures/param-leaf-definition`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

A	The operation SHALL support a query parameter `leaf` with the following characteristics (using an Open API Specification 3.0 fragment):
	`name: leaf`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  uniqueItems: true`
	`  minItems: 1`
	`  items:`
	`  type: string`
	`  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	
IDENTIFIER	`/req/movingfeatures/param-leaf-response`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 24	
A	The `leaf` parameter SHALL match all resources in the moving feature that are associated with temporal information.
B	If the `leaf` parameter is provided by the client, the endpoint SHALL return only temporal geometry coordinate (or temporal property value) with the ***pointAtTime*** operation at each date-time included in the `leaf` parameter, using interpolated trajectory according to the `interpolation` property.
C	If the `leaf` parameter is provided by the client, the `interpolation` property in the response SHALL be 'Discrete'.
D	Apply `leaf` only to resources that intersect a `bbox` or (and) a `datetime` parameter, if the `leaf` parameter is provided with a `bbox` or (and) a `datetime` parameter.
E	The `leaf` parameter SHALL not be used with the `subTrajectory` parameter.

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. Figure 18 shows valid expression examples of the `leaf` parameter.

```
(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"
```

Figure 18 — leaf parameter valid (and invalid) Examples

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. The `interpolation` property in the response shall be "Discrete".

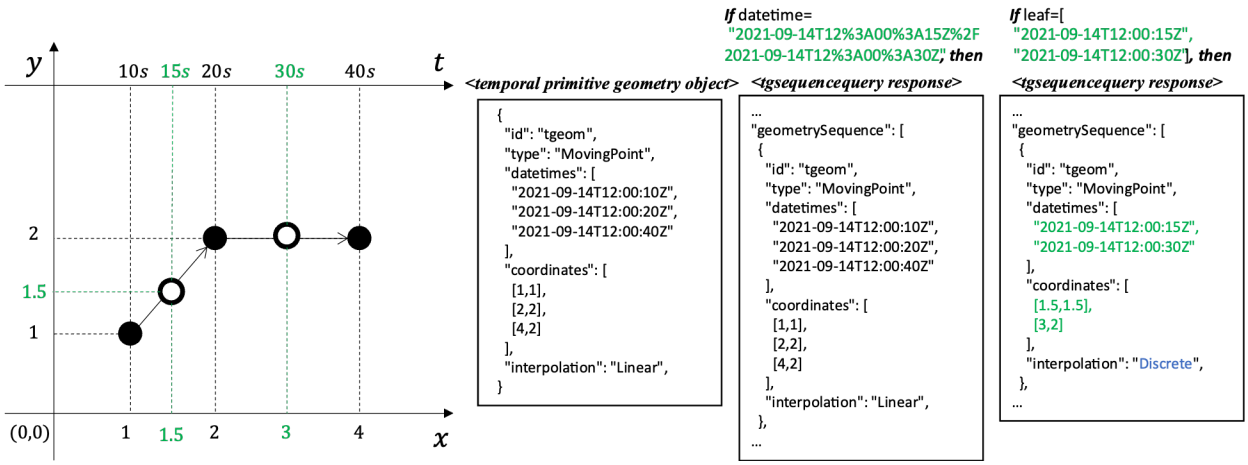

Figure 19 — Example of a response result with `leaf` parameter

8.5.3. Operation

8.5.3.1. Retrieve

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

REQUIREMENT 25	
IDENTIFI	`/req/movingfeatures/tgsequence-get`
INCLUDE IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
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.

8.5.3.1.1. Create

The create operation is defined in the `CREATE` conformance class of OGC API — 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 — Features.

REQUIREMENT 26	
IDENTIFIE	`/req/movingfeatures/tgsequence-post`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature `CREATE` operation requirement `/req/create-replace-delete/insert-post-op`.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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.

```
   Client
Server
    |
 |
    | POST /collections/mfc_1/items/mf_1/tgsequence  HTTP/1.1
 |
    | Content-Type: application/json
 |
    |
 |
    | {
 |
    |    "id": "tg_1",
 |
    |    "type": "MovingPoint",
 |
    |    "datetimes": [
 |
    |       "2011-07-14T22:01:06.000Z",
 |
```

```
    |       "2011-07-14T22:01:07.000Z",
    |       "2011-07-14T22:01:08.000Z"
    |    ],
    |    "coordinates": [
    |       [139.757716,35.627483,4.0],
    |       [139.757782,35.627483,4.0],
    |       [139.757843,35.627483,4.0]
    |    ],
    |    "interpolation": "Linear",
    |    "base": {
    |       "type": "glTF",
    |       "href": "http://.../example/car3dmodel.gltf"
    |    },
    |    "orientations": [
    |        {"scales": [1,1,1],"angles": [0,0,270]},
    |        {"scales": [1,1,1],"angles": [0,0,270]},
    |        {"scales": [1,1,1],"angles": [0,0,270]}
    |    ]
    | }
    |-------------------------------------------------------------------------
>   |
    | HTTP/1.1 201 Created
    | Location: /collections/mfc_1/items/mf_1/tgsequence/tg_1
    |<-------------------------------------------------------------------------
-   |
```

Figure 20 — An Example of Creating a New TemporalPrimitiveGeometry Object:

8.5.4. Response

8.5.4.1. 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	
IDENTIFIE	`/req/movingfeatures/tgsequence-get-success`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature `Features` response requirement `/req/core/fc-response`.
B	The response SHALL only includes temporal primitive geometry selected by request with `limit`, `bbox`, `datetime`, and `leaf` (or `subTrajectory`) parameters.
C	Each temporal primitive geometry in the response SHALL include the mandatory properties listed in Table 9.

```
type: object
required:
  – type
  – geometrySequence
properties:
  type:
    type: string
    enum:
      – ”TemporalGeometrySequence”
  geometrySequence:
    type: array
    items:
      $ref: ‘MF-JSON/Prism/temporalPrimitiveGeometry.yaml’
  links:
    type: array
    items:
      $ref: ‘https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/link.yaml’
  timeStamp:
    type: string
    format: date-time
  numberMatched:
    type: integer
    minimum: 0
  numberReturned:
    type: integer
```

```
    minimum: 0
```

Figure 21 — TemporalGeometrySequence GET Response Schema (temporalGeometrySequence.yaml):

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


```
{
  “type”: “TemporalGeometrySequence”,
  “geometrySequence”: [
    {
      “id”: “tg-1”,
      “type”: “MovingPoint”,
      “datetimes”: [
        “2011-07-14T22:01:02Z”,
        “2011-07-14T22:01:03Z”,
        “2011-07-14T22:01:04Z”
      ],
      “coordinates”: [
        [139.757399, 35.627701, 2.0],
        [139.757555, 35.627688, 4.0],
        [139.757651, 35.627596, 4.0]
      ],
      “interpolation”: “Linear”,
      “base”: {
        “type”: “glTF”,
        “href”: “https://www.opengis.net/spec/movingfeatures/json/1.0/prism/
example/car3dmodel.gltf”
      },
      “orientations”:[
        {
          “scales”: [1,1,1],
          “angles”: [0,355,0]
        },
        {
          “scales”: [1,1,1],
          “angles”: [0,0,330]
        },
        {
          “scales”: [1,1,1],
          “angles”: [0,0,300]
        }
      ],
      “crs”: {
        “type”: “Name”,
        “properties”: “urn:ogc:def:crs:OGC:1.3:CRS84”
      },
      “trs”: {
        “type”: “Name”,
        “properties”: “urn:ogc:data:time:iso8601”
      }
    }
  ],
  “links”: [
    {
      “href”: “https://data.example.org/collections/mfc-1/items/mf-1/
tgeometries”,
      “rel”: “self”,
      “type”: “application/json”
```

```
    },
    {
      “href”: “https://data.example.org/collections/mfc-1/items/mf-1/
tgeometries&offset=10&limit=1”,
      “rel”: “next”,
      “type”: “application/json”
    }
  ],
  “timeStamp”: “2021-09-01T12:00:00Z”,
  “numberMatched”: 100,
  “numberReturned”: 1
}
```


Figure 22 — An Example of a TemporalGeometrySequence GET operation:

8.5.4.1.1. Create

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

REQUIREMENT 28	
IDENTIFIE	/req/movingfeatures/tgsequence-post-success
INCLUDEI IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

8.6. Resource TemporalPrimitiveGeometry

8.6.1. Overview

A **TemporalPrimitiveGeometry** resource 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 the same as the ***TemporalPrimitiveGeometry*** object defined in MF-JSON. 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 primitive geometry.
type	M	A primitive geometry type of MF-JSON (i.e., one of 'MovingPoint', 'Moving LineString', '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	
IDENTIFIER	`/req/movingfeatures/tpgeometry-mandatory`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	TemporalPrimitiveGeometry object SHALL contain all the mandatory properties listed in Table 9.

