OGC® DOCUMENT: 22-003R3

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

Open Geospatial Consortium

OGC API - MOVING FEATURES - PART 1: CORE

STANDARD
Implementation

APPROVED

Version: 1.0
Submission Date: 2023-05-19
Approval Date: 2024-06-10
Publication Date: 2024-10-24
Editor: Taehoon Kim, Kyoung-Sook Kim, Mahmoud SAKR, Martin Desruisseaux

CONTENTS

I. ABSTRACT viii

II. KEYWORDS ix

III. PREFACE x

IV. SECURITY CONSIDERATIONS xi

V. SUBMITTING ORGANIZATIONS xii

VI. SUBMITTERS xii

1. SCOPE 2

2. CONFORMANCE 4

3. NORMATIVE REFERENCES 7

4. TERMS AND DEFINITIONS 10

5. CONVENTIONS 16
   5.1. Identifiers 16
   5.2. Use of HTTPS 16

6. OVERVIEW 18
   6.1. General 18
   6.2. Search 21
   6.3. Dependencies 21

7. REQUIREMENTS CLASS “MOVING FEATURE COLLECTION CATALOG” 24
   7.1. Overview 24
   7.2. Information Resources 24
   7.3. Resource Collections 25
   7.4. Resource Collection 30

8. REQUIREMENTS CLASS “MOVING FEATURES” 38
   8.1. Overview 38
   8.2. Information Resources 39
   8.3. Resource MovingFeatures 40
   8.4. Resource MovingFeature 49
   8.5. Resource TemporalGeometrySequence 54

8.6. Resource TemporalPrimitiveGeometry 61
8.7. TemporalGeometry Query Resources 64
8.8. Resource TemporalProperties 68
8.9. Resource TemporalProperty 75
8.10. Resource TemporalPrimitiveValue 82

9. COMMON REQUIREMENTS 86
9.1. Parameters 86
9.2. HTTP Status Codes 89

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE 92
A.1. Introduction 92
A.2. Conformance Class MovingFeature Collection Catalog 92
A.3. Conformance Class MovingFeatures 98

ANNEX B (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS 121
B.1. Static geometries, features and accesses 121
B.2. Temporal Geometries and Moving Features 126

ANNEX C (INFORMATIVE) REVISION HISTORY 130

BIBLIOGRAPHY 132

LIST OF TABLES

Table 1 — Overview of Resources viii
Table 2 — Conformance class URIs 4
Table 3 — Moving Features API Paths 18
Table 4 — Mapping OGC API — Moving Features Sections to OGC API — Common, OGC API — Features, and OGC MF-JSON Requirements Classes 21
Table 5 — Moving Feature Collection Catalog Resources 25
Table 6 — Table of collection properties 30
Table 7 — MovingFeatures Resources 39
Table 8 — Table of the properties related to the moving feature 49
Table 9 — Table of the properties related to the TemporalPrimitiveGeometry 62
Table 10 — Table of the query resources 65
Table 11 — Table of the properties related to a temporal property 76
Table 12 — Table of the properties related to the temporal primitive value 82
Table 13 — Typical HTTP status codes 89
Table A.1 — Schema and Tests for MovingFeature Collections content 94
Table A.2 — Schema and Tests for Request Body of {root}/collections POST 95

Table A.3 — Schema and Tests for MovingFeature Collection content 96
Table A.4 — Schema and Tests for MovingFeatures content 100
Table A.5 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items POST 101
Table A.6 — Schema and Tests for MovingFeature content 103
Table A.7 — Schema and Tests for TemporalGeometrySequence content 105
Table A.8 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items/{mFeatureId}/tgsequence POST 106
Table A.9 — Schema and Tests for TemporalProperties content 111
Table A.10 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items/{mFeatureId}/tproperties POST 112
Table A.11 — Schema and Tests for TemporalProperty content 114
Table A.12 — Schema and Tests for Request Body of {root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName} POST 114
Table B.1 — A non-exhaustive list of interpolation methods listed by ISO 19107 121
Table C.1 — Revision history 130

LIST OF FIGURES

Figure 1 — Class diagram for OGC API — Moving Features 20
Figure 2 — Example of a response result with a subTrajectory parameter 42
Figure 3 — Example of a response result with leaf parameter 56
Figure 4 — Example of time-to-distance curve [OGC Moving Features Access] 66
Figure B.1 — GM_Object from ISO 19107:2003 figure 6 122
Figure B.2 — General Feature Model from ISO 19109:2009 figure 5 124
Figure B.3 — Spatial operators from ISO 19143 figure 6 125
Figure B.4 — Trajectory type from ISO 19141 figure 3 126
Figure B.5 — Temporal geometry from ISO 19141 figure 6 127
Figure B.6 — Dynamic attribute from OGC 18-075 figure 3 127

LIST OF RECOMMENDATIONS

REQUIREMENTS CLASS 1: MOVING FEATURE COLLECTION CATALOG 24
REQUIREMENTS CLASS 2: MOVING FEATURES 38
REQUIREMENTS CLASS 3: MOVING FEATURES — COMMON 86
REQUIREMENT 1 26
REQUIREMENT 2 26

REQUIREMENT 3 27
REQUIREMENT 4 29
REQUIREMENT 5 30
REQUIREMENT 6 31
REQUIREMENT 7 32
REQUIREMENT 8 33
REQUIREMENT 9 33
REQUIREMENT 10 35
REQUIREMENT 11 36
REQUIREMENT 12 41
REQUIREMENT 13 41
REQUIREMENT 14 43
REQUIREMENT 15 43
REQUIREMENT 16 46
REQUIREMENT 17 48
REQUIREMENT 18 50
REQUIREMENT 19 50
REQUIREMENT 20 51
REQUIREMENT 21 52
REQUIREMENT 22 53
REQUIREMENT 23 55
REQUIREMENT 24 55
REQUIREMENT 25 57
REQUIREMENT 26 57
REQUIREMENT 27 59
REQUIREMENT 28 61
REQUIREMENT 29 62
REQUIREMENT 30 63
REQUIREMENT 31 63
REQUIREMENT 32 66
REQUIREMENT 33 67
REQUIREMENT 34 68
REQUIREMENT 35 69

REQUIREMENT 36 69
REQUIREMENT 37 70
REQUIREMENT 38 72
REQUIREMENT 39 74
REQUIREMENT 40 76
REQUIREMENT 41 77
REQUIREMENT 42 78
REQUIREMENT 43 79
REQUIREMENT 44 80
REQUIREMENT 45 81
REQUIREMENT 46 81
REQUIREMENT 47 83
REQUIREMENT 48 83
REQUIREMENT 49 84
REQUIREMENT 50 86
REQUIREMENT 51 87
REQUIREMENT 52 88
PERMISSION 1 67
CONFORMANCE CLASS A.1 92
CONFORMANCE CLASS A.2 98

ABSTRACT

Moving feature data can represent various phenomena, including vehicles, people, animals, weather patterns, etc. The OGC API — Moving Features 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 API specified in this Standard provides a way to manage data representing moving features, which can be helpful for applications in domains such as transportation management, disaster response, and environmental monitoring. This Standard also specifies operations for filtering, sorting, and aggregating moving feature data based on location, time, and other properties. The OGC API — Moving Features — Part 1: Core Standard specifies a set of RESTful interfaces and data formats for querying and updating moving feature data over the web. The Standard is part of the OGC API family of Standards and makes use of the OpenAPI Specification. OGC API Standards define modular API building blocks that spatially enable Web APIs in a consistent way. OpenAPI is used to define the reusable API building blocks with responses in JSON and HTML.

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

Table 1 — Overview of Resources

RESOURCE	PATH	HTTP METHOD	DOCUMENT REFERENCE
Collections metadata	`/collections`	GET, POST	Resource Collections
Collection instance metadata	`/collections/{collectionId}`	GET, DELETE, PUT	Resource Collection
MovingFeatures	`/collections/{collectionId}/items`	GET, POST	Resource MovingFeatures
MovingFeature instance	`/collections/{collectionId}/items/{mFeatureId}`	GET, DELETE	Resource MovingFeature
TemporalGeometrySequence	`/collections/{collectionId}/items/{mFeatureId}/tgsequence`	GET, POST	Resource TemporalGeometry Sequence
TemporalPrimitiveGeometry instance	`/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}`	DELETE	Resource TemporalPrimitive Geometry
Queries for TemporalPrimitive Geometry	`collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}`	GET	TemporalGeometry Query Resources
TemporalProperties	`/collections/{collectionId}/items/{mFeatureId}/tproperties`	GET, POST	Resource TemporalProperties

RESOURCE	PATH	HTTP METHOD	DOCUMENT REFERENCE
TemporalProperty instance	`/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyId}`	GET, POST, DELETE	Resource TemporalProperty
TemporalPrimitiveValue instance	`/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyId}/{tValueId}`	DELETE	Resource TemporalProperty

II

KEYWORDS

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

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

III PREFACE

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 Standard does not mandate any specific security controls. However, it was designed to support addition of security controls without impacting conformance, in a similar way to the OGC API — Common — Part 1: Core Standard.

This document therefore applies Requirement /req/oas30/security of OGC API — Common — Part 1: Core for OpenAPI 3.0 support of security controls.

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:

NAME	ORGANIZATION
Taehoon KIM	Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology
Kyoung-Sook 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 Standard is to provide a uniform way to access, communicate, and manage data about moving features across different applications, data providers, and data consumers in contexts where the effects of Einstein's relativity are not significant. The Standard 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 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 Standard is an extension of the OGC API — Common and the OGC API — Features Standards.

2

CONFORMANCE

2 CONFORMANCE

This Standard defines multiple requirements classes and conformance classes that describe different levels of conformance to the Standard. These requirements / conformance classes help to ensure interoperability between separate implementations of the Standard and enable data providers to specify which parts of the Standard they support. The standardization targets are “Web APIs”.

The conformance classes specified in this Standard are:

- Collection Catalog
- Moving Features
- Common Requirements

The conformance class defines the minimum requirements for an API to be compliant with the OGC API — Moving Features 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 — Moving Features 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 all the conformance classes listed in Table 2.

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
Common Requirements	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/conf/common

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. The schemas and

example API definition documents specified in this Standard can be found in the OGC Schema Repository.

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/1.0.0.

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/1.0.0.

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-1/1.0.0.

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.

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.

M. Nottingham, E. Wilde: IETF RFC 7807, *Problem Details for HTTP APIs*. RFC Publisher (2016). https://www.rfc-editor.org/info/rfc7807.

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.

Open API Initiative: OpenAPI Specification 3.0.3, https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.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. 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]

4.2. 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]

4.3. dataset

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

4.4. 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]

4.5. 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]

4.6. dynamic attribute

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

4.7. 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 19101-1:2014]

4.8. 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 19101-1:2014]

4.9. feature table

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

4.10. geographic feature

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

4.11. geometric object

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

4.12. leaf

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

4.13. 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]

4.14. property

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

4.15. 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]

4.16. 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]

4.17. trajectory

path of a moving point described by a one parameter set of points
[source: ISO 19141]

4.18. web API

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

4.19. **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 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 — Moving Features Standard extends both the OGC API — Features Standard and the OGC API — Common Standard. The OGC API — Features Standard enables 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 capabilities that are applicable to all implementations of OGC API Standards. Other OGC API Standards extend OGC API — Common with capabilities specific to a resource type.

The OGC API — Moving Features — Part 1: Core Standard defines an API with the goal of:

- Providing 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 implementation of an OGC API Standard may be accessed via a Uniform Resource Identifier (URI). The URI representation in this 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		

PATH TEMPLATE	RELATION	RESOURCE
{root}/collections	`data`	Metadata describing the Collection Catalog of data available from this API.
{root}/collections/{collectionId}		Metadata describing the Collection Catalog of data which has the unique identifier `{collectionId}`
MovingFeatures		
{root}/collections/{collectionId} /items	`items`	Static information of **MovingFeature** about available items in the specified **Collection**
{root}/collections/{collectionId}/items/{mFeatureId}	`item`	Static information describing the **MovingFeature** 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 **TemporalPrimitiveGeometry** instance
{root}/collections/{collectionId}/items/{mFeatureId}/tproperties	`items`	Temporal object information of **TemporalProperties** about available items in the specified **MovingFeature**
{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}	`item`	Temporal object describing the **TemporalProperty** of data which has the unique identifier `{tPropertyName}`
{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tProperty Name}/{tValueId}	`item`	Temporal object describing the **TemporalPrimitiveValue** of data which has the unique identifier `{tValueId}`

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
- `{tValueId}` = An identifier for a specific **TemporalPrimitiveValue** of a specific **TemporalProperty** of data

- `{quertyType}` = An identifier for the query pattern performed by an implementation instance of the OGC API — Moving Features Standard.

