Open Geospatial Consortium

Submission Date: 2017-01-17

Approval Date: 2017-03-24

Publication Date: 2017-06-15

External identifier of this OGC® document: http://www.opengis.net/doc/BP/mf-json/1.0

Internal reference number of this OGC® document: OGC 16-140r1

Version: 1.0

Category: OGC® Best Practice

Editor: Kyoung-Sook KIM, Hirotaka OGAWA

OGC Moving Features Encoding Extension - JSON

Copyright notice

Warning

This document defines an OGC Best Practice on a particular technology or approach related to an OGC standard. This document is **not** an OGC Standard and may not be referred to as an OGC Standard. This document is subject to change without notice. However, this document is an **official** position of the OGC membership on this particular technology topic.

Document type: OGC® Best Practice

Document subtype: Not applicable

Document stage: Approved for public release

Document language: English

License Agreement

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

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

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

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

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and

shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

Contents

i. Abstract

This document proposes a JavaScript Object Notation (JSON) encoding representation of movement of geographic features as an encoding extension of OGC Moving Features ([OGC 14-083r2] and [OGC 14-084r2]). A moving feature, typically a vehicle and pedestrian, can be expressed as a temporal geometry whose location continuously changes over time and contains dynamic non-spatial attributes whose values vary with time. This Best Practice describes how to share moving feature data based on JSON and GeoJSON (a JSON format for encoding geographic data structures). In addition, this document provides an example of RESTful approaches as a Feature Service Interface that has the potential for simplicity, scalability, and resilience with respect to exchange of moving feature data across the Web.

ii. Keywords

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

ogcdoc, OGC document, moving features, encoding, JSON, REST

iii. Preface

This Best p\Practice document follows the abstract data model defined in ISO 19141:2008 [ISO 19141:2008] with the encoding method referring to the IETF GeoJSON Format [IETF RFC 7946]. OGC Moving Features Encoding standards XML Core [OGC 14 083r2] and Simple CSV [OGC 14 084r2] have focused on the movement of 0-dimensional geometric primitives (Points), called trajectories. However, this document covers the movements of 0-dimensional Points, 1-dimensional curve LineStrings, and 2-dimensional surface Polygons to support application requirements such as disaster risk management, traffic information services, and geo-fencing services.

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. Submitting organizations

The following organization has submitted this Document to the Open GeoSpatial Consortium, Inc.:

- Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology

v. Submitters

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

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

vi. Future Work

Among the topics for future development are the following items:

- Testing RESTful API implementation;

- Moving features in an indoor space; and

- Representation of three-dimensional moving features.

1. Scope

This Best Practice provides a format for encoding moving features using JSON [IETF RFC7159] and an example of interfaces for the RESTful service of moving features. The Moving Feature JSON encoding defined in this document is an alternative to the OGC® Moving Features Encodings XML [OGC 14 083r2] and Simple Comma Separated Values (CSV) [OGC 14 084r2] standards. The data format of Moving Feature JSON is correspondingly designed to implement the abstract model of moving features as defined in ISO 19141:2008 [ISO 19141:2008] as shown in the Figure 1 with the concepts of foliation, prism, trajectory, and leaf.
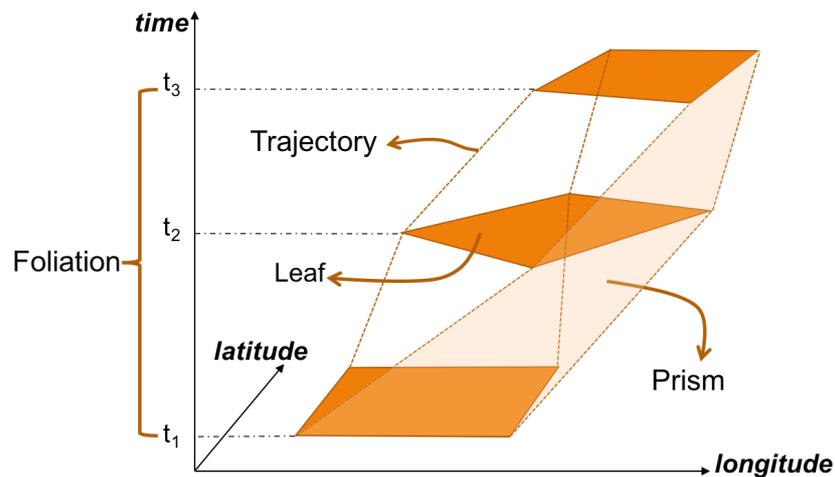
Figure 1. Conceptual Model of a Moving Feature: Foliation, Prism, Leaf, and Trajectory