8.6.2. Operation

8.6.2.1. Delete

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

REQUIREMENT 30	
IDENTIFIER	/req/movingfeatures/tpgeometry-delete
INCLUDE IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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

8.6.3.1. Delete

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

REQUIREMENT 31	
IDENTIFIE	/req/movingfeatures/tpgeometry-delete-success
INCLUDEI IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

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 — 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 — MF.

Table 10 provides a mapping of the initial query types proposed for the OGC API — 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. Query parameters

Query parameters are used in URLs to define the resources which are returned on a GET request. The query parameters `datetime` is inherited from OGC API — Common.

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 23 shows an example of the time-to-distance curve.

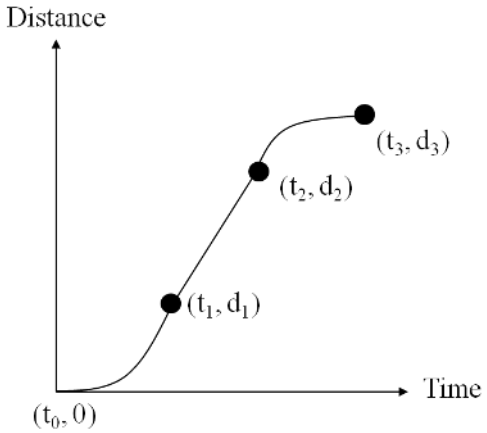

Figure 23 — Example of time-to-distance curve [OGC 16-120r3, OGC Moving Features Access]

The filter constraints are defined by the following query parameter:

8.7.3.1. Parameter datetime

The `datetime` parameter defines the specified date and time to return the distance value from the time-to-distance curve. If the `datetime` is provided, the `datetime` must be a date-time, not a half-bounded interval or bounded interval. When the `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:

8.7.4.1. Parameter datetime

The `datetime` parameter defines the specified date and time to return the velocity value from the time-to-velocity graph. If the `datetime` is provided, the `datetime` must be a date-time, not a half-bounded interval or bounded interval. When the `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:

8.7.5.1. Parameter datetime

The `datetime` parameter defines the specified date and time to return the acceleration value from the time-to-acceleration curve. If the `datetime` is provided, the `datetime` must be a date-time, not a half-bounded interval or bounded interval. When the `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	
IDENTIFI	/req/movingfeatures/tpgeometry-query
INCLUDE IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
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**)

Table 11

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.
D	If the datetime parameter is provided, the datetime SHALL be a date-time, not a half-bounded interval or bounded interval.

8.7.7. Response Requirements

REQUIREMENT 33	
IDENTIFIE	/req/movingfeatures/tpgeometry-query-success
INCLUDED IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
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.

REQUIREMENT 33	
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. The **TemporalProperties** resource returned to the response can be limited using the query parameters `limit`, `datetime`, and `subTemporalValue`.
2. A create operation posts a new **TemporalProperty** object to the **MovingFeature** that is specified by {mFeatureId}.

8.8.2. Query Parameters

Query parameters are used in URLs to define the resources which are returned on a GET request.

The query parameters `datetime` and `limit` are inherited from OGC API — Common.

8.8.2.1. Parameter subTemporalValue

The `subTemporalValue` query parameter is defined as follows:

REQUIREMENT 34	
IDENTIFIE	`/req/movingfeatures/param-subtemporalvalue-definition`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The operation SHALL support a query parameter `subTemporalValue` with the following characteristics (using an OpenAPI Specification 3.0 fragment): `name: subTemporalValue`

REQUIREMENT 34

```
in: query
required: false
schema:
  type: boolean
style: form
explode: false
```

B	The `subTemporalValue` parameter SHALL be used with `datetime` parameter.
C	If the `subTemporalValue` parameter is “true”, the `datetime` parameter SHALL be a bounded interval, not half-bounded intervals or a date-time.

REQUIREMENT 35

IDENTIFIE	`/req/movingfeatures/param-subtemporalvalue-response`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The endpoint SHALL return only a subsequence of the temporal primitive values derived from temporal property for specified a time interval (new start time and new end time) included in the `datetime` parameter, using interpolated temporal property values according to the `interpolation` property in **TemporalProperty**, if the `subTemporalValue` parameter is “ture”.
B	If the `subTemporalValue` parameter is “true”, the `datetime` parameter SHALL match all temporal property objects in the moving feature.
C	If the `subTemporalValue` parameter is “true”, the `interpolation` property in the response SHALL be the same as the temporal property’s `interpolation` property value.
D	The `subTemporalValue` parameter SHALL not be used with the `leaf` parameter.

The `subTemporalValue` query parameter is used to select a subset of **TemporalProperty** for the specified time interval. Each **TemporalProperty** in the **TemporalProperties** has a sequence of **TemporalPrimitiveValue**. The `subTemporalValue` parameter behaves functionally the same as the `subTrajectory` parameter. The difference is that `subTrajectory` is associated with temporal geometry, while `subTemporalValue` is associated with temporal properties.

8.8.3. Operation

8.8.3.1. Retrieve

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

REQUIREMENT 36	
IDENTIFI	/req/movingfeatures/tproperties-get
INCLUDE IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
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.

8.8.3.1.1. Create

The create operation is defined in the CREATE conformance class in the OGC API — 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 — Features.

REQUIREMENT 37	
IDENTIFIE	/req/movingfeatures/tproperties-post
INCLUDE IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature CREATE operation requirement /req/create-replace-delete/insert-post-op.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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 a TemporalProperty defined in this API or a ParametricValues object defined in OGC Moving Features JSON Encoding Standard.

```
type: object
required:
  - name
  - type
properties:
  name:
```

```
    type: string
  type:
    type: string
    enum:
      – 'TBoolean'
      – 'TText'
      – 'TInteger'
      – 'TReal'
      – 'TImage'
  form:
    oneOf:
      – type: string
        format: uri
      – type: string
        minLength: 3
        maxLength: 3
  valueSequence:
    type: array
    uniqueItems: true
    items:
      $ref: 'temporalPrimitiveValue.yaml'
  description:
    type: string
  links:
    type: array
    items:
      $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/link.yaml'
```

Figure 25 — TemporalProperties Request Body Schema (temporalProperty.yaml):

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.

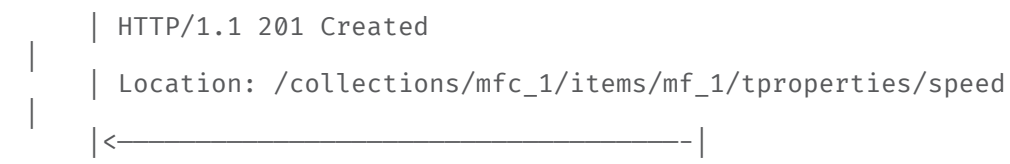

Figure 26 — An Example of Creating a New TemporalProperty Object:

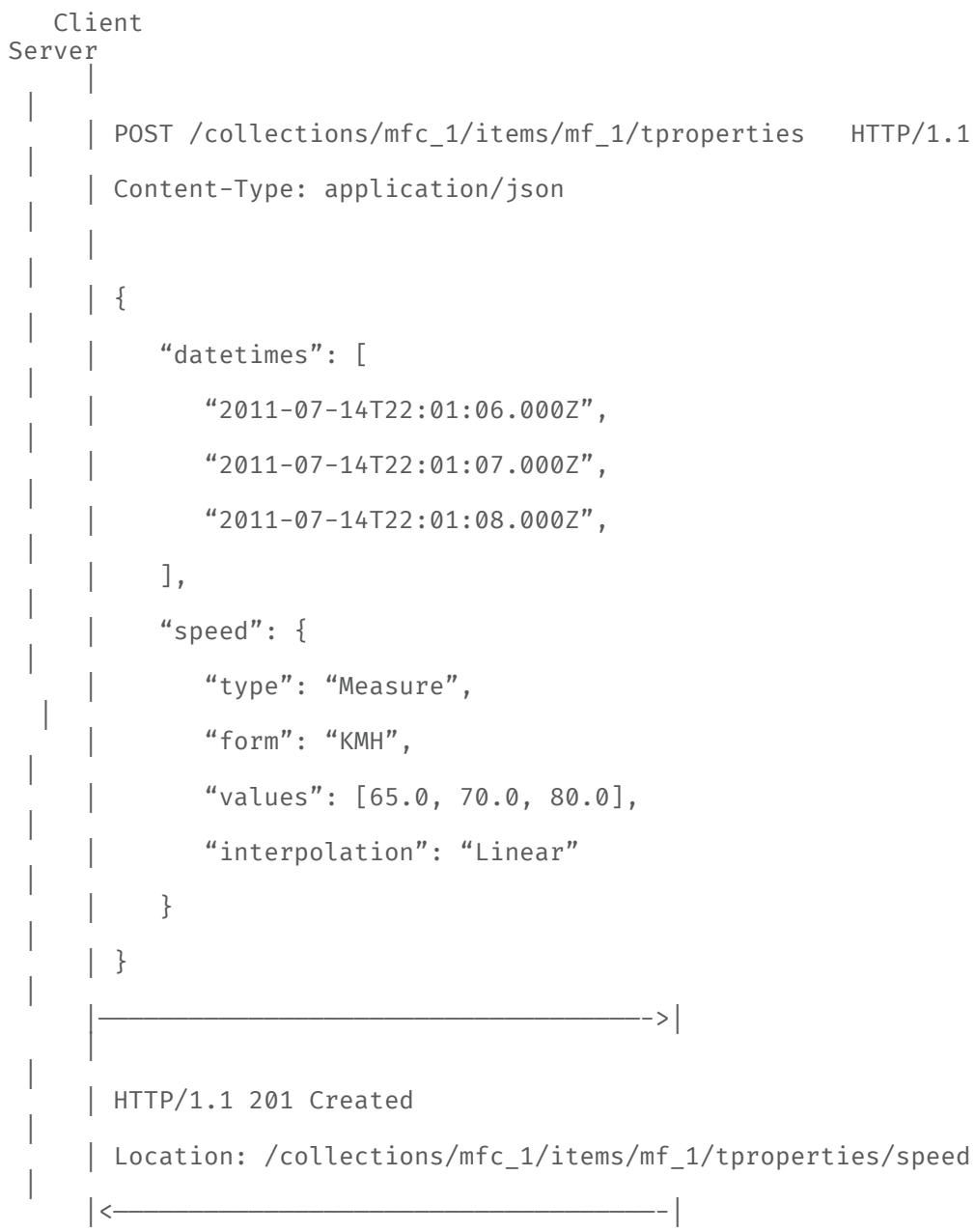

Figure 27 — An Example of Creating a New TemporalProperty Object with *ParametricValues* as a MF-JSON encoding:

8.8.4. Response

8.8.4.1. Retrieve

A successful response to the **TemporalProperties** GET operation is a document that contains the set of metadata (and static data) of **TemporalProperty** in the moving feature identified by the `{mFeatureId}` parameter. The response result does not include dynamic (and temporal information).

If the value of the `subTemporalValue` query parameter is provided, the temporal value of the corresponding `temporalProperties` property of the moving feature is calculated using the `subTemporalValue` parameter value and included in the result.

REQUIREMENT 38	
IDENTIFIE	`/req/movingfeatures/tproperties-get-success`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature `Features` response requirement `/req/core/fc-response`.
B	The response SHALL only include moving features selected by the request with `limit`, `datetime`, and `subTemporalValue` parameters.
C	Each temporal property object in the response SHALL include the mandatory properties listed in Table 12.

```
type: object
required:
  – temporalProperties
properties:
  temporalProperties:
    oneOf:
      – $ref: “MF-JSON/Prism/temporalProperties.yaml”
      – type: array
        items:
          $ref: “temporalProperty.yaml”
  links:
    type: array
    items:
      $ref: ‘https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/
schemas/link.yaml’
  timeStamp:
    type: string
    format: date-time
  numberMatched:
    type: integer
    minimum: 0
```

```
  numberReturned:
    type: integer
    minimum: 0
```

Figure 28 — TemporalProperties GET Response Schema (TemporalProperties.yaml):

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

```
{
  “temporalProperties”: [
    {
      “name”: “length”,
      “type”: “TReal”,
      “form”: “http://www.qudt.org/qudt/owl/1.0.0/quantity/Length”
    },
    {
      “name”: “speed”,
      “type”: “TReal”,
      “form”: “KHM”
    }
  ],
  “links”: [
    {
      “href”: “https://data.example.org/collections/mfc-1/items/mf-1/
tproperties”,
      “rel”: “self”,
      “type”: “application/json”
    },
    {
      “href”: “https://data.example.org/collections/mfc-1/items/mf-1/
tproperties&offset=2&limit=2”,
      “rel”: “next”,
      “type”: “application/json”
    }
  ],
  “timeStamp”: “2021-09-01T12:00:00Z”,
  “numberMatched”: 10,
  “numberReturned”: 2
}
```

Figure 29 — An Example of a TemporalProperties GET Operation:

The following JSON payload is an example of a response to an OGC API — Moving Features **TemporalProperties** GET operation with query parameter `subTemporalValue`.

```
{
  “temporalProperties”: [
    {
      “datetimes”: [“2011-07-14T22:01:01.450Z”, “2011-07-14T23:01:01.450Z”,
“2011-07-15T00:01:01.450Z”],
      “length”: {
        “type”: “Measure”,
        “form”: “http://www.qudt.org/qudt/owl/1.0.0/quantity/Length”,
        “values”: [1.0, 2.4, 1.0],
        “interpolation”: “Linear”
      },
      “discharge” : {
```

```
        "type" : "Measure",
        "form" : "MQS",
        "values" : [3.0, 4.0, 5.0],
        "interpolation": "Step"
      }
    },
    {
      "datetimes" : ["2011-07-14T22:01:01.450Z", "2011-07-14T23:01:01.450Z"],
      "camera" : {
        "type" : "Image",
        "values" : ["http://www.opengis.net/spec/movingfeatures/json/1.0/prism/
example/image1", "iVBORw0KGgoAAAANSUhEU……"],
        "interpolation": "Discrete"
      },
      "labels":{
        "type": "Text",
        "values": ["car", "human"],
        "interpolation": "Discrete"
      }
    }
  ],
  "links": [
    {
      "href": "https://data.example.org/collections/mfc-1/items/mf-1/
tproperties",
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "https://data.example.org/collections/mfc-1/items/mf-1/
tproperties&offset=2&limit=2",
      "rel": "next",
      "type": "application/json"
    }
  ],
  "timeStamp": "2021-09-01T12:00:00Z",
  "numberMatched": 10,
  "numberReturned": 2
}
```


Figure 30 — An Example of a TemporalProperties GET Operation with `subTemporalValue`:

8.8.4.1.1. Create

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

REQUIREMENT 39	
IDENTIFIE	/req/movingfeatures/tproperties-post-success
INCLUDEI IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature CREATE response requirement /req/create-replace-delete/insert-response and /req/create-replace-delete/insert-response-rid.

8.8.5. Error situations

The requirements for handling unsuccessful requests are provided in the Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

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 parameters `datetime`, `leaf` and `subTemporalValue`.
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. Table 12 defines the set of property that may be used to describe a temporal property.

Table 12 — 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.

PROPERTY	REQUIREMENT	DESCRIPTION
description	O	A short description.

Table 13 — 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 40	
IDENTIFIER	`/req/movingfeatures/tproperty-mandatory`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	A temporal property object SHALL contain all the mandatory properties listed in Table 12 and Table 13.

8.9.2. Query Parameters

Query parameters are used in URLs to define the resources which are returned on a GET request.

The query parameter `datetime` is inherited from OGC API — Common.

The `leaf` query parameter is defined in the **TemporalGeometrySequence** clause.

The `subTemporalValue` query parameter is defined in the **TemporalProperties** clause.

8.9.3. Operation

8.9.3.1. 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 — MF. The list of valid values for `{tPropertyName}` is provided in the `+{root}+/collections/+{collectionId}+/items/+{mFeatureId}+/tproperties` GET response.

REQUIREMENT 41	
IDENTIFI	`/req/movingfeatures/tproperty-get`
INCLUDE IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
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/+{tPropertyName}`
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. `tPropertyName` is a local identifier of the temporal property.

8.9.3.1.1. Create

The create operation is defined in the `CREATE` conformance class in the OGC API — Features. This operation targeted the new temporal primitive value object defined in Table 13.

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 — Features.