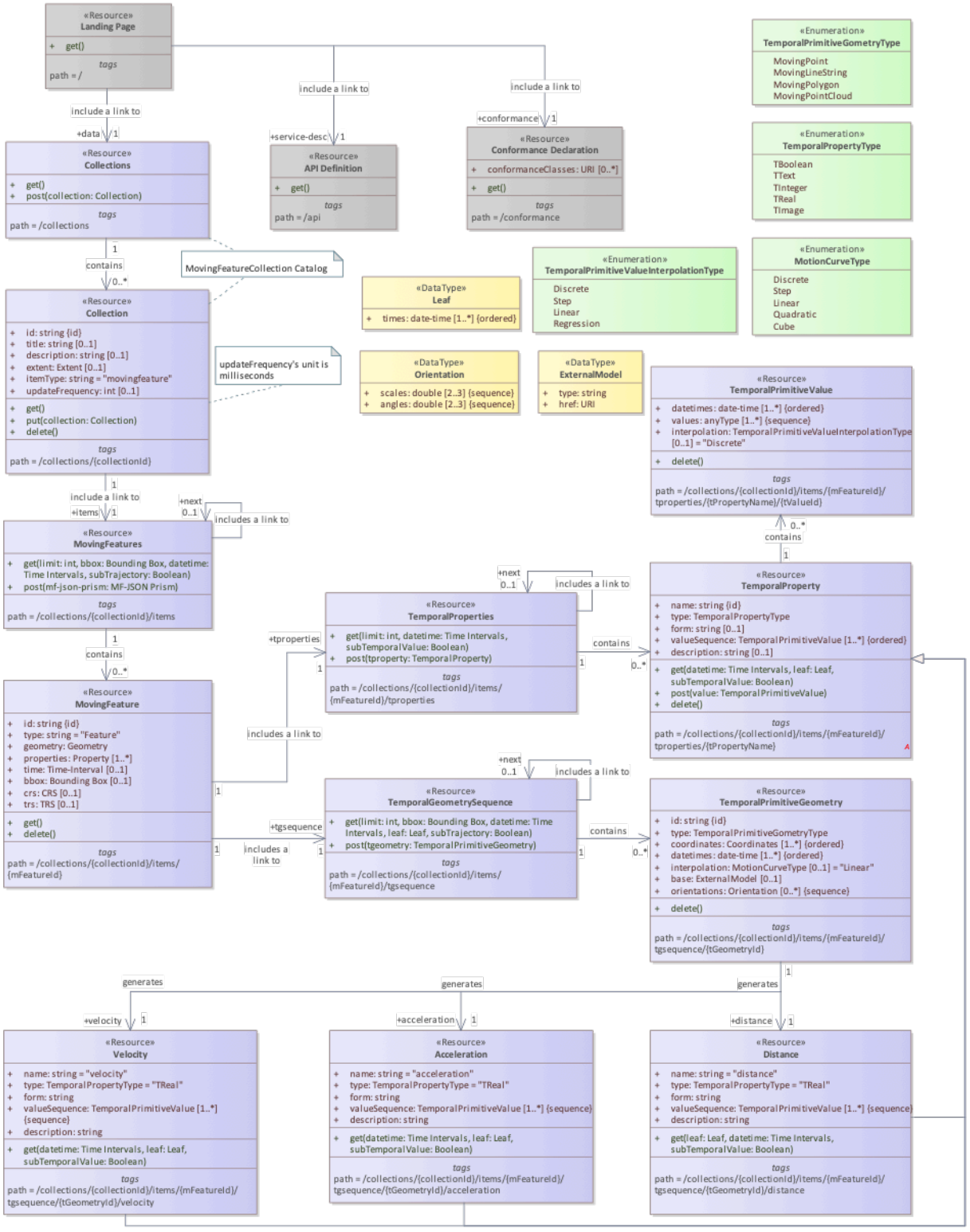

Figure 1 — Class diagram for OGC API — Moving Features

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

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

API — MF SECTION	API — MF REQUIREMENTS CLASS	API — COMMON, API — FEATURES, MF-JSON REQUIREMENTS CLASS
		http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete
MovingFeatures	/req/movingfeatures	http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core, http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete, http://www.opengis.net/spec/movingfeatures/json/1.0/req/trajectory, http://www.opengis.net/spec/movingfeatures/json/1.0/req/prism
HTML	**inherit all requirement (no modification)**	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/html
JSON	**inherit all requirement (no modification)**	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/json
GeoJSON	**inherit all requirement (no modification)**	http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson
OpenAPI 3.0	**inherit all requirement (no modification)**	http://www.opengis.net/spec/ogcapi-common-1/1.0/req/oas30

7

REQUIREMENTS CLASS "MOVING FEATURE COLLECTION CATALOG"

7 REQUIREMENTS CLASS "MOVING FEATURE COLLECTION CATALOG"

7.1. Overview

REQUIREMENTS CLASS 1: 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-1/1.0/req/core http://www.opengis.net/spec/ogcapi-common-2/1.0/req/collections 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
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 Requirements 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 from the **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` requirements 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	
IDENTIFIER	`/req/mf-collection/collections-get`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	The API SHALL support the HTTP GET operation at the path `{root}/collections`
B	The API SHALL support the HTTP GET operation on all links to a `Collections` Resource that have the relation type http://www.opengis.net/def/rel/ogc/1.0/data.

7.3.2.2. Create

The create operation is defined in the `CREATE` section of the "Create/Replace/Delete" requirements class of OGC API — Features. This operation targets the **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	
IDENTIFIER	`/req/mf-collection/collections-post`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	The server SHALL support the HTTP POST operation at the resource endpoint(`{root}/collections`).
B	A `Content-Type` header SHALL be used to declare the media type of the request body containing a representation of the resource to be added.
C	The content of the request body SHALL be based upon the **Collection** request body schema.

NOTE: See section 8.3 of RFC 9110 for details of `Content-Type`.

```
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"
```

Listing 1 — 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, below, to illustrate the details of the HTTP communication between the client and the server.

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

Listing 2 — 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 OGC API — Moving Features Standard, 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

IDENTIFIER `/req/mf-collection/collections-get-success`

REQUIREMENT 3	
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
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 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 inherited 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'
```

Listing 3 — 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"
    }
  ]
}
```

Listing 4 — An Example of a Collections JSON Payload:

7.3.3.2. Create

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

REQUIREMENT 4	
IDENTIFIER	`/req/mf-collection/collections-post-success`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	If the operation completes successfully, the server SHALL assign a new, unique identifier within the collection for the newly added resource.
B	A successful execution of the operation SHALL be reported as a response with an HTTP status code `201`.
C	A response with HTTP status code `201` SHALL include a `Location` header with the URI of the newly added resource (i.e., path of the resource endpoint).
D	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.

7.3.4. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

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 of 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 *itemType* properties were inherited from OGC API — Common and OGC API — Features.

NOTE 2: An update frequency is one of the most important properties of a moving feature collection. The update frequency can be used to handle the continuity of the moving feature’s trajectory.

REQUIREMENT 5	
IDENTIFIER	`/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 `Collection` section of the Collections requirements class of OGC API — Common. 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	
IDENTIFIER	`/req/mf-collection/collection-get`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	The API SHALL support the HTTP GET operation at the path `{root}/collections/{collectionId}`.
B	The parameter `collectionId` is each id property in the `collections` response (JSONPath: `$.collections[*].id`).

7.4.2.2. Replace

The replace operation is defined in the `REPLACE` section of the “Create/Replace/Delete” requirements class of OGC API — Features. This operation targets the **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	
IDENTIFIER	`/req/mf-collection/collection-put`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	For every resource in the collection, the server SHALL support the HTTP PUT operation.
B	The `Content-Type` header SHALL be used to indicate the media type of the request body containing the representation of the new resource content.
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 1: See section 8.3 of RFC 9110 for details of `Content-Type`.

NOTE 2: 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.

```
Client                                                          Server
  |                                                               |
  | PUT /collections/mfc_1   HTTP/1.1                             |
  | Content-Type: application/json                                |
  |                                                               |
  | {                                                             |
  |   "title": "MovingFeatureCollection_2",                       |
  |   "description": "Title is changed"                           |
  | }                                                             |
  |-------------------------------------------------------------->|
  |                                                               |
  | HTTP/1.1 204 OK                                               |
  |<--------------------------------------------------------------|
```

Listing 5 — An Example of Replacing an Existing Collection:

7.4.2.3. Delete

The delete operation is defined in the `DELETE` section of the "Create/Replace/Delete" requirements 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	
IDENTIFIER	`/req/mf-collection/collection-delete`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	For every resource in the collection (path `{root}/collections`), the server SHALL support the HTTP DELETE operation at the path `{root}/collections/{collectionId}`.
B	The parameter `collectionId` is each id property in the `collections` response (JSONPath: `$.collections[*].id`).

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.

Listing 6 — 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	
IDENTIFIER	`/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	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200`.
B	The response SHALL only include collection metadata selected by the request.
C	The content of that response SHALL be based upon the **Collection** response schema.

REQUIREMENT 9

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
```

Listing 7 — 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"
    }
  ]
}
```


Listing 8 — 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	
IDENTIFIER	`/req/mf-collection/collection-put-success`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/mf-collection`
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	If the representation of the resource submitted in the request body contained a resource identifier, the server SHALL ignore this identifier.

REQUIREMENT 10	
D	If the target resource does not exist and the server does not support creating new resources using PUT, the server SHALL indicate an unsuccessful execution of the operation with an HTTP status code `404`.
E	If the request includes an `If-Match` header and the resource does not exist, the server SHALL not create a new resource and indicate an unsuccessful execution of the operation with an HTTP status code `412`.

7.4.3.3. Delete

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

REQUIREMENT 11	
IDENTIFIER	`/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	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	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

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

8

REQUIREMENTS CLASS "MOVING FEATURES"

8 REQUIREMENTS CLASS "MOVING FEATURES"

8.1. Overview

REQUIREMENTS CLASS 2: 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-common-1/1.0/req/core http://www.opengis.net/spec/ogcapi-common-2/1.0/req/core 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
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` 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` 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`

REQUIREMENTS CLASS 2: MOVING FEATURES
Requirement 34: /req/movingfeatures/param-subtemporalvalue-definition Requirement 35: /req/movingfeatures/param-subtemporalvalue-response Requirement 36: /req/movingfeatures/tproperties-get Requirement 37: /req/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-delete Requirement 44: /req/movingfeatures/tproperty-get-success Requirement 45: /req/movingfeatures/tproperty-post-success Requirement 46: /req/movingfeatures/tproperty-delete-success Requirement 47: /req/movingfeatures/tpvalue-mandatory Requirement 48: /req/movingfeatures/tpvalue-delete Requirement 49: /req/movingfeatures/tpvalue-delete-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 Requirements Class are summarized in Table 7.

Table 7 — MovingFeatures Resources

RESOURCE	URI	HTTP METHOD
MovingFeatures	{root}/collections/{collectionId}/items	GET, POST
MovingFeature	{root}/collections/{collectionId}/items/{mfeatureId}	GET, DELETE
TemporalGeometry Sequence	{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence	GET, POST
TemporalPrimitiveGeometry	{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}	DELETE
TemporalGeometry Query	{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/{queryType}	GET

RESOURCE	URI	HTTP METHOD
TemporalProperties	`{root}/collections/{collectionId}/items/{mFeatureId}/tproperties`	GET, POST
TemporalProperty	`{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertiesName}`	GET, POST, DELETE
TemporalPrimitiveValue	`{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertiesName}/{tValueId}`	DELETE

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 — Moving Features `Items` query is an OGC API — Features endpoint that may be used to catalog a pre-existing collection resource, such as one representing 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 the `subTrajectory` parameter.

8.3.2. Query Parameters

Query parameters are used in URLs to define the resources which are returned in response to 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	
IDENTIFIER	`/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 a `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	
IDENTIFIER	`/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 subset of the trajectory derived from the 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**(Clause 8.6), 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`(Clause 8.5.2.1) parameter.

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

If the `subTrajectory` parameter is provided by the client, the endpoint SHALL return only a subset of the trajectory derived from the 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 the **TemporalPrimitiveGeometry**. The `interpolation` property in the response shall be the same as the original temporal primitive geometry.