In the illustration above, a 2D rectangle moves and rotates. Each representation of the rectangle at a given time is a leaf. The path traced by each corner point of the rectangle (and by each of its other points) is a trajectory. The set of points contained in all of the leaves, and in all of the trajectories, forms a prism. The set of leaves also forms a foliation, meaning that there is a complete and separate representation of the geometry of the feature for each specific time. The prism of the moving feature can be viewed as a bundle of trajectories of points on the local engineering representation of the feature's geometry.

In general, Moving Feature JSON encoding applies to representations and formats of GeoJSON [IETF RFC 7946]; however, new terms are added to specify dynamic attributes of moving features:

- *Temporal geometric objects* whose location changes over time: movements of 0-dimensional, 1-dimensional, 2-dimensional geometric primitives, and their collections; and

- *Dynamic non-spatial attributes* whose value varies with time: changes of a quantity.

This document also describes interfaces for accessing moving features based on the data format:

- *RESTful APIs* for handling moving feature data over HTTP: Create, Read, Update, and Delete (CRUD) operations.

This Best Practice document has left many issues out of its scope. Design issues with complex interpolation methods and service capabilities were not considered.

2. Conformance

Not applicable for this Best Practice document.

3. References

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

- OGC: OGC 06-121r9 (http://www.opengeospatial.org/standards/common), *OGC Web Services Common Standard*, version 2.0.0, 2010.

- OGC: OGC 14-083r2 (http://www.opengeospatial.org/standards/movingfeatures), *OGC Moving Features Encoding Part I: XML Core*, 2015.

- OGC: OGC 14-084r2 (http://www.opengeospatial.org/standards/movingfeatures), *OGC Moving Features Encoding Extension: Simple Comma Separated Values (CSV)*, 2015.

- OGC: OGC 16-120r3 (http://www.opengeospatial.org/standards/movingfeatures), *OGC Moving Features Access*, 2017.

- OGC: OGC 15-078r6 (http://www.opengeospatial.org/standards/sensorthings), *OGC SensorThings API Part 1: Sensing*, 2016.

- ISO: ISO/IEC Directives (http://isotc.iso.org/livelink/livelink?func=ll&objId=4230456&objAction=browse&sort=subtype), *Part 2. Rules for the structure and drafting of International Standards*

- ISO: ISO 8601:2004 (http://www.iso.org/iso/catalogue_detail?csnumber=40874), *Data elements and interchange formats - Information interchange - Representation of dates and time*, 2004.

- ISO: ISO 19101:2014 (http://www.iso.org/iso/catalogue_detail?csnumber=59164), *Geographic information — Reference model — Part 1: Fundamentals*, 2014.

- ISO: ISO 19103:2015 (http://www.iso.org/iso/catalogue_detail?csnumber=56734), *Geographic information — Conceptual schema language*, 2015.

- ISO: ISO 19107:2003 (http://www.iso.org/iso/catalogue_detail?csnumber=26012) *Geographic Information □- Spatial schema*, 2003.

- ISO: ISO 19119:2006 (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=59221), *Geographic information - Services*, 2006.

- ISO: ISO 19141:2008 (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=41445), *Geographic information - Schema for moving features*, 2008.

- IETF: IETF RFC 3986 (https://www.ietf.org/rfc/rfc3986.txt), *Uniform Resource Identifier (URI): Generic Syntax*, 2005.

- IETF: IETF RFC 2616 (https://www.ietf.org/rfc/rfc2616.txt), *Hypertext Transfer Protocol — HTTP/1.1*, 1999.

- IETF: IETF RFC 7159 (https://www.ietf.org/rfc/rfc7159.txt), *The JavaScript Object Notation (JSON) Data Interchange Format*, 2014.

- IETF: IETF RFC 7464 (https://www.ietf.org/rfc/rfc7464.txt), *JavaScript Object Notation (JSON) Text Sequences*, 2015.

- IETF: IETF RFC 7946 (https://www.ietf.org/rfc/rfc7946.txt), *The GeoJSON Format*, 2016.