REQUIREMENT 42	
IDENTIFIE	`/req/movingfeatures/tproperty-post`

REQUIREMENT 42	
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature `CREATE` operation requirement `/req/create-replace-delete/insert-post-op`.
B	An implementation of the OGC API — MF SHALL comply with the OGC API — 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***.

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

Figure 31 — TemporalProperty Request Body Schema (TemporalPrimitiveValue.yaml):

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.

```
   Client
Server
```

Figure 32 — An Example of Creating a New TemporalPrimitiveValue Object:

8.9.4. Response

8.9.4.1. Retrieve

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

REQUIREMENT 43	
IDENTIFIER	`/req/movingfeatures/tproperty-get-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 43	
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 `leaf` and `subTemporalValue` 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 — Moving Features **TemporalProperty** GET operation.


```
{
  "temporalProperties": [
    {
      "datetimes":[
        "2011-07-14T22:01:02Z",
        "2011-07-14T22:01:03Z",
        "2011-07-14T22:01:04Z"
      ],
      "values":[
        65.0,
        70.0,
        80.0
      ],
      "interpolation": "Linear"
    },
    {
      "datetimes":[
        "2011-07-15T08:00:00Z",
        "2011-07-15T08:00:01Z",
        "2011-07-15T08:00:02Z"
      ],
      "values":[
        0.0,
        20.0,
        50.0
      ],
      "interpolation": "Linear"
    }
  ],
  "links": [
    {
      "href": "https://data.example.org/collections/mfc-1/items/mf-1/
tproperties/speed",
      "rel": "self",
      "type": "application/json"
    }
  ]
}
```


Figure 33 — An Example of TemporalProperty GET Operation:

8.9.4.1.1. Create

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

REQUIREMENT 44	
IDENTIFIE	`/req/movingfeatures/tproperty-post-success`
INCLUDEI IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	An implementation of the OGC API — MF SHALL comply with the OGC API — Feature `CREATE` response requirement `/req/create-replace-delete/insert-response` and `/req/create-replace-delete/insert-response-rid`.

8.9.5. Error situations

The requirements for handling unsuccessful requests are provided in the Clause 9.2. General guidance on HTTP status codes and how they should be handled is provided in Clause 9.3.

9

COMMON REQUIREMENTS

9 COMMON REQUIREMENTS

9.1. Parameters

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

REQUIREMENTS CLASS 3: REQUIREMENTS CLASS 'MOVING FEATURES — COMMON'	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`
TARGET TYPE	Web API
NORMATIVE STATEMENTS	Requirement 45: `/req/common/param-limit` Requirement 46: `/req/common/param-bbox` Requirement 47: `/req/common/param-datetime` Requirement 48: `/req/common/http-response`

9.1.1. Parameter limit

REQUIREMENT 45	
IDENTIFIER	`/req/common/param-limit`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`
A	An implementation instance of the OGC API — MF SHALL support the Limit parameter for the operation.
B	Requests which include the Limit parameter SHALL comply with OGC API — Common requirement `/req/collections/rc-limit-definition`.
C	Responses to Limit requests SHALL comply with OGC API — Common requirements `/req/collections/rc-limit-response`

9.1.2. Parameter bbox

REQUIREMENT 46	
IDENTIFIE	`/req/common/param-bbox`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`
A	An implementation instance of the OGC API — MF SHALL support the Bounding Box (`bbox`) parameter for the operation.
B	Requests which include the Bounding Box parameter SHALL comply with OGC API — Common requirement `/req/collections/rc-bbox-definition`.
C	Responses to Bounding Box requests SHALL comply with OGC API — Common requirement `/req/collections/rc-bbox-response`.

9.1.3. Parameter datetime

REQUIREMENT 47	
IDENTIFIE	`/req/common/param-datetime`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`
A	An implementation instance of the OGC API — MF SHALL support the DateTime (`datetime`) parameter for the operation.
B	Requests which include the DateTime parameter SHALL comply with OGC API — Common requirement `/req/collections/rc-time-definition`.
C	Responses to DateTime requests SHALL comply with OGC API — 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 48	
IDENTIFIER	/req/common/http-response
INCLUDED IN	Requirements class 3: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common
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 Figure 34.

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

Figure 34 — An Example of the HTTP Response Schema:

9.3. HTTP Status Codes

Table 14 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 14 — 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.

STATUS CODE	DESCRIPTION
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.
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.

A

ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE)

A

ANNEX A (INFORMATIVE) 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 — 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 — 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 A.1	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/mf-collection`

CONFORMANCE CLASS A.1	
REQUIREMENTS CLASS	Requirements class 1: `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-1/1.0/conf/html http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete
CONFORMANCE TESTS	Abstract test A.1: `/conf/mf-collection/collections-get` Abstract test A.2: `/conf/mf-collection/collections-get-success` Abstract test A.3: `/conf/mf-collection/collections-post` Abstract test A.4: `/conf/mf-collection/collections-post-success` Abstract test A.5: `/conf/mf-collection/collection-get` Abstract test A.6: `/conf/mf-collection/collection-get-success` Abstract test A.7: `/conf/mf-collection/collection-put` Abstract test A.8: `/conf/mf-collection/collections-put-success` Abstract test A.9: `/conf/mf-collection/collection-delete` Abstract test A.10: `/conf/mf-collection/collections-delete-success`

A.2.1. MovingFeature Collections

A.2.1.1. HTTP GET Operation

ABSTRACT TEST A.1	
IDENTIFIER	`/conf/mf-collection/collections-get`
REQUIREMENTS	Requirement 1: `/req/mf-collection/collections-get` Requirement 3: `/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 A.2	
IDENTIFIER	`/conf/mf-collection/collections-get-success`
REQUIREMEI	Requirement 3: `/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 — Common /conf/collections/rc-md-success 2. Validate the **Collections** resource for all supported media types using the resources and tests identified in Table A.1 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 A.1 — Schema and Tests for MovingFeature Collections content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	collections.yaml	/conf/html/content
JSON	collections.yaml	/conf/json/content

A.2.1.2. HTTP POST Operation

ABSTRACT TEST A.3	
IDENTIFIER	`/conf/mf-collection/collections-post`
REQUIREMENTS	Requirement 2: `/req/mf-collection/collections-post` Requirement 4: `/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 — 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 A.2 3. Validate that the request body complies OGC API — Features POST request body requirements

ABSTRACT TEST A.3	
	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 A.2 — 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 A.4	
IDENTIFIER	`/conf/mf-collection/collections-post-success`
REQUIREMEI	Requirement 4: `/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 — Features — Part 4 POST response requirements

A.2.2. MovingFeature Collection

A.2.2.1. HTTP GET Operation

ABSTRACT TEST A.5	
IDENTIFIER	`/conf/mf-collection/collection-get`
REQUIREMENTS	Requirement 6: `/req/mf-collection/collection-get` Requirement 9: `/req/mf-collection/collection-get-success`
TEST PURPOSE	Validate that the **MovingFeature Collection** can be retrieved from the expected location.

ABSTRACT TEST A.5	
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 A.6	
IDENTIFIER	`/conf/mf-collection/collection-get-success`
REQUIREMEN	Requirement 9: `/req/mf-collection/collection-get-success` Requirement 5: `/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 — Common /conf/collections/src-md-success 2. Validate the **Collection** resource for all supported media types using the resources and tests identified in Table A.3 and Table 6

Table A.3 — Schema and Tests for MovingFeature Collection content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	collection.yaml	/conf/html/content
JSON	collection.yaml	/conf/json/content

A.2.2.2. HTTP PUT Operation

ABSTRACT TEST A.7	
IDENTIFIER	`/conf/mf-collection/collection-put`
REQUIREMENTS	Requirement 7: `/req/mf-collection/collection-put` Requirement 10: `/req/mf-collection/collection-put-success`
TEST PURPOSE	Validate that the **MovingFeature Collection** can be replaced at the expected location.