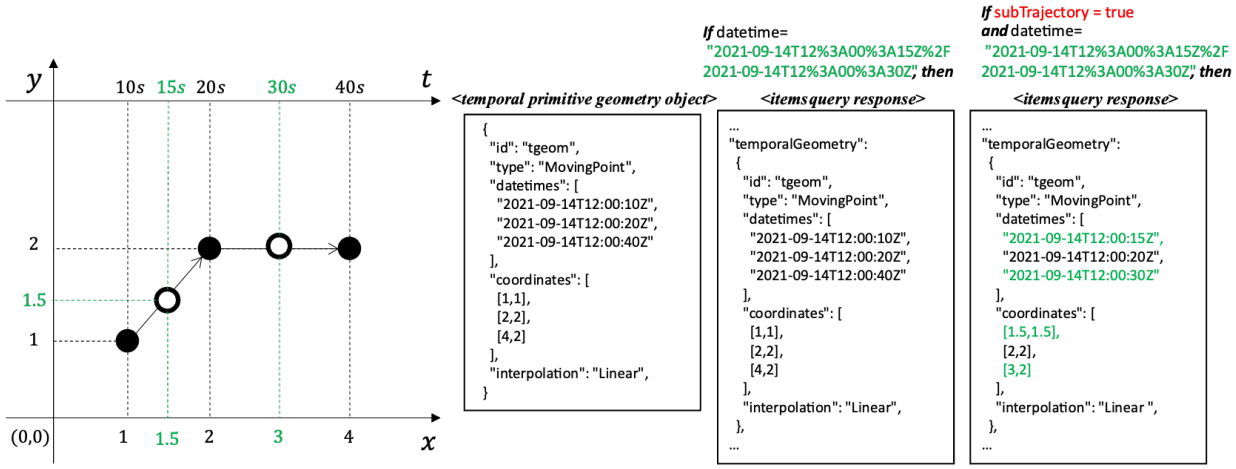

Figure 2 — Example of a response result with a `subTrajectory` parameter

8.3.3. Operation

8.3.3.1. Retrieve

The retrieve operation is defined in the `Features` section of the 'Core' requirements class of OGC API — Features. Additional support for the `subTrajectory` query parameter is needed to support the **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	
IDENTIFIER	`/req/movingfeatures/features-get`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	For every moving feature collection identified in the moving feature collections response (path `{root}/collections`), the server SHALL support the HTTP GET operation at the path `{root}/collections/{collectionId}/items`.
B	The parameter collectionId is each id property in the feature collections response (JSONPath: `$.collections[*].id`).

8.3.3.2. Create

The create operation is defined in the `CREATE` section of the "Create/Replace/Delete" requirements class of OGC API — Features. This operation targets a single or collection of **MovingFeature** resources.

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	
IDENTIFIER	`/req/movingfeatures/features-post`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The server SHALL support the HTTP POST operation at the resource endpoint(`{root}/collections/{collectionId}/items`).
B	A `Content-Type` header SHALL be used to declare the media type of the request body containing a representation of the resource to be added.
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.

NOTE: See section 8.3 of RFC 9110 for details of `Content-Type`.

The following example adds a new feature (***MovingFeature*** object in MF-JSON) to the specific **Collection**. The feature is represented as a ***MovingFeature*** object (or ***MovingFeatureCollection*** object) in MF-JSON. A pseudo-sequence diagram notation is used, below, 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://qudt.org/vocab/quantitykind/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"
         }
      }
   ]
 }
------------------------------------------------------------------------->

 HTTP/1.1 201 Created
 Location: /collections/mfc_1/items/mf_1
<-------------------------------------------------------------------------
```

Listing 10 — 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 OGC API — Moving Features, the **MovingFeatures** GET response will list the **MovingFeature** resources.

REQUIREMENT 16	
IDENTIFIER	/req/movingfeatures/features-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 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: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/crs"
  trs:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/trs"
  bbox:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/bbox"
  time:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/time"
  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
```

Listing 11 — 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
}
```

Listing 12 — An Example of a MovingFeatures GET Operation:

8.3.4.2. Create

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

A ***MovingFeatureCollection*** object of MF-JSON is a collection of MovingFeature objects. Posting a collection of resources is the same as posting a single resource consecutively. However, because the result must be returned in a single response, the `Locations` header includes a list of the URIs of the newly added resources.

REQUIREMENT 17	
IDENTIFIER	`/req/movingfeatures/features-post-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	If the operation completes successfully, the server SHALL assign a new, unique identifier within the collection for each newly added resource.
B	A successful execution of the operation SHALL be reported as a response with an HTTP status code `201`.
C	A response with HTTP status code `201` SHALL include a `Locations` header with the list of the URIs of the newly added resources (i.e., path of each moving feature resource endpoint).
D	The elements in the `Locations` header SHALL be in the same order and size as the collection of resources contained in the body of the POST request.
E	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.

8.3.5. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

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 objects, 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 the GeoJSON standard. 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.
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.

PROPERTY	REQUIREM	DESCRIPTION
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	
IDENTIFIER	`/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` section of the "Core" requirements 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	
IDENTIFIER	`/req/movingfeatures/mf-get`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 19	
A	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}`
B	The path parameter `collectionId` is each `id` property in the **Collection**(Clause 7.4) GET operation response where the value of the `itemType` property is specified as **movingfeature**. The path parameter `mFeatureId` is an `id` property of **MovingFeatures**(Clause 8.3) GET response.

8.4.2.2. Delete

The delete operation is defined in the `DELETE` section of the "Create/Replace/Delete" requirements 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	
IDENTIFIER	`/req/movingfeatures/mf-delete`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	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}`
B	The path parameter `collectionId` is each `id` property in the **Collection**(Clause 7.4) GET operation response where the value of the `itemType` property is specified as **movingfeature**. The path parameter `mFeatureId` is an `id` property of **MovingFeatures**(Clause 8.3) GET response.

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	
IDENTIFIER	/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 metadata as defined in the response schema.

```
type: object
required:
  - id
  - type
properties:
  type:
    type: string
    enum:
      - 'Feature'
  temporalGeometry:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/temporalGeometry"
  temporalProperties:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/temporalProperties"
  crs:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/crs"
  trs:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/trs"
  bbox:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/bbox"
  time:
    $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/time"
  geometry:
    $ref: 'https://schemas.opengis.net/ogcapi/features/part1/1.0/openapi/schemas/
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'
```

Listing 13 — 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"
  }
}
```

Listing 14 — 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	
IDENTIFIER	`/req/movingfeatures/mf-delete-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 22	
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	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

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

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 is 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 is 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 OpenAPI 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`
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 coordinates (or temporal property values) 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.

REQUIREMENT 24

E	The `leaf` parameter SHALL not be used with the `subTrajectory`(Clause 8.3.2.1) 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. Listing 16 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"
```

Listing 16 — 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”.

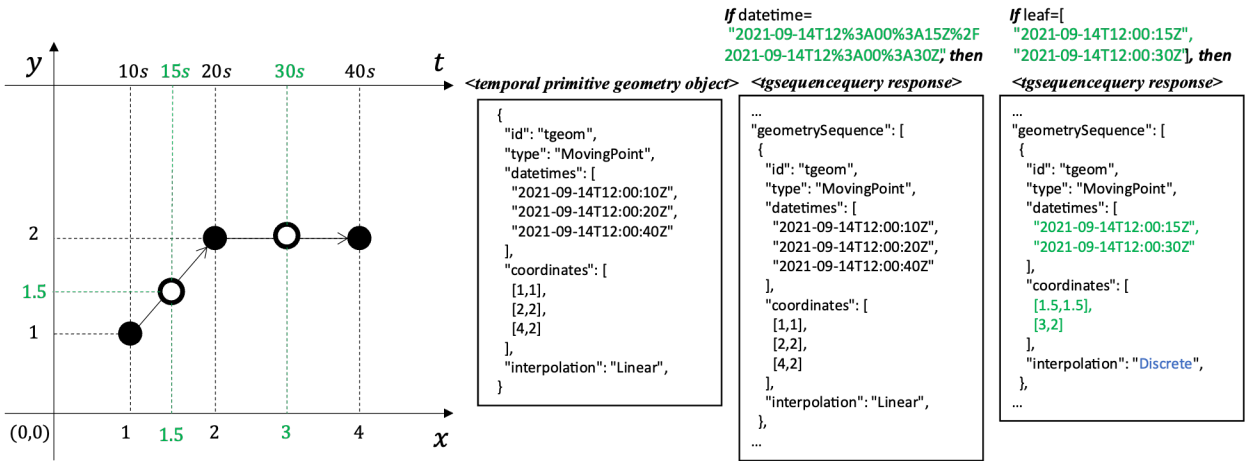

Figure 3 — Example of a response result with `leaf` parameter

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	
IDENTIFIER	`/req/movingfeatures/tgsequence-get`
INCLUDED 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**(Clause 8.3) 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**(Clause 7.4) GET response where the value of the `itemType` property is specified as **movingfeature**. The path parameter `mFeatureId` is each `id` property in the **MovingFeatures**(Clause 8.3) GET response.

8.5.3.2. Create

The create operation is defined in the `CREATE` section of the "Create/Replace/Delete" requirements class of OGC API — Features. This operation targets the **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	
IDENTIFIER	`/req/movingfeatures/tgsequence-post`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The server SHALL support the HTTP POST operation at the resource endpoint(`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence`).
B	A `Content-Type` header SHALL be used to declare the media type of the request body containing a representation of the resource to be added.
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**(Clause 8.4), 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***.

NOTE: See section 8.3 of RFC 9110 for details of `Content-Type`.

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 a ***TemporalPrimitiveGeometry*** object in MF-JSON, which is an extension of the JSON. A pseudo-sequence diagram notation is used, below, 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               |
  |<----------------------------------------------------------------------|
```

Listing 17 — 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	
IDENTIFIER	/req/movingfeatures/tgsequence-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 response SHALL only include temporal primitive geometry selected by a 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: 'https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.schema.
json#/definitions/temporalPrimitiveGeometry'
  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
```

Listing 18 — 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/tgsequence",
      "rel": "self",
      "type": "application/json"
    },
    {
      "href": "https://data.example.org/collections/mfc-1/items/mf-1/
tgsequence&offset=10&limit=1",
      "rel": "next",
      "type": "application/json"
    }
  ],
  "timeStamp": "2021-09-01T12:00:00Z",
  "numberMatched": 100,
  "numberReturned": 1
}
```


Listing 19 — An Example of a TemporalGeometrySequence GET operation:

8.5.4.2. Create

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

REQUIREMENT 28	
IDENTIFIER	`/req/movingfeatures/tgsequence-post-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	If the operation completes successfully, the server SHALL assign a new, unique identifier within the collection for the newly added resource.
B	A successful execution of the operation SHALL be reported as a response with an HTTP status code `201`.
C	A response with HTTP status code `201` SHALL include a `Location` header with the URI of the newly added resource (i.e., path of the resource endpoint).
D	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.

8.5.5. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

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 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	A temporal primitive geometry 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 single temporal primitive geometry 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`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	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}`
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.

8.6.3. Response

8.6.3.1. Delete

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

REQUIREMENT 31	
IDENTIFIER	`/req/movingfeatures/tpgeometry-delete-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` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	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

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

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 — Moving Features 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 — Moving Features.

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

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 **Temporal Property**.
`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/velocity`	Velocity	Return a graph of the time to velocity function as a form of the **Temporal Property**.
`{root}/collections/{collectionId}/items/{mFeatureId}/tgsequence/{tGeometryId}/acceleration`	Acceleration	Return a graph of the time to acceleration function as a form of the **Temporal Property**.

8.7.2. Query parameters

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

The `datetime` query parameter 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.

8.7.3. Distance Query

The Distance query returns a *time-to-distance curve* of the **TemporalPrimitiveGeometry** object as a form of the **TemporalProperty**. An implementation instance (endpoint) of the API returns derived *time-to-distance curve* data from all available time of the specified **TemporalPrimitiveGeometry** object in the absence of query parameters.

Figure 4 shows an example of the *time-to-distance curve*.

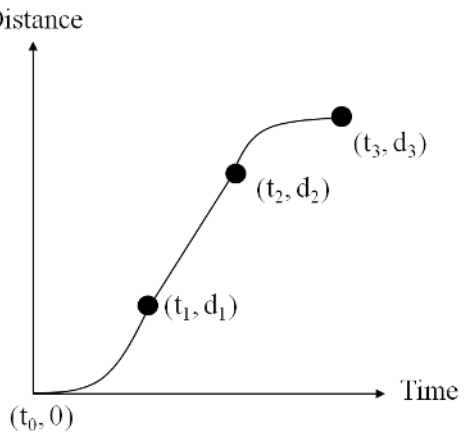

Figure 4 — Example of time-to-distance curve [OGC Moving Features Access]

8.7.4. Velocity Query

The Velocity query returns a *time-to-velocity curve* of the **TemporalPrimitiveGeometry** object as a form of the **TemporalProperty**. An implementation instance (endpoint) of the API returns derived *time-to-velocity curve* data from all available time of the specified **TemporalPrimitiveGeometry** object in the absence of query parameters.