- OASIS: OData-Part1 (http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html), *OData Version 4.0. Part 1: Protocol Plus Errata 03.* Edited by Michael Pizzo, Ralf Handl, and Martin Zurmuehl. OASIS Standard incorporating Approved Errata 03, 2016.

Additionally the following informative documents are addressed:

- OGC: OGC 15-052r1, *OGC Testbed 11 REST Interface Engineering Report*, 2016.

4. Terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 06-121r9], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular:

- SHALL is the verb form used to indicate a requirement to be strictly followed to conform to this specification, from which no deviation is permitted

- MAY ☐is the verb form used to indicate an action permissible within the limits of this specification

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

4.1. API

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

4.2. client

software component that can invoke an operation from a server
[OGC 06-121r9]

4.3. dynamic attributes

characteristic of a feature in which its value varies with time

4.4. feature

abstraction of real world phenomena
[ISO 19101:2014]

4.5. foliation

one parameter set of geometries such that each point in the prism of the set is in one and only one trajectory and in one and only one leaf
[ISO 19141:2008]

4.6. geometric object

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

4.7. geometric primitive

geometric object representing a single, connected, homogeneous element of space
[ISO 19107:2003]

4.8. interface

named set of operations that characterize the behaviour of an entity
[ISO 19119:2006]

4.9. leaf

<one parameter set of geometries> geometry at a particular value of the parameter
[ISO 19141:2008]

4.10. moving feature

feature whose location changes over time
[ISO 19141:2008]
NOTE   Its base representation uses a local origin and local coordinate vectors of a geometric object at a given reference time.

4.11. one parameter set of geometries

function f from an interval t $\in$ [a, b] such that f(t) is a geometry and for each point P $\in$ f(a) there is a one parameter set of points (called the trajectory of P) P(t):[a,b] $\rightarrow$ P(t) such that P(t) $\in$ f(t)
[ISO 19141:2008]
EXAMPLE   A curve C with constructive parameter t is a one parameter set of points c(t).

4.12. period

one-dimensional geometric primitive representing extent in time
[ISO 19141:2008]

4.13. prism

<one parameter set of geometries> set of points in the union of the geometries (or the union of the trajectories) of a one parameter set of geometries
[ISO 19141:2008]

4.14. request

invocation of an operation by a client
[OGC 06-121r9]

4.15. resource

any addressable unit of information or service
[IETF RFC 3986]

4.16. response

result of an operation, returned from a server to a client
[OGC 06-121r9]

4.17. server

a particular instance of a service
[OGC 06-121r9]

4.18. service

distinct part of the functionality that is provided by an entity through interfaces
[ISO 19119:2006]

4.19. temporal geometry

one parameter set of geometries in which the parameter is time

4.20. trajectory

path of a moving point described by a one parameter set of points
[ISO 19141:2008]

5. Conventions

This section provides details and examples for conventions used in the document.

All examples in this document illustrated by gray or orange boxes are informative only.

5.1. JSON notation

The notation of JSON in this document is based on the specification of [RFC 7159].

The ordering of the members of any JSON object must be considered irrelevant. Some examples use a JavaScript single line comment (//) and an ellipsis (…) as placeholder notation for a specific JSON instance. Whitespace is used in the examples inside this document to help illustrate the data structures, but is not required. Unquoted whitespace is not significant in JSON.

5.2. UML notation

Unified Modeling Language (UML) static structure diagrams appearing in this document are used as described in Subclause 5.2 of OGC Web Services Common [OGC 06-121r9].

5.3. Abbreviated terms

The following symbols and abbreviated terms are used in this best practice paper:

API	Application Program Interface
CRS	Coordinate Reference Systems
CRUD	Create, Read, Update, Delete
CSV	Comma Separated Values
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
REST	Representational State Transfer

UML	Unified Modeling Language
URI	Uniform Resource Identifiers
URL	Uniform Resource Locators
WKT	Well Known Text
XML	Extensible Markup Language
1D	One Dimensional
2D	Two Dimensional

6. JSON Encoding

This clause specifies the data format for encoding moving features by using JSON objects.

6.1. Overview

Moving Features JSON (MF-JSON) defines new fields by extending "foreign members" of GeoJSON (https://www.ietf.org/rfc/rfc7946.txt) [IETF RFC 7946]. MF-JSON provides several types of JSON objects to represent geographical movements, dynamic properties, and spatiotemporal extents of moving features, based on the two reference systems of World Geodetic System 1984 (WGS84) and Coordinated Universal Time (UTC).