ABSTRACT TEST A.7	
TEST METHOD	1. Validate that the server complies with OGC API — Features — Part 4 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 A.2 3. Validate that the request body complies OGC API — Features — Part 4 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 A.8	
IDENTIFIER	`/conf/mf-collection/collections-put-success`
REQUIREMEI	Requirement 10: `/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 — Features PUT response requirements

A.2.2.3. HTTP DELETE Operation

ABSTRACT TEST A.9	
IDENTIFIER	`/conf/mf-collection/collection-delete`
REQUIREMENTS	Requirement 8: `/req/mf-collection/collection-delete` Requirement 11: `/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 — Features — Part 4 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 A.10	
IDENTIFIER	`/conf/mf-collection/collections-delete-success`
REQUIREMEI	Requirement 11: `/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 — Features — Part 4 DELETE response requirements

A.3. Conformance Class MovingFeatures

CONFORMANCE CLASS A.2	
IDENTIFIER	`http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/movingfeatures`
REQUIREMENTS CLASS	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
TARGET TYPE	Web API
DEPENDENCY	http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/html http://www.opengis.net/spec/ogcapi-common-1/1.0/conf/json

CONFORMANCE CLASS A.2	
	http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/collections http://www.opengis.net/spec/ogcapi-common-2/1.0/conf/simple-query http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete
CONFORMANCE TESTS	Abstract test A.11: `/conf/movingfeatures/features-get` Abstract test A.12: `/conf/movingfeatures/features-get-success` Abstract test A.13: `/conf/movingfeatures/features-post` Abstract test A.14: `/conf/movingfeatures/features-post-success`

CONFORMANCE CLASS A.2

Abstract test A.15: /conf/movingfeatures/mf-get

Abstract test A.16: /conf/movingfeatures/mf-get-success

Abstract test A.17: /conf/movingfeatures/mf-delete

Abstract test A.18: /conf/movingfeatures/mf-delete-success

Abstract test A.19: /conf/movingfeatures/param-leaf-definition

Abstract test A.20: /conf/movingfeatures/param-leaf-response

Abstract test A.21: /conf/movingfeatures/tgsequence-get

Abstract test A.22: /conf/movingfeatures/tgsequence-get-success

Abstract test A.23: /conf/movingfeatures/tgsequence-post

Abstract test A.24: /conf/movingfeatures/

CONFORMANCE CLASS A.2

tgsequence-post-success

Abstract test A.25: /conf/movingfeatures/tpgeometry-delete

Abstract test A.26: /conf/mf-collection/tpgeometry-delete-success

Abstract test A.27: /conf/movingfeatures/tpgeometry-query-distance

Abstract test A.28: /conf/movingfeatures/tpgeometry-query-velocity

Abstract test A.29: /conf/movingfeatures/tpgeometry-query-acceleration

Abstract test A.30: /conf/movingfeatures/tproperties-get

Abstract test A.31: /conf/movingfeatures/tproperties-get-success

Abstract test A.32: /conf/movingfeatures/tproperties-post

CONFORMANCE CLASS A.2
Abstract test A.33: `/conf/movingfeatures/tproperties-post-success` Abstract test A.34: `/conf/movingfeatures/tproperty-get` Abstract test A.35: `/conf/movingfeatures/tproperty-get-success` Abstract test A.36: `/conf/movingfeatures/tproperty-post` Abstract test A.37: `/conf/movingfeatures/tproperty-post-success`

A.3.1. MovingFeatures

A.3.1.1. HTTP GET Operation

ABSTRACT TEST A.11	
IDENTIFIER	`/conf/movingfeatures/features-get`
REQUIREMENTS	Requirement 14: `/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 **Moving Feature Collections** content

ABSTRACT TEST A.11	
	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 — 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 A.12	
IDENTIFIER	`/conf/movingfeatures/features-get-success`
REQUIREMEI	/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 — Features /conf/core/fc-response 2. Validate the Collections resource for all supported media types using the resources and tests identified in Table A.4

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 A.4 — Schema and Tests for **MovingFeatures** content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	movingFeatureCollection.yaml	/conf/html/content
GeoJSON	movingFeatureCollection.yaml	/conf/geojson/content

A.3.1.2. HTTP POST Operation

ABSTRACT TEST A.13	
IDENTIFIER	`/conf/movingfeatures/features-post`
REQUIREMENTS	Requirement 18: `/req/movingfeatures/mf-mandatory` Requirement 15: `/req/movingfeatures/features-post` Requirement 17: `/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 — 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 A.5 and Table 8 3. Validate that the request body complies OGC API — 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 A.5 — 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 A.14	
IDENTIFIER	`/conf/movingfeatures/features-post-success`
REQUIREMEI	Requirement 17: `/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 — Features POST response requirements

A.3.2. MovingFeature

A.3.2.1. HTTP GET Operation

ABSTRACT TEST A.15	
IDENTIFIER	`/conf/movingfeatures/mf-get`
REQUIREMENT	/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 **Moving Feature 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 A.16	
IDENTIFIER	`/conf/movingfeatures/mf-get-success`
REQUIREMEI	/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 — Features /conf/core/f-success 2. Validate the Collections resource for all supported media types using the resources and tests identified in Table A.6

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 A.6 — Schema and Tests for MovingFeature content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	movingFeature.yaml	/conf/html/content
GeoJSON	movingFeature.yaml	/conf/geojson/content

A.3.2.2. HTTP DELETE Operation

ABSTRACT TEST A.17	
IDENTIFIER	`/conf/movingfeatures/mf-delete`
REQUIREMENTS	Requirement 20: `/req/movingfeatures/mf-delete` Requirement 22: `/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 — 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 A.18	
IDENTIFIER	`/conf/movingfeatures/mf-delete-success`
REQUIREMEI	Requirement 22: `/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 — Features DELETE response requirements

A.3.3. Parameter Leaf

ABSTRACT TEST A.19	
IDENTIFIER	`/conf/movingfeatures/param-leaf-definition`
REQUIREMEI	Requirement 23: `/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 A.20	
IDENTIFIER	`/conf/movingfeatures/param-leaf-response`
REQUIREMENTS	Requirement 23: `/req/movingfeatures/param-leaf-definition` Requirement 24: `/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 ***point AtTime*** 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

A.3.4.1. HTTP GET Operation

ABSTRACT TEST A.21	
IDENTIFIER	`/conf/movingfeatures/tgsequence-get`
REQUIREMENTS	Requirement 25: `/req/movingfeatures/tgsequence-get` Requirement 27: `/req/movingfeatures/tgsequence-get-success`
TEST PURPOSE	Validate that the **TemporalGeometrySequence** can be identified and extracted from a **Moving Feature** 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 **Moving**

ABSTRACT TEST A.21

	Feature 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 — Common and OGC API — 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 A.22

IDENTIFIER	`/conf/movingfeatures/tgsequence-get-success`
REQUIREMEI	Requirement 27: `/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 **Temporal PrimitiveGeometry** 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 — Features / conf/core/fc-links 5. If the `timeStamp` property is present, validate that it complies with OGC API — Features / conf/core/fc-timeStamp 6. If the `numberMatched` property is present, validate that it complies with OGC API — Features / conf/core/fc-numberMatched 7. If the `numberReturned` property is present, validate that it complies with OGC API — Features /conf/core/fc-numberReturned 8. Validate the TemporalGeometry resource for all supported media types using the resources and tests identified in Table A.7

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 A.7 — Schema and Tests for **TemporalGeometrySequence** content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	temporalGeometrySequence.yaml	/conf/html/content
JSON	temporalGeometrySequence.yaml	/conf/json/content

A.3.4.2. HTTP POST Operation

ABSTRACT TEST A.23	
IDENTIFIER	`/conf/movingfeatures/tgsequence-post`
REQUIREMENTS	Requirement 29: `/req/movingfeatures/tpgeometry-mandatory` Requirement 26: `/req/movingfeatures/tgsequence-post` Requirement 28: `/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 — 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 A.8 and Table 9 3. Validate that the request body complies OGC API — 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 A.8 — 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 A.24	
IDENTIFIER	`/conf/movingfeatures/tgsequence-post-success`

ABSTRACT TEST A.24	
REQUIREMEI	Requirement 28: `/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 — Features POST response requirements

A.3.5. TemporalPrimitiveGeometry

A.3.5.1. HTTP DELETE Operation

ABSTRACT TEST A.25	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-delete`
REQUIREMEN	Requirement 30: `/req/movingfeatures/tpgeometry-delete` Requirement 31: `/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 — 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 A.26	
IDENTIFIER	`/conf/mf-collection/tpgeometry-delete-success`
REQUIREMEI	Requirement 31: `/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 — Features DELETE response requirements

A.3.6. TemporalGeometry Query

A.3.6.1. HTTP GET Operation

ABSTRACT TEST A.27	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-distance`
REQUIREMEN	Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/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 — 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 A.28	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-velocity`
REQUIREMEN	Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/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 — 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`

ABSTRACT TEST A.28	
	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 A.29	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-acceleration`
REQUIREMEN	Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/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 — 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