8.7.5. Acceleration Query

The Acceleration query returns a *time-to-acceleration curve* of the **TemporalPrimitiveGeometry** object as a form of the **TemporalProperty**. An implementation instance (endpoint) of the API returns derived *time-to-acceleration curve* data from all available time of the specified **TemporalPrimitiveGeometry** object in the absence of query parameters.

8.7.6. Operation Requirements

REQUIREMENT 32	
IDENTIFIER	`/req/movingfeatures/tpgeometry-query`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	For every TemporalPrimitiveGeometry identified in the **TemporalGeometrySequence**(Clause 8.5) 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**.

REQUIREMENT 32	
	The path parameter `mFeatureId` is an `id` property of the moving feature. The path parameter `tGeometryId` is an `id` property of the temporal geometry.
C	The path parameter `queryType` SHALL be one of the predefined query type (**distance, velocity**, and **acceleration**)

PERMISSION 1	
IDENTIFIER	`/per/movingfeatures/tpgeometry-query`
A	A distance query GET operation MAY include a `datetime`, `leaf`, or `subTemporalValue` query parameter.
B	A velocity query GET operation MAY include a `datetime`, `leaf`, or `subTemporalValue` query parameter.
C	An acceleration query GET operation MAY include a `datetime`, `leaf`, or `subTemporalValue` query parameter.

8.7.7. Response Requirements

REQUIREMENT 33	
IDENTIFIER	`/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 temporal property that is defined in the response schema.
C	The `type` property SHALL be **"TReal"**

8.8. Resource TemporalProperties

8.8.1. Overview

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

1. A retrieve operation returns a list of the available abbreviated copy of **TemporalProperty** object in the specified moving feature. 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	
IDENTIFIER	`/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`
	`in: query`
	`required: false`
	`schema:`
	`  type: boolean`
	`style: form`
	`explode: false`

REQUIREMENT 34	
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	
IDENTIFIER	`/req/movingfeatures/param-subtemporalvalue-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 subset of the temporal primitive values derived from the temporal property for a specified 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**(Clause 8.9), 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`(Clause 8.5.2.1) 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** objects. 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	
IDENTIFIER	`/req/movingfeatures/tproperties-get`

REQUIREMENT 36	
INCLUDED 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**(Clause 8.3) 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**(Clause 7.4) GET response where the value of the `itemType` property is specified as **movingfeature**. The path parameter `mFeatureId` is each `id` property in the **MovingFeatures**(Clause 8.3) GET response.

8.8.3.2. Create

The create operation is defined in the `CREATE` conformance class in the OGC API — Features Standard. This operation targets a single or a collection of **TemporalProperty** resources.

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	
IDENTIFIER	`/req/movingfeatures/tproperties-post`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	The server SHALL support the HTTP POST operation at the resource endpoint(`{root}/collections/{collectionId}/items/{mFeatureId}/tproperties`).
B	A `Content-Type` header SHALL be used to declare the media type of the request body containing a representation of the resource to be added.
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.

NOTE: See section 8.3 of RFC 9110 for details of `Content-Type`.

```
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'
```

Listing 21 — 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 of a MovingFeature Example. The feature is represented as a JSON payload. A pseudo-sequence diagram notation is used, below, to illustrate the details of the HTTP communication between the client and the server.

```
Client                                                                  Server
  |                                                                       |
  | POST /collections/mfc_1/items/mf_1/tproperties   HTTP/1.1             |
  | Content-Type: application/json                                        |
  |                                                                       |
  | {                                                                     |
  |    "name": "speed",                                                   |
  |    "type": "TReal",                                                   |
  |    "form": "KMH"                                                      |
  | }                                                                     |
  |---------------------------------------------------------------------->|
  |                                                                       |
  | HTTP/1.1 201 Created                                                  |
  | Location: /collections/mfc_1/items/mf_1/tproperties/speed             |
  |<----------------------------------------------------------------------|
```

Listing 22 — An Example of Creating a New TemporalProperty Object:

```
 Client                                                                 Server
  |                                                                       |
  | POST /collections/mfc_1/items/mf_1/tproperties   HTTP/1.1             |
  | Content-Type: application/json                                        |
  |                                                                       |
  | {                                                                     |
  |    "datetimes": [                                                     |
  |       "2011-07-14T22:01:06.000Z",                                     |
  |       "2011-07-14T22:01:07.000Z",                                     |
```

```
|        "2011-07-14T22:01:08.000Z",                                          |
|     ],                                                                      |
|     "speed": {                                                              |
|        "type": "Measure",                                                   |
|        "form": "KMH",                                                       |
|        "values": [65.0, 70.0, 80.0],                                        |
|        "interpolation": "Linear"                                            |
|     }                                                                       |
|  }                                                                          |
|-----------------------------------------------------------------------------|>
|                                                                             |
|  HTTP/1.1 201 Created                                                       |
|  Location: /collections/mfc_1/items/mf_1/tproperties/speed                  |
|<----------------------------------------------------------------------------|
```

Listing 23 — 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	
IDENTIFIER	`/req/movingfeatures/tproperties-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 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 11.

```
type: object
required:
  - temporalProperties
properties:
  temporalProperties:
    oneOf:
```

```
      - $ref: "https://schemas.opengis.net/movingfeatures/1.0/MF-JSON_Prism.
schema.json#/definitions/temporalProperties"
      - 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
```

Listing 24 — 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://qudt.org/vocab/quantitykind/Length"
    },
    {
      "name": "speed",
      "type": "TReal",
      "form": "KMH"
    }
  ],
  "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
}
```

Listing 25 — 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": ["2011-07-14T22:01:06.000Z", "2011-07-14T22:01:07.000Z",
"2011-07-14T22:01:08.000Z"],
      "length": {
        "type": "Measure",
        "form": "http://qudt.org/vocab/quantitykind/Length",
        "values": [1.0, 2.4, 1.0],
        "interpolation": "Linear"
      },
      "speed" : {
        "type" : "Measure",
        "form" : "KMH",
        "values" : [65.0, 70.0, 80.0],
        "interpolation": "Linear"
      }
    }
  ],
  "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
}
```


Listing 26 — An Example of a TemporalProperties GET Operation with `subTemporalValue`:

8.8.4.2. Create

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

A ***ParametricValues*** Object in MF-JSON can be a collection of TemporalProperty objects. Posting a collection of resources is the same as posting a single resource consecutively. However, because the result must be returned in a single response, the `Locations` header includes a list of the URIs of the newly added resources.

REQUIREMENT 39

IDENTIFIER `/req/movingfeatures/tproperties-post-success`

REQUIREMENT 39	
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	If the operation completes successfully, the server SHALL assign a new, unique identifier within the collection for each newly added resource.
B	A successful execution of the operation SHALL be reported as a response with an HTTP status code `201`.
C	A response with HTTP status code `201` SHALL include a `Locations` header with the list of the URIs of the newly added resources (i.e., path of each moving feature resource endpoint).
D	The elements in the `Locations` header SHALL be in the same order and size as the collection of resources contained in the body of the POST request.
E	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.

8.8.5. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

8.9. Resource TemporalProperty

8.9.1. Overview

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

1. A retrieve operation returns a **TemporalProperty** resource which is included in the **TemporalProperties** that is 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 **TemporalPrimitiveValue** resource to the **TemporalProperties** that is specified by {tPropertyName}.
3. A delete operation deletes an existing **TemporalProperty** resource that is 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 11 defines the set of properties that may be used to describe a temporal property object.

Table 11 — Table of the properties related to a temporal property

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

NOTE: The detailed information and requirements for each property are described in the OGC Moving Features Encoding Extension — JSON Standard (OGC 19-045r3).

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 11.

8.9.2. Query Parameters

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

The `datetime` query parameter 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 — Moving Features. The list of valid values for `{tPropertyName}` is provided in the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties` GET response.

REQUIREMENT 41	
IDENTIFIER	`/req/movingfeatures/tproperty-get`
INCLUDED 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**(Clause 7.4) GET response where the value of the `itemType` property is specified as a **movingfeature** object. The path parameter `mFeatureId` is each `id` property in the **MovingFeatures**(Clause 8.3) GET response. `tPropertyName` is a local identifier of the temporal property.

8.9.3.2. Create

The create operation is defined in the `CREATE` conformance class in the OGC API — Features Standard. This operation targets the new **TemporalPrimitiveValue** object.

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	
IDENTIFIER	/req/movingfeatures/tproperty-post
INCLUDED IN	Requirements class 2: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures
A	The server SHALL support the HTTP POST operation at the resource endpoint({root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}).
B	A Content-Type header SHALL be used to declare the media type of the request body containing a representation of the resource to be added.
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 primitive value object in **Temporal Property**(Clause 8.9), determined by tPropertyName, SHALL be earlier than the beginning date-time instance (***t_new***) in the temporal primitive value object in the request body, i.e., ***t_end*** < ***t_new***.

NOTE: See section 8.3 of RFC 9110 for details of Content-Type.

```yaml
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'
```

Listing 27 — TemporalProperty Request Body Schema (TemporalPrimitiveValue.yaml):

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

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

```
 {
    "datetimes": [
       "2011-07-14T22:01:09.000Z",
       "2011-07-14T22:01:010.000Z",
    ],
    "values": [
       90.0,
       95.0,
    ],
    "interpolation": "Linear"
 }
------------------------------------------------------------------------------>

 HTTP/1.1 201 Created
 Location: /collections/mfc_1/items/mf_1/tproperties/speed
<------------------------------------------------------------------------------
```

Listing 28 — An Example of Creating a New TemporalPrimitiveValue Object:

8.9.3.3. Delete

The delete operation is defined in the `DELETE` section of the "Create/Replace/Delete" requirements class of OGC API — Features.

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

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

REQUIREMENT 43	
IDENTIFIER	`/req/movingfeatures/tproperty-delete`
INCLUDED 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}`), the server SHALL support the HTTP DELETE operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}`
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 `tPropertyName` is a local identifier of the temporal property.