> 🛈 In IETF GeoJSON format (https://www.ietf.org/rfc/rfc7946.txt), types are not extensible. GeoJSON allows only the fixed types: FeatureCollection, Feature, Point, LineString, MultiPoint, Polygon, MultiLineString, MultiPolygon, and GeometryCollection. Despite compliant GeoJSON not permitting extension of types, this document extends new types for representing moving features with time-varying geometries and properties.

Figure 2 compares the difference of members between GeoJSON and MF-JSON.

GeoJSON

```
{
    "type": "Feature",
    "geometry": ··· ,
    "properties": ··· ,
    "bbox": ···
}
```

MF-JSON

```
{
    "type": "MovingFeature",
    "temporalGeometry": ··· ,
    "temporalProperties": ··· ,
    "stBoundedBy": ··· ,
    "properties": ···
}
```
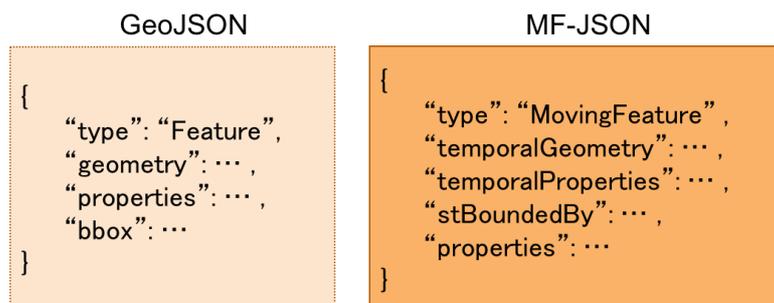
Figure 2. JSON Objects in the Moving Features JSON format

Example 6.1:

JAVASCRIPT

```javascript
{
    "type": "MovingFeature",  //(REQUIRED) Moving Feature Type
    "temporalGeometry": { //(REQUIRED) temporal geometric object, extended from
'geometry'
        "type": "MovingPoint", // a geometry type to represent a trajectory
        "coordinates": [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0] ],
        "datetimes": ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z", "2011-07-15T01:01:01Z"],
        "interpolations": ["Linear"] // a pre-defined interpolation method between two
consecutive instants in the "datetimes" field
    },
    "temporalProperties": [  //(OPTIONAL) dynamic non-spatial attributes, extended from
'properties'
        { // a group of temporal properties that are measured at the same times
            "datetimes": ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z"],
            "length": {
              "uom": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length", // a URI
denoting a unit-of-measure
              "values": [1.0, 2.4, 1.0], // the array of values for "length", with the
same number of elements as "datetimes"
              "interpolations": ["Stepwise"],
              "description": "description1" //(OPTIONAL)
            },
            "message":{
              "uom": "text", // a predefined unit for a string value
              "values": ["A", "B", "C"], // the array of values for "message", with the
same number of elements as "datetimes"
              "interpolations": ["Discrete"],
              "description": "description2" //(OPTIONAL)
            }
        },
        {
            "datetimes" : ["2011-07-14T22:02:01Z", "2011-07-15T01:11:22Z"],
            "discharge" : {
              "uom" : "m^3/s", // a symbol from UCUM
              "values" : [3.0, 4.0],
              "interpolations": [ // two user-defined interpolations having two sub-time
intervals
                  {
                      "coefficients": [1.0, 3.0, 4.1], // v = 1.0t^2 + 3.0t + 4.1
                      "period": {
                        "begin": "2011-07-14T22:02:01Z",
                        "end" : "2011-07-14T23:11:14Z"
                      }
                  },
                  {
                      "coefficients": [5.0, 8.2], // v = 5.0t + 8.2
                      "period": {
                        "begin": "2011-07-14T23:11:14Z",
                        "end" : "2011-07-15T01:11:22Z"
                      }
                  }
              ],
              "description": "it has two user-defined interpolations" //(OPTIONAL)
            }
        }
```

```
    ],
    "stBoundedBy": {  //(OPTIONAL) spatiotemporal bounding box to include the moving
feature
        "bbox": [100.0, 0.0, 101.0, 1.0],
        "period": {
            "begin": "2011-07-14T22:01:00Z",
            "end" : "2011-07-15T21:14:00Z"
        }
    },
    "properties": {  //(OPTIONAL) static non-spatial attributes regardless of time: the
same representation of GeoJSON
        "name": "bus1",
        "state":"test1"
    }
}
```

6.2. Moving Features

In the MF-JSON format, two moving feature types are added as follows:

- **MovingFeature**: a JSON object to represent a moving feature instance, having two REQUIRED members of `"type"` and `"temporalGeometry"`, and three OPTIONAL members of `"temporalProperties"`, `"stBoundedBy"`, and `"properties"` depending on the application requirements.