A.3.7.1. HTTP GET Operation

ABSTRACT TEST A.30	
IDENTIFIER	`/conf/movingfeatures/tproperties-get`
REQUIREMENTS	Requirement 36: `/req/movingfeatures/tproperties-get` Requirement 38: `/req/movingfeatures/tproperties-get-success`

ABSTRACT TEST A.30

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 **Moving Feature 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 — 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 A.31

IDENTIFIER	`/conf/movingfeatures/tproperties-get-success`
REQUIREMEI	Requirement 38: `/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 — Features / conf/core/fc-links 5. If the `timeStamp` property is present, validate that it complies with OGC API — Features / conf/core/fc-timeStamp 6. If the `numberMatched` property is present, validate that it complies with OGC API — Features / conf/core/fc-numberMatched 7. If the `numberReturned` property is present, validate that it complies with OGC API — Features /conf/core/fc-numberReturned 8. Validate the **TemporalProperties** resource for all supported media types using the resources and tests identified in Table A.9

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 A.9 — Schema and Tests for **TemporalProperties** content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	temporalPropertyCollection.yaml	/conf/html/content
JSON	temporalPropertyCollection.yaml	/conf/json/content

A.3.7.2. HTTP POST Operation

ABSTRACT TEST A.32	
IDENTIFIER	`/conf/movingfeatures/tproperties-post`
REQUIREMENTS	Requirement 40: `/req/movingfeatures/tproperty-mandatory` Requirement 37: `/req/movingfeatures/tproperties-post` Requirement 39: `/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 — 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 A.10 and Table 12 3. Validate that the request body complies OGC API — 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 A.10 — 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

FORMAT	SCHEMA DOCUMENT	TEST ID
JSON	MF-JSON_Prism.schema.json	/conf/json/content

ABSTRACT TEST A.33	
IDENTIFIER	`/conf/movingfeatures/tproperties-post-success`
REQUIREMEI	Requirement 39: `/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 — Features POST response requirements

A.3.8. TemporalProperty

A.3.8.1. HTTP GET Operation

ABSTRACT TEST A.34	
IDENTIFIER	`/conf/movingfeatures/tproperty-get`
REQUIREMENTS	Requirement 41: `/req/movingfeatures/tproperty-get` Requirement 43: `/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 — Common and OGC API — MF: - Limit: Limit Tests

ABSTRACT TEST A.34	
	- 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 A.35	
IDENTIFIER	`/conf/movingfeatures/tproperty-get-success`
REQUIREMEI	Requirement 43: `/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 — Features / conf/core/fc-links 3. If the `timeStamp` property is present, validate that it complies with OGC API — Features / conf/core/fc-timeStamp 4. If the `numberMatched` property is present, validate that it complies with OGC API — Features / conf/core/fc-numberMatched 5. If the `numberReturned` property is present, validate that it complies with OGC API — Features /conf/core/fc-numberReturned 6. Validate the **TemporalProperty** resource for all supported media types using the resources and tests identified in Table A.11

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 A.11 — Schema and Tests for **TemporalProperty** content

FORMAT	SCHEMA DOCUMENT	TEST ID
HTML	temporalProperty.yaml	/conf/html/content
JSON	temporalProperty.yaml	/conf/json/content

A.3.8.2. HTTP POST Operation

ABSTRACT TEST A.36	
IDENTIFIER	`/conf/movingfeatures/tproperty-post`
REQUIREMENTS	Requirement 40: `/req/movingfeatures/tproperty-mandatory` Requirement 42: `/req/movingfeatures/tproperty-post` Requirement 44: `/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 — 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 A.12 and Table 13 3. Validate that the request body complies OGC API — 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 A.12 — 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 A.37	
IDENTIFIER	`/conf/movingfeatures/tproperty-post-success`
REQUIREMEI	Requirement 44: `/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 — Features POST response requirements

B

ANNEX B (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE)

B

ANNEX B (INFORMATIVE) 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 – 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 presented in Table B.1.

Table B.1 – A non-exhaustive list of interpolation methods listed by ISO 19107

TERM	DEFINITION
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 1: if the three points are co-linear, the circular arc becomes a straight line.

TERM	DEFINITION
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 1: If the four points are co-linear, the arc becomes a straight line. Note 2: 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(…)` 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.

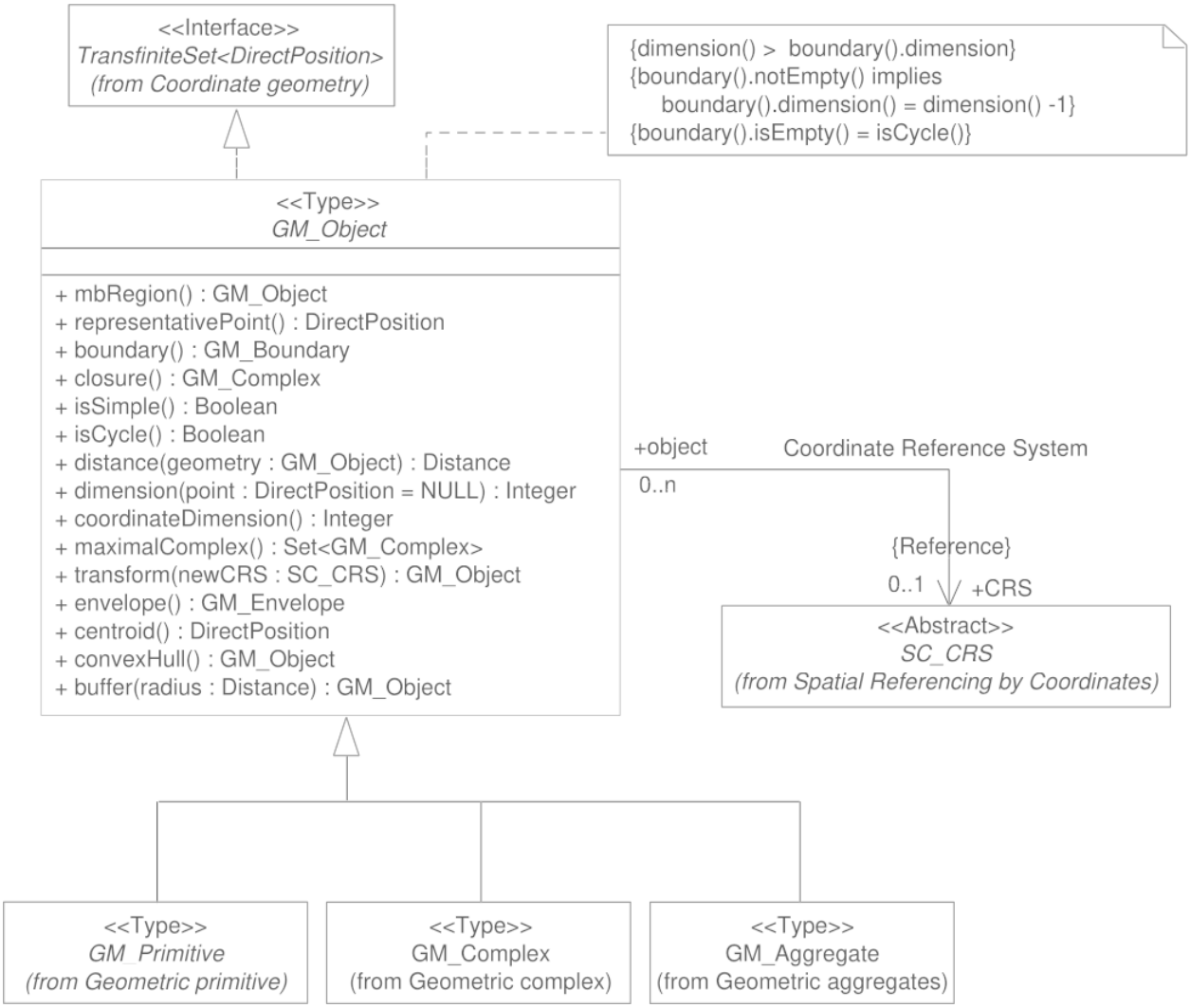

Figure B.1 — GM_Object from ISO 19107:2003 figure 6

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 — Rules for application schema* standard defines types for the definition of features. A feature is an abstraction of a real-world phenomena. The terms "feature type" and "feature instance" are used to separate the following concepts of "feature":

Feature type	The whole collection of real-world phenomena classified in a concept. For example the "bridge" feature type is the abstraction of the collection of all real-world phenomena that is classified into the concept behind the term "bridge".
Feature instance	A certain occurrence of a feature type. For example "Tower Bridge" 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.