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 44	
IDENTIFIER	`/req/movingfeatures/tproperty-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 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"
    }
  ]
}
```


Listing 29 — An Example of TemporalProperty GET Operation:

8.9.4.2. Create

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

REQUIREMENT 45	
IDENTIFIER	`/req/movingfeatures/tproperty-post-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	If the operation completes successfully, the server SHALL assign a new, unique identifier within the collection for the newly added resource.
B	A successful execution of the operation SHALL be reported as a response with an HTTP status code `201`.
C	A response with HTTP status code `201` SHALL include a `Location` header with the URI of the newly added resource (i.e., path of the resource endpoint).
D	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.

8.9.4.3. Delete

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

REQUIREMENT 46	
IDENTIFIER	`/req/movingfeatures/tproperty-delete-success`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`

REQUIREMENT 46	
A	A successful execution of the operation SHALL be reported as a response with an HTTP status code `200` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	If no resource with the identifier exists in the collection, the server SHALL respond with a not-found exception (`404`).

8.9.5. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

8.10. Resource TemporalPrimitiveValue

8.10.1. Overview

The **TemporalPrimitiveValue** resource represents the dynamic change of a non-spatial attribute's value with time. An abbreviated copy of this information is returned for each **TemporalPrimitiveValue** in the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` response.

The schema for the temporal primitive value object presented in this clause is a part of the ***ParametricValues*** Object defined in MF-JSON. Table 12 defines the set of properties that shall be used to describe a temporal primitive value.

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

PROPERTY	REQUIREMENT	DESCRIPTION
id	M	A unique identifier to the temporal primitive value.
datetimes	M	A sequence of monotonic increasing instants.
values	M	A sequence of dynamic values having the same number of elements as "datetimes".

PROPERTY	REQUIREMENT	DESCRIPTION
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 Features Encoding Extension — JSON Standard (OGC 19-045r3).

REQUIREMENT 47	
IDENTIFIER	`/req/movingfeatures/tpvalue-mandatory`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	A temporal primitive value object SHALL contain all the mandatory properties listed in Table 12.

8.10.2. Operation

8.10.2.1. Delete

The delete operation is defined in the `DELETE` section of the "Create/Replace/Delete" requirements class of OGC API — Features.

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

The {tValueId} parameter is the unique identifier for a single temporal primitive value object offered by the API. The list of valid values for `{tValueId}` is provided in the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}` GET response.

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

REQUIREMENT 48	
IDENTIFIER	`/req/movingfeatures/tpvalue-delete`
INCLUDED IN	Requirements class 2: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/movingfeatures`
A	For every temporal primitive value in a temporal property (path `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}`), the server SHALL

REQUIREMENT 48	
	support the HTTP DELETE operation at the path `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tPropertyName}/{tValueId}`
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 `tPropertyName` is a local identifier of the temporal property. The path parameter `tValueId` is an `id` property of the temporal primitive value.

8.10.3. Response

8.10.3.1. Delete

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

REQUIREMENT 49	
IDENTIFIER	`/req/movingfeatures/tpvalue-delete-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` or `204`.
B	If the operation is not executed immediately, but is added to a processing queue, the response SHALL have an HTTP status code `202`.
C	If no resource with the identifier exists in the collection, the server SHALL respond with a not-found exception (`404`).

8.10.4. Error situations

General guidance on HTTP status codes and how they should be handled is provided in Clause 9.2.

9

COMMON REQUIREMENTS

9 COMMON REQUIREMENTS

REQUIREMENTS CLASS 3: MOVING FEATURES — COMMON	
IDENTIFIER	http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common
TARGET TYPE	Web API
PREREQUISITES	http://www.opengis.net/spec/ogcapi-features-1/1.0/req/core http://www.opengis.net/spec/ogcapi-common-1/1.0/req/core http://www.opengis.net/spec/ogcapi-common-2/1.0/req/core
NORMATIVE STATEMENTS	Requirement 50: /req/common/param-limit Requirement 51: /req/common/param-bbox Requirement 52: /req/common/param-datetime

9.1. Parameters

The query parameters bbox, datetime, and limit are inherited from OGC API — Common. All requirements and recommendations in OGC API — Common regarding these parameters also apply to OGC API — Moving Features.

9.1.1. Parameter limit

REQUIREMENT 50	
IDENTIFIER	/req/common/param-limit
INCLUDED IN	Requirements class 3: http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common

A	The limit parameter SHALL possess the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: limit`
	`in: query`
	`required: false`
	`schema:`
	`  type: integer`
	`  minimum: 1`
	`  maximum: 10000`
	`  default: 10`
	`style: form`
	`explode: false`

REQUIREMENT 50	
B	If the `limit` parameter is provided by the client and supported by the server, then the response SHALL NOT contain more collections than specified by the `limit` parameter.
C	If the API definition specifies a maximum value for the `limit` parameter, the response SHALL NOT contain more collections than this maximum value.
D	Only items are counted that are on the first level of the collection. Any nested objects contained within the explicitly requested items SHALL not be counted.

NOTE: The values for `minimum`, `maximum` and `default` are only examples and MAY be changed.

9.1.2. Parameter bbox

REQUIREMENT 51	
IDENTIFIER	`/req/common/param-bbox`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`

A	The `bbox` parameter SHALL possess the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: bbox`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  oneOf:`
	`  - minItems: 4`
	`  maxItems: 4`
	`  - minItems: 6`
	`  maxItems: 6`
	`  items:`
	`  type: number`
	`style: form`
	`explode: false`

B	The bounding box SHALL be provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (height or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Minimum value, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Maximum value, coordinate axis 3 (optional)

REQUIREMENT 51	
C	If the bounding box consists of four numbers, the coordinate reference system of the values SHALL be interpreted as WGS 84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
D	If the bounding box consists of six numbers, the coordinate reference system of the values SHALL be interpreted as WGS 84 longitude/latitude/ellipsoidal height (http://www.opengis.net/def/crs/OGC/0/CRS84h) unless a different coordinate reference system is specified in a parameter `bbox-crs`.
E	Only features that have a spatial geometry that intersects the bounding box SHALL be part of the result set, if the `bbox` parameter is provided.
F	If a feature has multiple spatial geometry properties, it is the decision of the server whether only a single spatial geometry property is used to determine the extent or all relevant geometries.
G	The `bbox` parameter SHALL match all features in the collection that are not associated with a spatial geometry, too.
H	The coordinate values SHALL be within the extent specified for the coordinate reference system.

9.1.3. Parameter datetime

REQUIREMENT 52	
IDENTIFIER	`/req/common/param-datetime`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`

A	The `datetime` parameter SHALL have the following characteristics (using an OpenAPI Specification 3.0 fragment):
	`name: datetime`
	`in: query`
	`required: true`
	`schema:`
	`  type: string`
	`style: form`
	`explode: false`

B	Only features that have a temporal primitive geometry (and temporal primitive value) that intersects the temporal information in the `datetime` parameter SHALL be part of the result set, if the parameter is provided.

C	Temporal primitive geometries (and temporal primitive values) are a half-bounded time interval (i.e. , ***(t_s, t_e + updateFrequency]***). The parameter value SHALL conform to the following syntax (using ABNF):
	`interval-bounded = date-time "/" date-time`
	`interval-half-bounded-start = [".."] "/" date-time`
	`interval-half-bounded-end = date-time "/" [".."]`
	`interval = interval-bounded / interval-half-bounded-start / interval-half-bounded-end`
	`datetime = date-time / interval`

REQUIREMENT 52	
D	Server implementations SHALL interpret the date-time as specified by RFC 3339, 5.6 supporting at least UTC time with the notation ending with a Z (with support for local time offsets optional).
E	When a double-dot (`..`) or an empty string is specified in a time interval, the implementation SHALL interpret it as a half-bounded or an unbounded interval (open range).
F	If the `datetime` parameter is provided by the client and supported by the server, then only collections whose temporal extent intersects the interval or instant of the `datetime` parameter and collections that do not describe a temporal extent SHALL be part of the result set.