  JAVASCRIPT

  ```
  {
      "type": "MovingFeature",    //(REQUIRED)
      "temporalGeometry": {...},    //(REQUIRED)
      "temporalProperties": [...],  //(OPTIONAL)
      "stBoundedBy": {...},         //(OPTIONAL)
      "properties": {...}           //(OPTIONAL)
  }
  ```

  - The value of the `"type"` member SHALL be the string of `"MovingFeature"`.

  - The value of the `"temporalGeometry"` member SHALL be a JSON object as described in Temporal Geometries. An instance of `"MovingFeature"` SHALL only one `"temporalGeometry"` member.

  - The value of the `"temporalProperties"` member SHALL be a JSON array as described in Temporal Properties.

  - The value of the `"stBoundedBy"` member SHALL be a JSON object as described in Spatiotemporal Bounding Box, representing the spatiotemporal bounding box of the `"temporalGeometry"` instance.

  - The value of the `"properties"` member is an object (any JSON object or a JSON null value).

- **MovingFeatureCollection**: a JSON object to represent a collection of moving feature instances, having two REQUIRED members of `"type"` and `"features"` and one OPTIONAL member of `"stBoundedBy"`.

```javascript
{
    "type": "MovingFeatureCollection",  //(REQUIRED)
    "features": [                       //(REQUIRED)
        {
            "type": "MovingFeature",
            "temporalGeometry": {...},
            "temporalProperties": [...],
            ...
        },
        {
            "type": "MovingFeature",
            "temporalGeometry": {...},
            "temporalProperties": [...],
            ...
        }
    ],
    "stBoundedBy": {...}                 //(OPTIONAL)
}
```

- The value of the `"type"` member SHALL be the string of `"MovingFeatureCollection"`.

- The value of the `"features"` member SHALL be a JSON array of moving feature instances as described in Temporal Geometries.

- The value of the `"stBoundedBy"` member SHALL be a JSON object as described in Spatiotemporal Bounding Box, representing the spatiotemporal bounding box to cover all of the `"temporalGeometry"` instances in the `"features"` elements.

6.3. Temporal Geometries

The value of `"temporalGeometry"` member of a moving feature SHALL be a JSON object where the value of the `"type"` member is one of the following strings: `"MovingPoint"`, `"MovingLineString"`, `"MovingPolygon"`, `"MultiMovingPoint"`, `"MultiMovingLineString"`, `"MultiMovingPolygon"`, and `"MovingGeometryCollection"`. A temporal geometry is conceptualized as a prism of the set of points contained in all of the leaves (a foliation) and trajectories, representing the geographical movement of a moving feature. The temporal geometry is mathematically modeled as a mapping function from time to a geometric object: *temporalGeometry: timePosition → Geometry* (Point (http://geojson.org/geojson-spec.html#point), LineString (http://geojson.org/geojson-spec.html#linestring), Polygon (http://geojson.org/geojson-spec.html#polygon), MultiPoint (http://geojson.org/geojson-spec.html#multipoint), MultiLineString (http://geojson.org/geojson-spec.html#multilinestring), MultiPolygon (http://geojson.org/geojson-spec.html#multipolygon#polygon), or GeometryCollection (http://geojson.org/geojson-spec.html#geometry-collection)).

- **MovingPoint**: A temporal geometry represents the trajectory of a time-parameterized 0-dimensional geometric primitive (Point) representing a single geographic position at a time position (instant) within its temporal domain. Intuitively, this type depicts a set of curves in a spatiotemporal domain. A MovingPoint is used to express mf:AbstractTrajectory (http://docs.opengeospatial.org/is/14-083r2/14-083r2.html#78) in the OGC Moving Features standard. For example, the movement information of people, vehicles, or hurricanes can be shared by instances of the `"MovingPoint"` type.