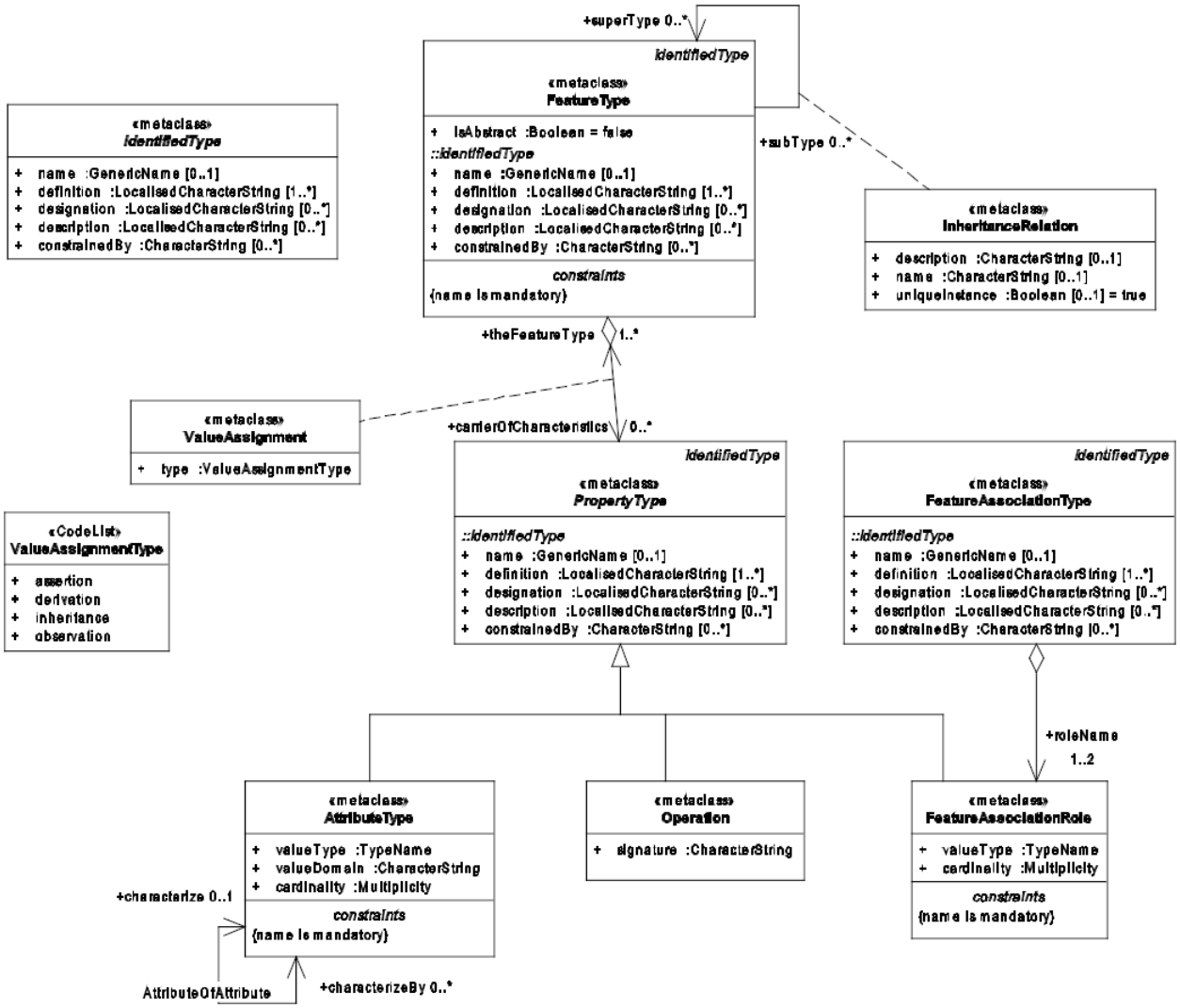

Figure B.2 — General Feature Model from ISO 19109:2009 figure 5

B.1.3. Simple Features SQL

The Simple Feature Access — 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 — Filter encoding* standard (also OGC Standard) provides types for constructing queries. These objects can be transformed into a SQL "SELECT … FROM … WHERE … ORDER BY …" 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 “contains” or “intersects”.

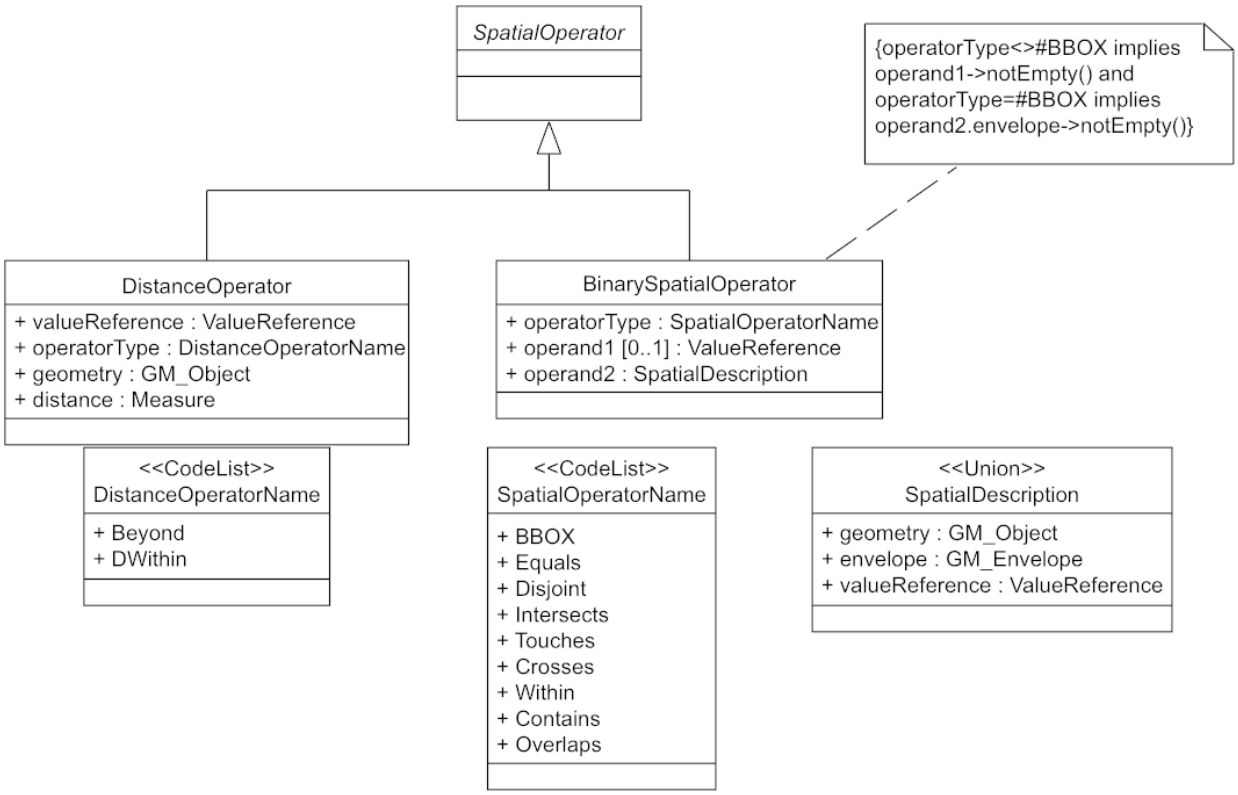

Figure B.3 — Spatial operators from ISO 19143 figure 6

B.1.5. Features web API

The OGC 17-069, *Features — 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 — 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 – Schema for moving features* standard extends the ISO 19107 spatial schema for addressing features whose locations change over time. Despite the "Moving Features" name, that standard is more about "Moving geometries". The UML below shows how the `MF_Trajectory` type extends the "static" types from ISO 19107.