9.2. HTTP Status Codes

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

Table 13 — Typical HTTP status codes

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

STATUS CODE	DESCRIPTION
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.
412	The status code indicates that one or more conditions given in the request header fields evaluated to false when tested by the server.
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.
415	The server is refusing to service the request because the content is in a format not supported by this method on the target resource.
500	An internal error occurred in the server.

The status codes described in Table 13 do not cover all possible conditions. See IETF RFC 7231 for a complete list of HTTP status codes. When a server encounters an error in the processing of a request, the server may wish to include information in addition to the status code in the response. Since Web API interactions are often machine-to-machine, a machine-readable report would be preferred. IETF RFC 7807 addresses this need by providing “Problem Details” response schemas for both JSON and XML.

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A

ANNEX A (NORMATIVE) ABSTRACT TEST SUITE

A.1. Introduction

The Abstract Test Suite (ATS) presented in this Annex is a compendium of test assertions applicable to implementations of the OGC API — Moving Features — Part 1: Core Standard. 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 classes defined in the OGC API — Moving Features — Part 1: Core Standard.

Implementations of OGC API Standards 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-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 Abstract test A.2: `/conf/mf-collection/collections-get-success`

ABSTRACT TEST A.2	
IDENTIFIER	`/conf/mf-collection/collections-get-success`
REQUIREMENT	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 an `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 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 the body of a POST request, for all supported media types, using the resources and tests identified in Table A.2 3. Validate that the request body complies with 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 Abstract test A.4: `/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`
REQUIREMENT	Requirement 4: `/req/mf-collection/collections-post-success`
TEST PURPOSE	Validate that the response of the `{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 Abstract test A.6: `/conf/mf-collection/collection-get-success`

ABSTRACT TEST A.6	
IDENTIFIER	`/conf/mf-collection/collection-get-success`
REQUIREMENTS	Requirement 5: `/req/mf-collection/mandatory-collection` Requirement 9: `/req/mf-collection/collection-get-success`
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 the body of a PUT request, for all supported media types, using the resources and tests identified in Table A.2 3. Validate that the request body complies with 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 Abstract test A.8: `/conf/mf-collection/collections-put-success`

ABSTRACT TEST A.8	
IDENTIFIER	`/conf/mf-collection/collections-put-success`
REQUIREMENT	Requirement 10: `/req/mf-collection/collection-put-success`
TEST PURPOSE	Validate that the response of the `{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 Abstract test A.10: `/conf/mf-collection/collections-delete-success`

ABSTRACT TEST A.10	
IDENTIFIER	`/conf/mf-collection/collections-delete-success`
REQUIREMENT	Requirement 11: `/req/mf-collection/collection-delete-success`
TEST PURPOSE	Validate that the response of the `{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` Requirements class 3: `http://www.opengis.net/spec/ogcapi-movingfeatures-1/1.0/req/common`
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-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` 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/tgsequence-get` Abstract test A.20: `/conf/movingfeatures/tgsequence-get-success` Abstract test A.21: `/conf/movingfeatures/tgsequence-post`

CONFORMANCE CLASS A.2	
	Abstract test A.22: `/conf/movingfeatures/tgsequence-post-success` Abstract test A.23: `/conf/movingfeatures/tpgeometry-delete` Abstract test A.24: `/conf/movingfeatures/tpgeometry-delete-success` Abstract test A.25: `/conf/movingfeatures/tpgeometry-query-distance` Abstract test A.26: `/conf/movingfeatures/tpgeometry-query-velocity` Abstract test A.27: `/conf/movingfeatures/tpgeometry-query-acceleration` Abstract test A.28: `/conf/movingfeatures/tproperties-get` Abstract test A.29: `/conf/movingfeatures/tproperties-get-success` Abstract test A.30: `/conf/movingfeatures/tproperties-post` Abstract test A.31: `/conf/movingfeatures/tproperties-post-success` Abstract test A.32: `/conf/movingfeatures/tproperty-get` Abstract test A.33: `/conf/movingfeatures/tproperty-get-success` Abstract test A.34: `/conf/movingfeatures/tproperty-post` Abstract test A.35: `/conf/movingfeatures/tproperty-post-success` Abstract test A.36: `/conf/movingfeatures/tproperty-delete` Abstract test A.37: `/conf/movingfeatures/tproperty-delete-success` Abstract test A.38: `/conf/movingfeatures/tpvalue-delete` Abstract test A.39: `/conf/movingfeatures/tpvalue-delete-success` Abstract test A.40: `/conf/movingfeatures/param-leaf-definition` Abstract test A.41: `/conf/movingfeatures/param-leaf-response` Abstract test A.42: `/conf/movingfeatures/param-subtrajectory-definition` Abstract test A.43: `/conf/movingfeatures/param-subtrajectory-response` Abstract test A.44: `/conf/movingfeatures/param-subtemporalvalue-definition` Abstract test A.45: `/conf/movingfeatures/param-subtemporalvalue-response`

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` Requirement 16: `/req/movingfeatures/features-get-success` Requirement 50: `/req/common/param-limit` Requirement 51: `/req/common/param-bbox` Requirement 52: `/req/common/param-datetime`
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}`

ABSTRACT TEST A.11	
	is the id property for a **MovingFeature Collection** described in the **MovingFeature Collections** content 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using Abstract test A.12: `/conf/movingfeatures/features-get-success` Repeat these tests using the following parameter tests that are defined in the OGC API — Common and OGC API — Moving Features Standards: - Limit: Limit Tests - Bounding Box: Bounding Box Tests - Date-Time: Date-Time Tests - SubTrajectory: SubTrajectory Definition Test and SubTrajectory Response Test Execute requests with combinations of the `"bbox"`, `"datetime"`, and `"subTrajectory"` query parameters and verify that only features are returned that match both selection criteria.

ABSTRACT TEST A.12	
IDENTIFIER	`/conf/movingfeatures/features-get-success`
REQUIREMENT	Requirement 16: `/req/movingfeatures/features-get-success`
TEST PURPOSE	Validate that the **MovingFeatures** comply 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 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 15: `/req/movingfeatures/features-post` Requirement 17: `/req/movingfeatures/features-post-success` Requirement 18: `/req/movingfeatures/mf-mandatory`
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 the body of a POST request, 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 with 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 Abstract test A.14: `/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`
REQUIREMENT	Requirement 17: `/req/movingfeatures/features-post-success`
TEST PURPOSE	Validate that the response of the `{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`
REQUIREMENTS	Requirement 19: `/req/movingfeatures/mf-get` Requirement 21: `/req/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 Abstract test A.16: `/conf/movingfeatures/mf-get-success`

ABSTRACT TEST A.16	
IDENTIFIER	`/conf/movingfeatures/mf-get-success`
REQUIREMENT	Requirement 21: `/req/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 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 Abstract test A.18: `/conf/movingfeatures/mf-delete-success`

ABSTRACT TEST A.18	
IDENTIFIER	`/conf/movingfeatures/mf-delete-success`
REQUIREMENT	Requirement 22: `/req/movingfeatures/mf-delete-success`
TEST PURPOSE	Validate that the response of the `{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. TemporalGeometrySequence

A.3.3.1. HTTP GET Operation

ABSTRACT TEST A.19	
IDENTIFIER	`/conf/movingfeatures/tgsequence-get`
REQUIREMENTS	Requirement 25: `/req/movingfeatures/tgsequence-get` Requirement 27: `/req/movingfeatures/tgsequence-get-success` Requirement 50: `/req/common/param-limit` Requirement 51: `/req/common/param-bbox` Requirement 52: `/req/common/param-datetime`
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 **MovingFeature Collection** described in the **MovingFeature Collection** content and `{mFeatureId}` is the id property for the **MovingFeature** 1. Validate that a document was returned with a status code `200` 2. Validate the contents of the returned document using Abstract test A.20: `/conf/movingfeatures/tgsequence-get-success` Repeat these tests using the following parameter tests that are defined in the OGC API — Common and OGC API — Moving Features Standards: - Limit: Limit Tests - Bounding Box: Bounding Box Tests - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test - SubTrajectory: SubTrajectory Definition Test and SubTrajectory Response Test Execute requests with combinations of the `"bbox"`, `"datetime"`, and `"leaf"` (or `"subTrajectory"`) query parameters and verify that only features are returned that match both selection criteria.