- **MovingLineString**: A temporal geometry represents the prism of a time-parameterized 1-dimensional (1D) geometric primitive (LineString) whose leaf at a time position is 1-dimensional linear object in a particular time period. Intuitively this type depicts a set of surfaces in a spatiotemporal domain. For example, the movement information of weather fronts or traffic congestion on roads can be shared by instances of the `"MovingLineString"` type.

- **MovingPolygon**: A temporal geometry represents the prism of a time-parameterized 2-dimensional (2D) geometric primitive (Polygon) whose leaf at a time position is 2-dimensional polygonal object in a particular time period. Intuitively this type depicts a set of polyhedrons that are the convex hulls of two congruent polygons in a spatiotemporal domain. For example, the changes of flooding areas or the movement information of air pollution can be shared by instances of the `"MovingPolygon"` type.

- **MultiMovingPoint**: A temporal geometry represents a set of moving points.

- **MultiMovingLineString**: A temporal geometry represents a set of moving linestrings.

- **MultiMovingPolygon**: A temporal geometry represents a set of moving polygons.

- **MovingGeometryCollection**: Represents a collection of temporal geometries that have time-varing locations. Each element in the collection belongs to one of the above types.

This practice defines two encoding mthods for a temporal geometry instance: simple and collection form.

6.3.1. Simple Temporal Geometries

The simple form of temporal geometry instances is a JSON object with four REQUIRED members: `"type"`, `"coordinates"`, `"datetimes"`, and `"interpolations"`. The simple form represents the movement of one geometric primitive that comprises non-decomposed objects, i.e., a moving point, moving linestring, and moving polygon. If viewed in a 4-dimensional spatio-temporal coordinate system, the temporal geometry is a single continuum. A moving point, linestring, and polygon is a spatio-temporal curve, surface, and solid, respectively. Their JSON representation is as follows.

```javascript
                                                                    JAVASCRIPT
{
   ...,
   "temporalGeometry": {
     "type": "MovingPoint | MovingLineString | MovingPolygon",  // (REQUIRED) vbar | as
a means to select ONE type.
     "coordinates": [...],   //(REQUIRED)
     "datetimes" : [...],    //(REQUIRED)
     "interpolations": [...] //(REQUIRED)
   },
   ...
}
```

- "type" : A case-sensitive string that is one of "MovingPoint", "MovingLineString", and
  "MovingPolygon".

- "coordinates" : The object SHALL be a list of leaf geometric primitives (points, linestrings,
  polygons) at times. The number of elements is same as the "datetimes" ones with a
  temporal order. There is an one-to-one correspondence between the elements of
  "coordinates" and "datetimes" as a temporal sequence of pairs (g, t), where g is a leaf
  geometry and t is its sampling time.

Types	Formats of the "coordinates" object	Comments
MovingPoint	[[x1,y1(,z1)], [x2,y2(,z2)], ...]	a list of points at each leaf, increasing time order
MovingLineString	[[[x11,y11(,z11)], [x12,y12(,z12)], ...], [[x21,y21(,z21)], [x22,y22(,z22)], ...], ...]	a list of linestrings at each leaf, increasing time order
MovingPolygon	[[[[ox11,oy11(,oz11)], [ox12,oy12(,oz12)], ...], [[ix11,iy11(,iz11)],[ix12,iy12(,iz12)], ...],...], [[[ox21,oy21(,oz21)], [ox22,oy22(,oz22)], ...], [[ix21,iy21(,iz21)],[ix22,iy22(,iz22)], ...], ...], ...]	a list of polygons at each leaf, increasing time order

> [IETF GeoJSON format (https://www.rfc-editor.org/info/rfc7946)] A position is
> represented by an array of numbers, which must contain two or more elements.
> The first two elements are longitude/easting (x) and latitude/northing (y),
> precisely in that order and using decimal numbers. Altitude/elevation (z) MAY be
> included as an optional third element.

- "datetimes" : The object SHALL be a list of time instants encoded as a character string of [ISO 8601:2004](http://www.iso.org/iso/home/standards/iso8601.htm) date-time formatter in chronological order, which does not allow duplicates.