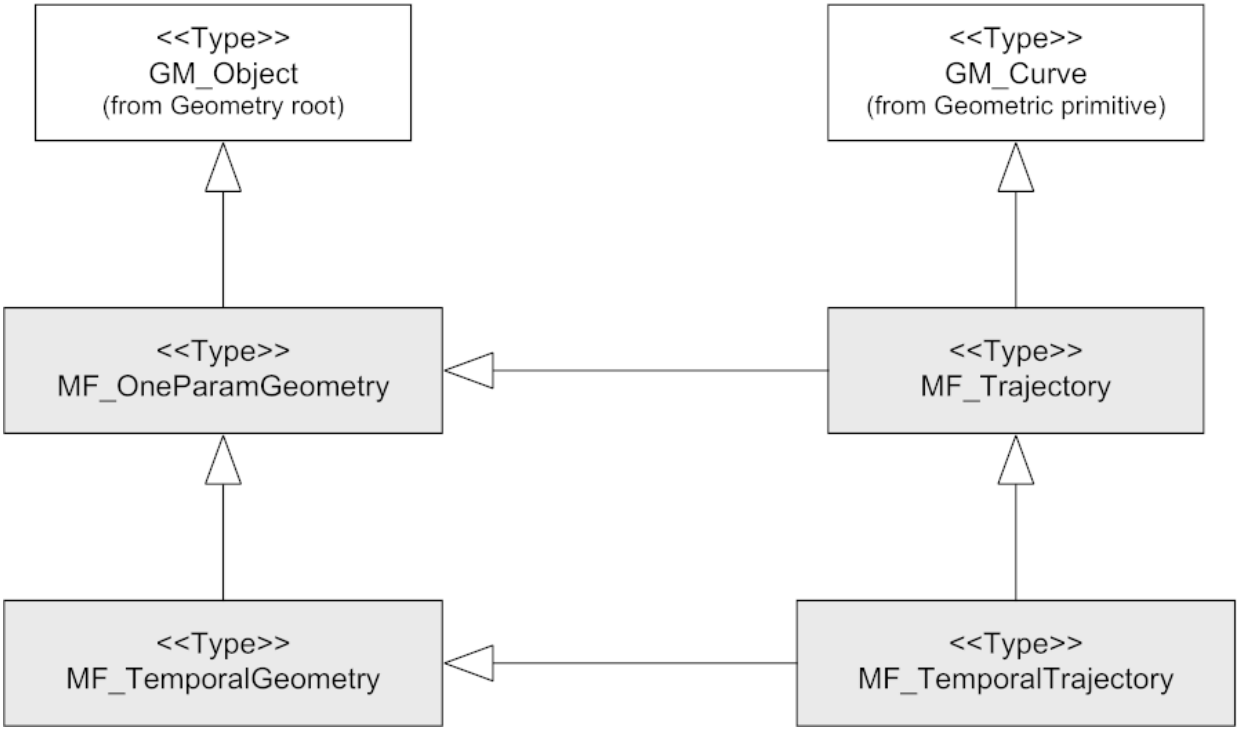

Figure B.4 – Trajectory type from ISO 19141 figure 3

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

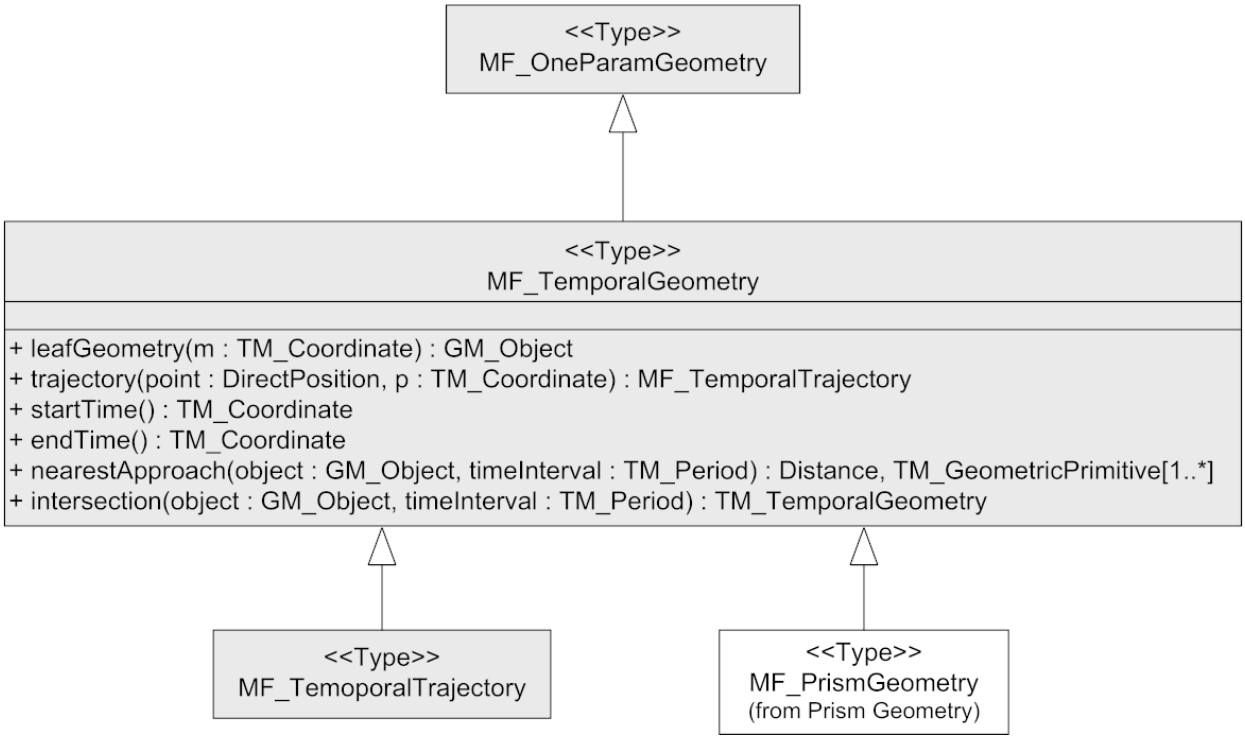

Figure B.5 — Temporal geometry from ISO 19141 figure 6

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:

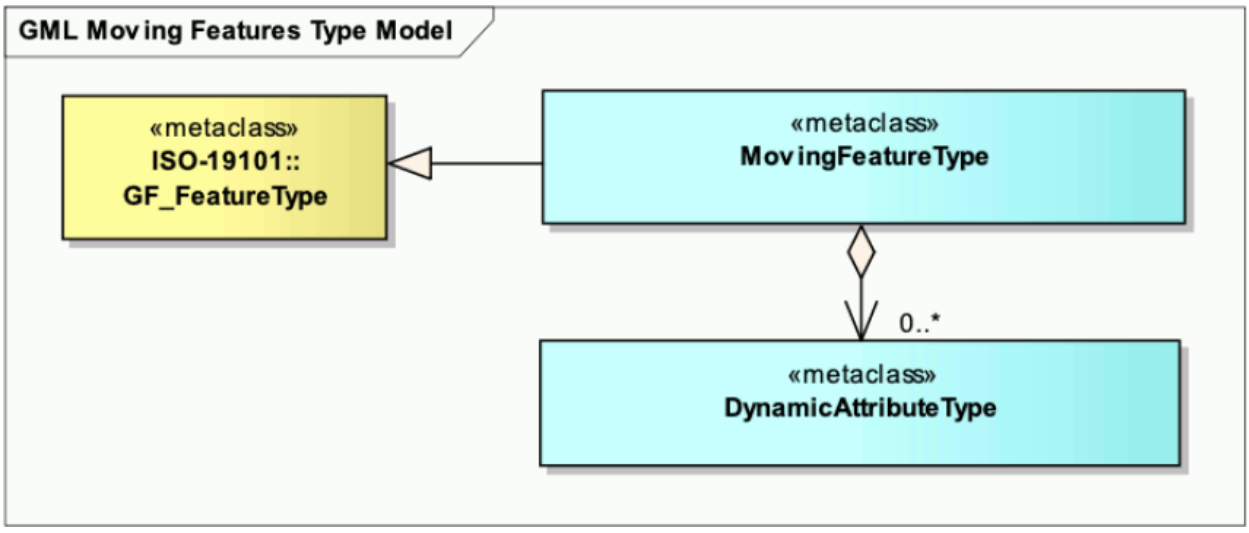

Figure B.6 — Dynamic attribute from OGC 18-075 figure 3

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

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

C

ANNEX C (INFORMATIVE) REVISION HISTORY

C

ANNEX C (INFORMATIVE) REVISION HISTORY

Table C.1

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

BIBLIOGRAPHY

BIBLIOGRAPHY

1. OGC: OGC API — Common website, https://ogcapi.ogc.org/common/
2. OGC: OGC API — Features website, https://ogcapi.ogc.org/features/
3. OGC: OGC API website, https://ogcapi.ogc.org/
4. OpenAPI initiative website, https://www.openapis.org/
5. OGC: OGC API — Features — Part 4: Create, Replace, Update and Delete candidate Standard. (2020). https://docs.ogc.org/DRAFTS/20-002.html