ABSTRACT TEST A.20	
IDENTIFIER	`/conf/movingfeatures/tgsequence-get-success`
REQUIREMENT	Requirement 27: `/req/movingfeatures/tgsequence-get-success`

ABSTRACT TEST A.20	
TEST PURPOSE	Validate that the **TemporalGeometrySequence** complies with the required structure and contents.
TEST METHOD	1. Validate that the `type` property is present and has a value of `MovingGeometryCollection` 2. Validate the `prism` property is present and that it is populated with an array of **TemporalPrimitiveGeometry** items 3. Validate that only **TemporalPrimitiveGeometry** which match the selection criteria are included in the **MovingFeature** 4. If the `links` property is present, validate that all entries comply with OGC API — 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 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.3.3.2. HTTP POST Operation

ABSTRACT TEST A.21	
IDENTIFIER	`/conf/movingfeatures/tgsequence-post`
REQUIREMENTS	Requirement 26: `/req/movingfeatures/tgsequence-post` Requirement 28: `/req/movingfeatures/tgsequence-post-success` Requirement 29: `/req/movingfeatures/tpgeometry-mandatory`

ABSTRACT TEST A.21	
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 the body of a POST request, 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 with 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 Abstract test A.22: `/conf/movingfeatures/tgsequence-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.22	
IDENTIFIER	`/conf/movingfeatures/tgsequence-post-success`
REQUIREMENT	Requirement 28: `/req/movingfeatures/tgsequence-post-success`
TEST PURPOSE	Validate that the response of the `{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.3.4. TemporalPrimitiveGeometry

A.3.4.1. HTTP DELETE Operation

ABSTRACT TEST A.23	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-delete`
REQUIREMENTS	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 Abstract test A.24: `/conf/movingfeatures/tpgeometry-delete-success`

ABSTRACT TEST A.24	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-delete-success`
REQUIREMENT	Requirement 31: `/req/movingfeatures/tpgeometry-delete-success`
TEST PURPOSE	Validate that the response of the `{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.3.5. TemporalGeometry Query

A.3.5.1. HTTP GET Operation

ABSTRACT TEST A.25	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-distance`
REQUIREMENTS	Permission 1: `/per/movingfeatures/tpgeometry-query` Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/req/movingfeatures/tpgeometry-query-success` Requirement 52: `/req/common/param-datetime`

ABSTRACT TEST A.25

TEST PURPOSE	Validate that resources can be identified and extracted from a **TemporalPrimitiveGeometry** with a `Distance` query using query parameters.
TEST METHOD	IF any of the query parameters are not empty, validate that the query parameters with the following parameter tests are defined in the OGC API — Common and OGC API — Moving Features: - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test - SubTemporalValue: SubTemporalValue Definition Test and SubTemporalValue Response Test 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 any of the query parameters are not empty: Execute requests with used query parameter and verify the correctly calculated value is returned. IF all query parameters are empty: Verify that a *time-to-distance curve* is correctly returned according to the specified **TemporalPrimitiveGeometry** resource by {tGeometryId}.

ABSTRACT TEST A.26

IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-velocity`
REQUIREMENTS	Permission 1: `/per/movingfeatures/tpgeometry-query` Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/req/movingfeatures/tpgeometry-query-success` Requirement 52: `/req/common/param-datetime`
TEST PURPOSE	Validate that resources can be identified and extracted from a **TemporalPrimitiveGeometry** with a `Velocity` query using query parameters.
TEST METHOD	IF any of the query parameters are not empty, validate that the query parameters with the following parameter tests are defined in the OGC API — Common and OGC API — Moving Features: - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test - SubTemporalValue: SubTemporalValue Definition Test and SubTemporalValue Response Test 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 any of the query parameters are not empty: Execute requests with used query parameter and verify the correctly calculated value is returned.

ABSTRACT TEST A.26	
	IF all query parameters are empty: Verify that a *time-to-velocity curve* is correctly returned according to the specified **TemporalPrimitiveGeometry** resource by {tGeometryId}.

ABSTRACT TEST A.27	
IDENTIFIER	`/conf/movingfeatures/tpgeometry-query-acceleration`
REQUIREMENTS	Permission 1: `/per/movingfeatures/tpgeometry-query` Requirement 32: `/req/movingfeatures/tpgeometry-query` Requirement 33: `/req/movingfeatures/tpgeometry-query-success` Requirement 52: `/req/common/param-datetime`
TEST PURPOSE	Validate that resources can be identified and extracted from a **TemporalPrimitiveGeometry** with an `Acceleration` query using query parameters.
TEST METHOD	IF any of the query parameters are not empty, validate that the query parameters with the following parameter tests are defined in the OGC API — Common and OGC API — Moving Features: - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test - SubTemporalValue: SubTemporalValue Definition Test and SubTemporalValue Response Test 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 any of the query parameters are not empty: Execute requests with used query parameter and verify the correctly calculated value is returned. IF all query parameters are empty: Verify that a *time-to-acceleration curve* is correctly returned according to the specified **TemporalPrimitiveGeometry** resource by {tGeometryId}.

A.3.6. TemporalProperties

A.3.6.1. HTTP GET Operation

ABSTRACT TEST A.28	
IDENTIFIER	`/conf/movingfeatures/tproperties-get`
REQUIREMENTS	Requirement 36: `/req/movingfeatures/tproperties-get` Requirement 38: `/req/movingfeatures/tproperties-get-success`

ABSTRACT TEST A.28

	Requirement 50: `/req/common/param-limit` Requirement 52: `/req/common/param-datetime`
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 Abstract test A.29: `/conf/movingfeatures/tproperties-get-success` Repeat these tests using the following parameter tests that are defined in the OGC API — Common and OGC API — Moving Features Standards: - Limit: Limit Tests - Date-Time: Date-Time Tests - SubTemporalValue: SubTemporalValue Definition Test and SubTemporalValue Response Test Execute requests with combinations of the `"datetime"` and `"subTemporalValue"` query parameters and verify that only features are returned that match both selection criteria.

ABSTRACT TEST A.29

IDENTIFIER	`/conf/movingfeatures/tproperties-get-success`
REQUIREMENT	Requirement 38: `/req/movingfeatures/tproperties-get-success`
TEST PURPOSE	Validate that the **TemporalProperties** property complies with the required structure and contents.
TEST METHOD	1. Validate that 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 that 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

ABSTRACT TEST A.29
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 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.3.6.2. HTTP POST Operation

ABSTRACT TEST A.30	
IDENTIFIER	`/conf/movingfeatures/tproperties-post`
REQUIREMENTS	Requirement 37: `/req/movingfeatures/tproperties-post` Requirement 39: `/req/movingfeatures/tproperties-post-success` Requirement 40: `/req/movingfeatures/tproperty-mandatory`
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 the body of a POST request using for all supported media types using the resources and tests identified in Table A.10 and Table 11 3. Validate that the request body complies with 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 Abstract test A.31: `/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
JSON	MF-JSON_Prism.schema.json	/conf/json/content

ABSTRACT TEST A.31	
IDENTIFIER	`/conf/movingfeatures/tproperties-post-success`
REQUIREMENT	Requirement 39: `/req/movingfeatures/tproperties-post-success`
TEST PURPOSE	Validate that the response of the `{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.3.7. TemporalProperty

A.3.7.1. HTTP GET Operation

ABSTRACT TEST A.32	
IDENTIFIER	`/conf/movingfeatures/tproperty-get`
REQUIREMENTS	Requirement 41: `/req/movingfeatures/tproperty-get` Requirement 44: `/req/movingfeatures/tproperty-get-success` Requirement 50: `/req/common/param-limit` Requirement 52: `/req/common/param-datetime`
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**

ABSTRACT TEST A.32

	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 Abstract test A.33: `/conf/movingfeatures/tproperty-get-success` Repeat these tests using the following parameter tests that are defined in the OGC API — Common and OGC API — Moving Features Standards: - Date-Time: Date-Time Tests - Leaf: Leaf Definition Test and Leaf Response Test - SubTemporalValue: SubTemporalValue Definition Test and SubTemporalValue Response Test Execute requests with combinations of the `"datetime"` and `"leaf"` (or `"subTemporalValue"`) query parameters and verify that only features are returned that match both selection criteria.

ABSTRACT TEST A.33

IDENTIFIER	`/conf/movingfeatures/tproperty-get-success`
REQUIREMENT	Requirement 44: `/req/movingfeatures/tproperty-get-success`
TEST PURPOSE	Validate that the **TemporalProperty** complies with the required structure and contents.
TEST METHOD	1. Validate that 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 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.3.7.2. HTTP POST Operation

ABSTRACT TEST A.34	
IDENTIFIER	`/conf/movingfeatures/tproperty-post`
REQUIREMENTS	Requirement 40: `/req/movingfeatures/tproperty-mandatory` Requirement 42: `/req/movingfeatures/tproperty-post` Requirement 45: `/req/movingfeatures/tproperty-post-success` Requirement 47: `/req/movingfeatures/tpvalue-mandatory`
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 the body of a POST request, for all supported media types, using the resources and tests identified in Table A.12 and Table 12 3. Validate that the request body complies with 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 Abstract test A.35: `/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.35	
IDENTIFIER	`/conf/movingfeatures/tproperty-post-success`
REQUIREMENT	Requirement 45: `/req/movingfeatures/tproperty-post-success`
TEST PURPOSE	Validate that the response of the `{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

A.3.7.3. HTTP DELETE Operation

ABSTRACT TEST A.36	
IDENTIFIER	`/conf/movingfeatures/tproperty-delete`
REQUIREMENTS	Requirement 43: `/req/movingfeatures/tproperty-delete` Requirement 46: `/req/movingfeatures/tproperty-delete-success`
TEST PURPOSE	Validate that the **TemporalProperty** 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}/tproperties/{tpropertyName}` 3. Validate the contents of the response using Abstract test A.37: `/conf/movingfeatures/tproperty-delete-success`

ABSTRACT TEST A.37	
IDENTIFIER	`/conf/movingfeatures/tproperty-delete-success`
REQUIREMENT	Requirement 46: `/req/movingfeatures/tproperty-delete-success`
TEST PURPOSE	Validate that the response of the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tpropertyName}` 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.3.8. TemporalPrimitiveValue

A.3.8.1. HTTP DELETE Operation

ABSTRACT TEST A.38	
IDENTIFIER	`/conf/movingfeatures/tpvalue-delete`
REQUIREMENTS	Requirement 48: `/req/movingfeatures/tpvalue-delete` Requirement 49: `/req/movingfeatures/tpvalue-delete-success`
TEST PURPOSE	Validate that the **TemporalPrimitiveValue** 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}/tproperties/{tpropertyName}/{tValueId}` 3. Validate the contents of the response using Abstract test A.39: `/conf/movingfeatures/tpvalue-delete-success`

ABSTRACT TEST A.39	
IDENTIFIER	`/conf/movingfeatures/tpvalue-delete-success`
REQUIREMENT	Requirement 49: `/req/movingfeatures/tpvalue-delete-success`
TEST PURPOSE	Validate that the response of the `{root}/collections/{collectionId}/items/{mFeatureId}/tproperties/{tpropertyName}/{tValueId}` 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.3.9. Parameters

A.3.9.1. Parameter Leaf

ABSTRACT TEST A.40	
IDENTIFIER	`/conf/movingfeatures/param-leaf-definition`
REQUIREMENT	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.41	
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 has a `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.3.9.2. Parameter SubTrajectory

ABSTRACT TEST A.42	
IDENTIFIER	`/conf/movingfeatures/param-subtrajectory-definition`
REQUIREMENT	Requirement 12: `/req/movingfeatures/param-subtrajectory-definition`
TEST PURPOSE	Validate that the `subTrajectory` query parameter is constructed correctly.
TEST METHOD	• Verify that the `subTrajectory` query parameter complies with the definition (using an OpenAPI Specification 3.0 fragment) • If the `subTrajectory` parameter is “true”: 1. Verify that the `datetime` parameter is a bounded interval with a start time and end time. 2. Verify that the `leaf` parameter is not used if it can be used.

ABSTRACT TEST A.43	
IDENTIFIER	`/conf/movingfeatures/param-subtrajectory-response`
REQUIREMENTS	Requirement 12: `/req/movingfeatures/param-subtrajectory-definition` Requirement 13: `/req/movingfeatures/param-subtrajectory-response`
TEST PURPOSE	Validate that the `subTrajectory` query parameter is processed correctly.
TEST METHOD	If the `subTrajectory` parameter is "true", DO FOR each temporal primitive geometry resource: 1. Calculate a temporal geometry coordinate with the ***subTrajectory*** query at a time interval (new start time and new end time) included in the `datetime` parameter, using interpolated trajectory according to the `interpolation` property 2. Verify that the calculated temporal geometry coordinate intersects the interpolated trajectory according to the `interpolation` property, using the time interval value defined by the `datetime` parameter 3. If the `bbox` parameter is not empty, verify that the calculated temporal geometry coordinate intersects the bounding box with the `bbox` parameter

A.3.9.3. Parameter SubTemporalValue

ABSTRACT TEST A.44	
IDENTIFIER	`/conf/movingfeatures/param-subtemporalvalue-definition`
REQUIREMENT	Requirement 34: `/req/movingfeatures/param-subtemporalvalue-definition`
TEST PURPOSE	Validate that the `subTemporalValue` query parameter is constructed correctly.
TEST METHOD	• Verify that the `subTemporalValue` query parameter complies with the definition (using an OpenAPI Specification 3.0 fragment) • If the `subTemporalValue` parameter is "true": 1. Verify that the `datetime` parameter is a bounded interval with a start time and end time. 2. Verify that the `leaf` parameter is not used if it can be used.

ABSTRACT TEST A.45	
IDENTIFIER	`/conf/movingfeatures/param-subtemporalvalue-response`
REQUIREMENTS	Requirement 34: `/req/movingfeatures/param-subtemporalvalue-definition` Requirement 35: `/req/movingfeatures/param-subtemporalvalue-response`
TEST PURPOSE	Validate that the `subTemporalValue` query parameter is processed correctly.

ABSTRACT TEST A.45

TEST METHOD	If the `subTemporalValue` parameter is "true", DO FOR each temporal property resource: 1. Calculate a temporal property value with the ***subTrajectory*** query at a time interval (new start time and new end time) included in the `datetime` parameter, using interpolated trajectory according to the `interpolation` property 2. Verify that the calculated temporal property value intersects the interpolated trajectory according to the `interpolation` property, using the time interval value defined by the `datetime` parameter

B

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

B

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

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 class diagram 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.

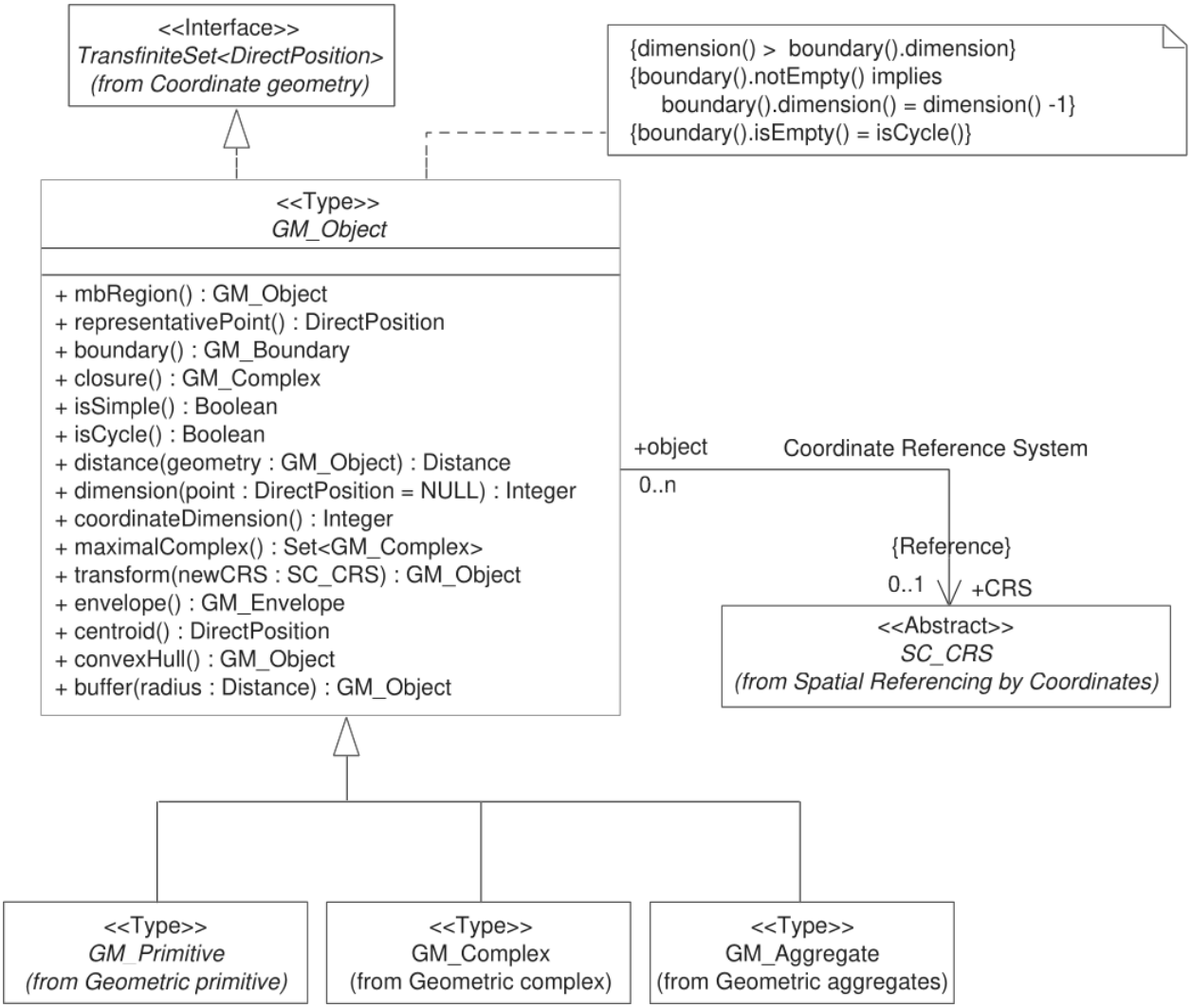

Figure B.1 — GM_Object from ISO 19107:2003 figure 6

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 class diagram 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.

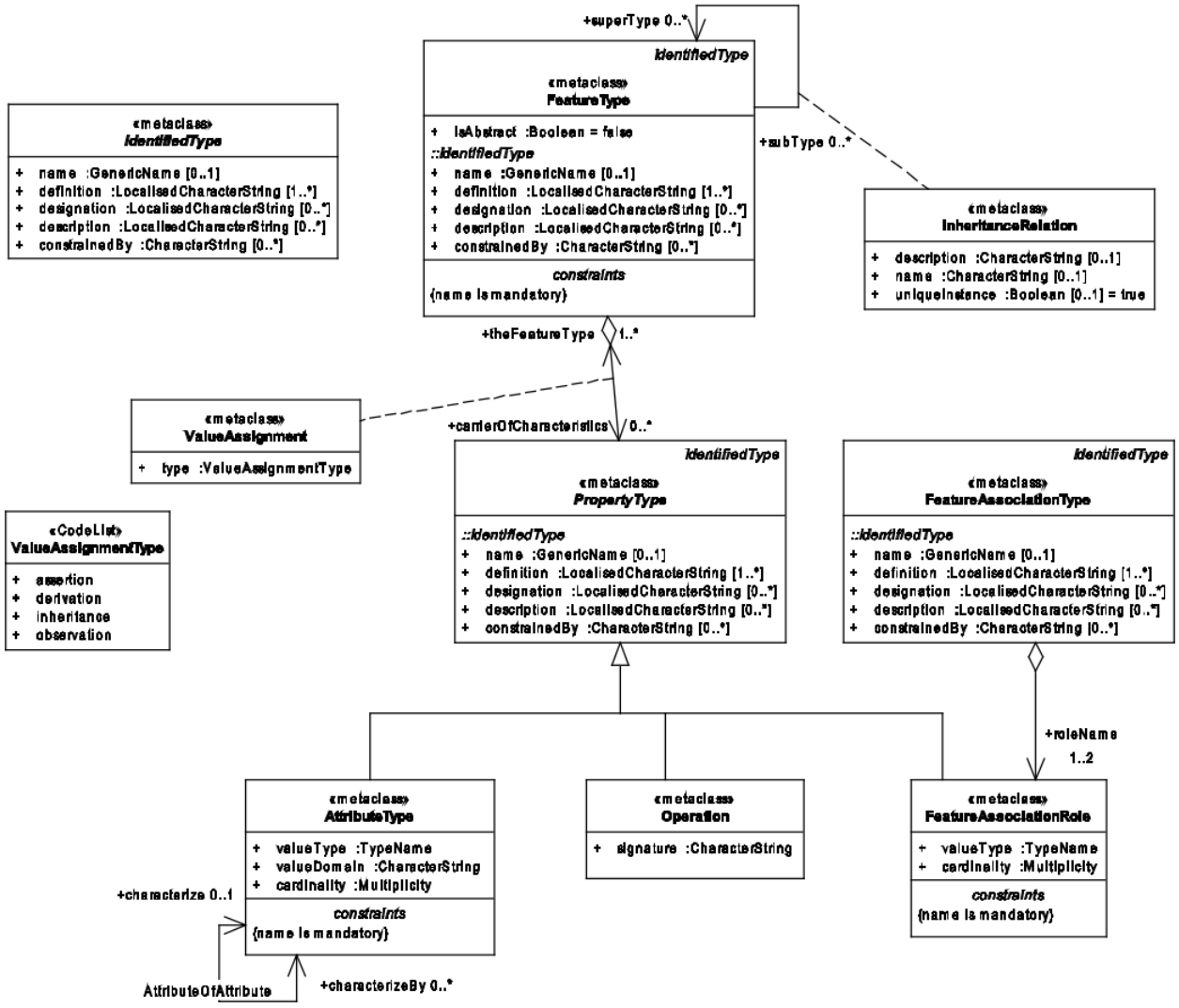

Figure B.2 — General Feature Model from ISO 19109:2009 figure 5

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 class diagram below shows the objects used for querying a subset based on spatial operations such as “contains” or “intersects”.

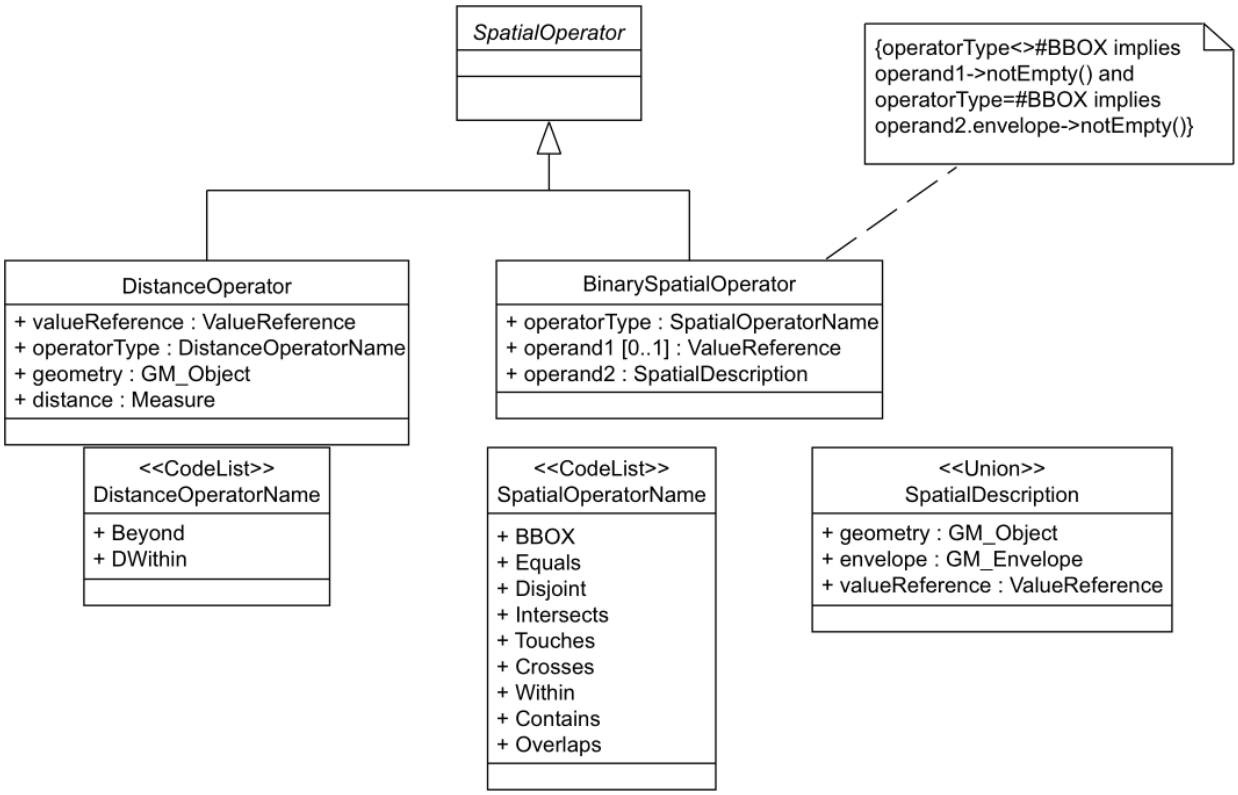

Figure B.3 — Spatial operators from ISO 19143 figure 6

B.1.5. OGC API — Features — Part 1: Core

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

B.1.6. OGC API — Features — Part 2: Coordinate Reference Systems by Reference

The OGC API — Features — Part 2: Coordinate Reference Systems by Reference Standard specifies an extension to the OGC API — Features — Part 1: Core Standard that defines the behavior of a server that supports the ability to present geometry valued properties in a response document in one from a list of supported Coordinates Reference Systems (CRS). This part extends the core capabilities specified in Part 1: Core with the ability to use coordinate reference system identifiers other than the defaults defined in the core.

B.1.7. OGC API — Features — Part 3: Filtering

The OGC API — Features — Part 3: Filtering Standard (OGC 19-079r2) extends the Web API defined in OGC API — Features — Part 1: Core 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 class diagram below shows how the `MF_Trajectory` type extends the "static" types from ISO 19107.

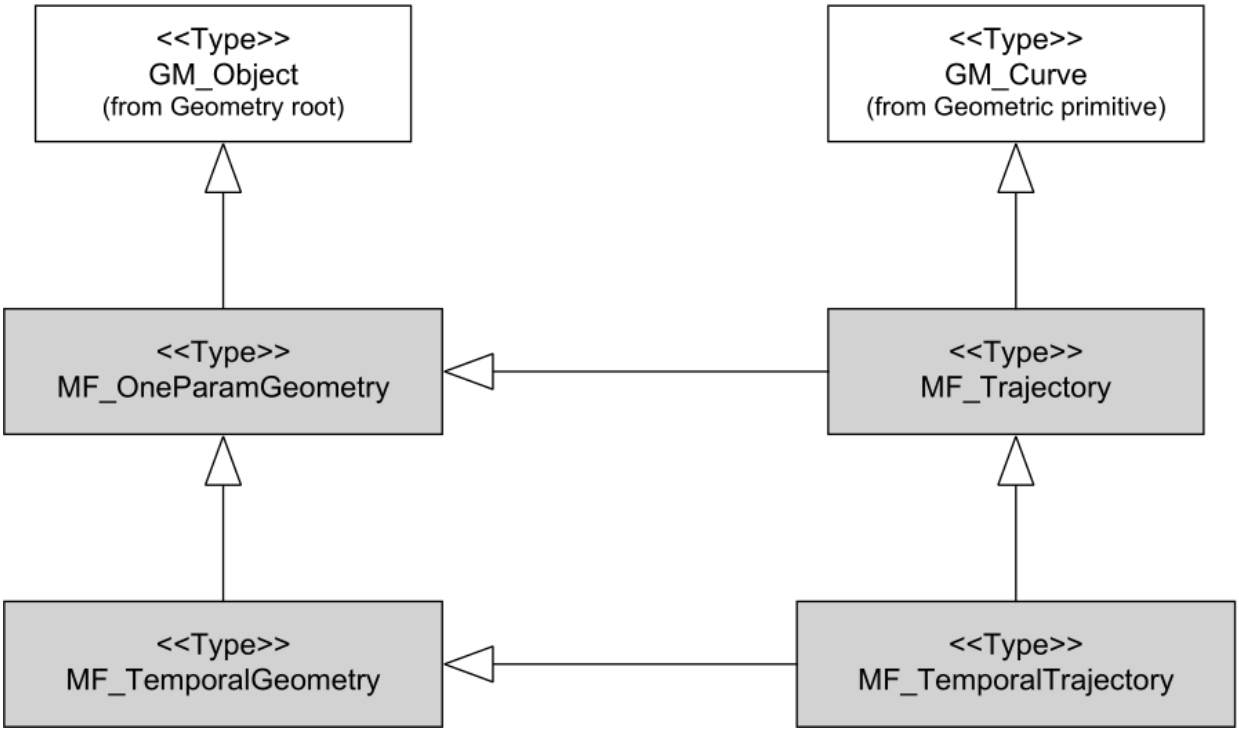

Figure B.4 — Trajectory type from ISO 19141 figure 3

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.

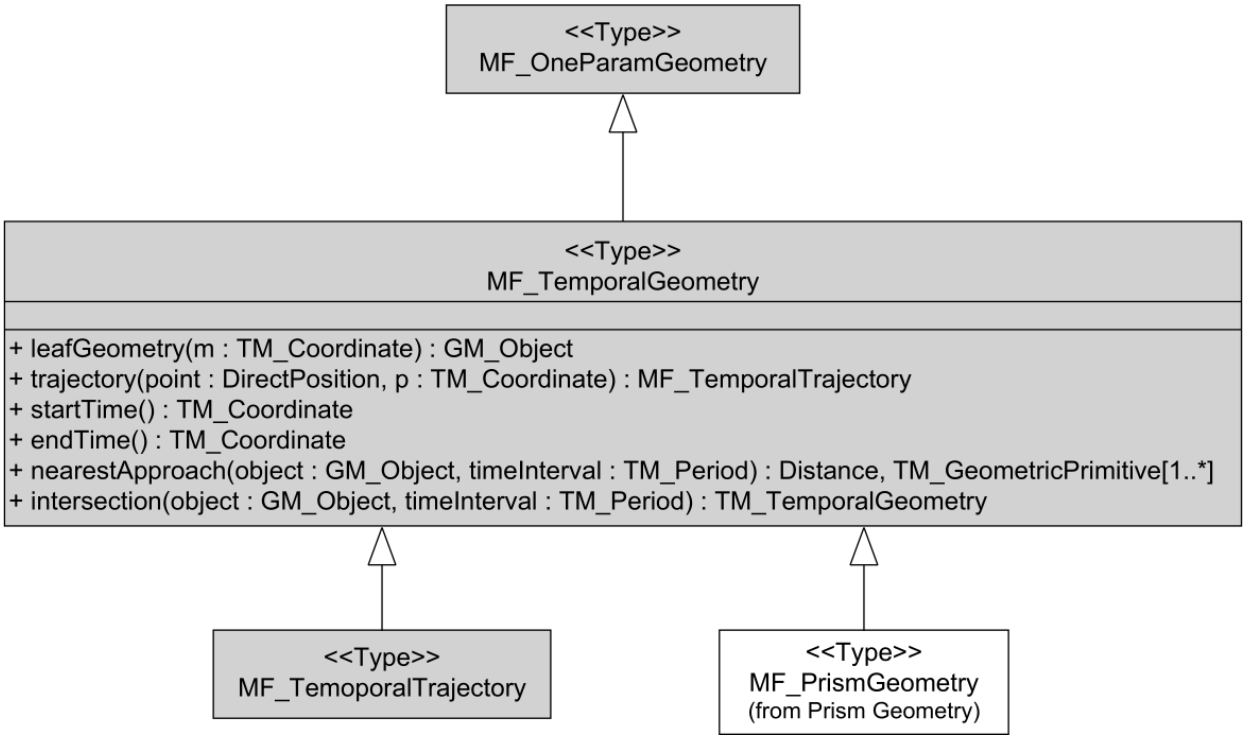

Figure B.5 — Temporal geometry from ISO 19141 figure 6

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 values change over time. This extension to the above *General Feature Model* is shown below:

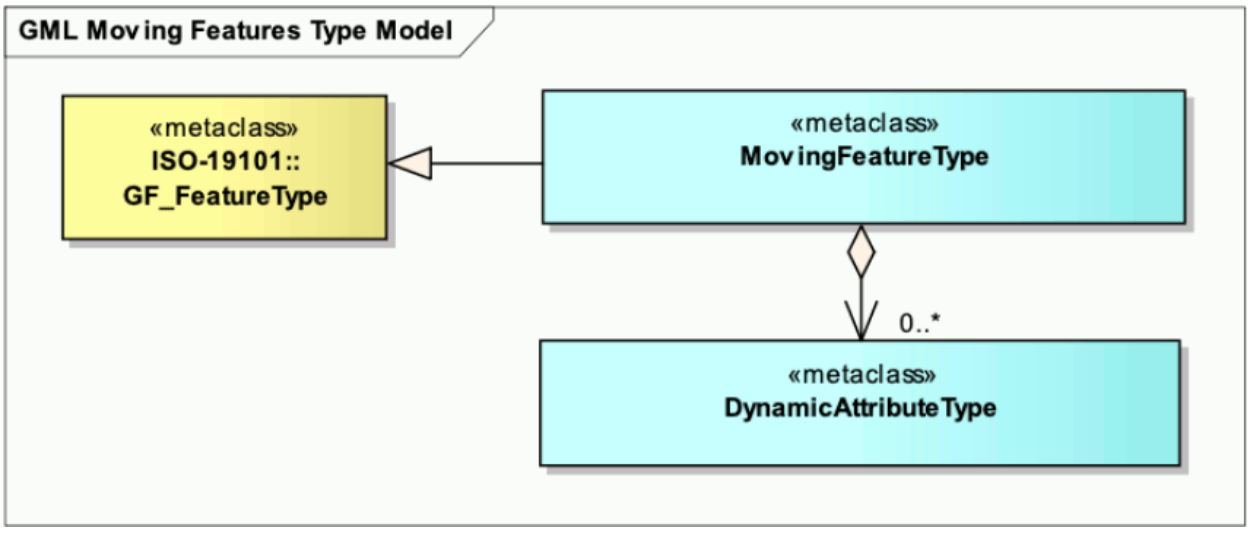

Figure B.6 — Dynamic attribute from OGC 18-075 figure 3

B.2.3. Moving Features JSON encoding (OGC 19-045r3)

The OGC 19-045r3 *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 — Revision history

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-09-14	0.1	Taehoon Kim, Kyoung-Sook Kim, and Martin Desruisseaux	all	first draft version
2022-03-01	0.2	Taehoon Kim, Kyoung-Sook Kim	all	revised sections related to resources to add CRUD operations
2022-10-09	0.3	Taehoon Kim, Kyoung-Sook Kim	all	added TemporalGeometry Query resources
2023-02-21	0.9	Taehoon Kim, Kyoung-Sook Kim, Mahmoud, and Esteban	all	finalize draft version
2023-05-19	0.9.9	Taehoon Kim, Kyoung-Sook Kim, Mahmoud, and Esteban	all	finalize draft version before OAB
2023-07-10	1.0.draft	Taehoon Kim, Kyoung-Sook Kim	all	finalize a draft version before public comment
2024-02-14	1.0.draft	Taehoon Kim, Kyoung-Sook Kim	all	revised by public comments
2024-07-31	1.0	Taehoon Kim, Kyoung-Sook Kim	all	final revision before publication

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/