Types	Formats the "datetimes" object	Comments
DateTime (http://www.w3.org/TR/xmlschema11-2/#dateTime)	["yyyy-MM-dd'T'HH:mm:ss'Z'", "yyyy-MM-dd'T'HH:mm:ss'Z'", ...]	*a list of monotonic increasing instants*

- "interpolations" : The object SHALL be a JSON array of interpolation methods. Interpolation (http://mathworld.wolfram.com/Interpolation.html) is a method of finding new values for any function using the given set of values. Here, the interpolation object approximates geographic positions at non sampling time instants for constructing the trajectory or prism of a moving feature in a spatiotemporal domain. The unknown position at a particular time can be found using many interpolation methods. In this practice, there are two expressions for an instance of interpolation methods: Predefined Interpolation Methods and Interpolation Formulas.

[Predefined Interpolation Methods]

A predefined method SHALL be a case-sensitive string of one of "Discrete" , "Stepwise" , "Linear" , and "Spline" . The new position is differently derived by each method. For the predefined method, there is the restriction of the same number positions of all leaf geometries.

Types	Descriptions	Comments
Discrete		*There is no interpolation position between two successive positions.*
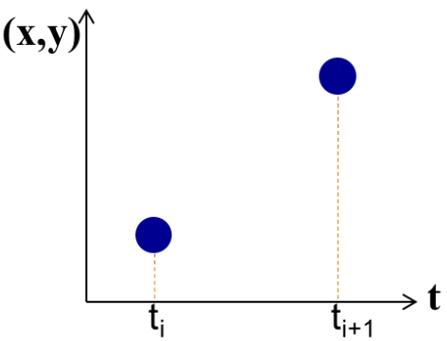
Types	Descriptions	Comments
Stepwise		*The interpolation position between two successive positions equals to the first position.*
Linear		*The new position is found from the linear interpolation formula with the two successive positions.* *Default*
Spline		*An interpolation position is derived from a cubic spline function on each interval between data positions.*
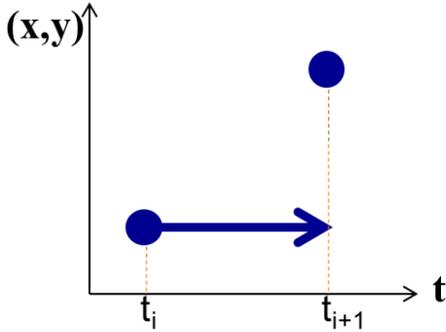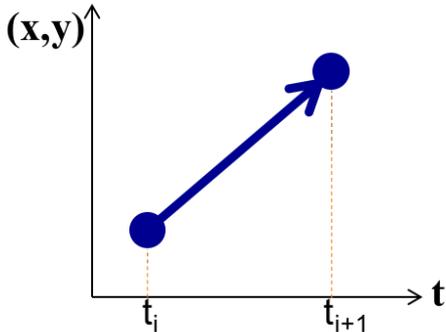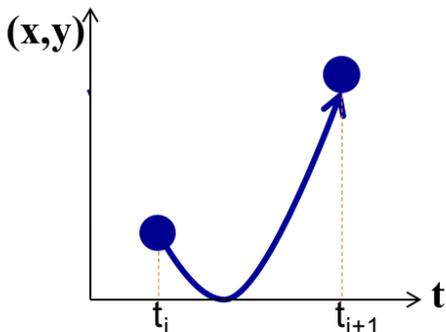
JAVASCRIPT

```javascript
{
    "type": "MovingPoint",
    "coordinates": [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0]],
    "datetimes": ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z"],
    "interpolations": ["Linear"] // an interpolation method during the period of
datetimes
}
```

If a moving feature needs different predefined interpolation methods for several sub-intervals of time during its lifespan, this practice recommends the use of user-defined interpolation formulas corresponding to each sub-interval.

[Interpolation Formulas]

A temporal geometry MAY have a several interpolation formulas within its temporal domain. An element of interpolation formulas SHALL be represented by two members of `"coefficients"` and `"period"`. The new geographical position at a given time position is derived from a `"coefficients"` instance as a multidimensional array of **polynomial interpolation formulas** of (x, y, z) coordinates during a particular time period. If a time position does not belong to any time period of the elements of formula, there is no interpolation position at that time. The order of arrays for the interpolation formula of a temporal position SHALL follow x (longitude), y (latitude), z (altitude) order. The time periods between any two elements of formulas only allows empty or 0-dimensional intersection. This practice converts the time position to a signed 64-bit integer(long) value that represents milliseconds for computing a value of each axis of coordinates at an arbitrary time position formatted by a character string of [ISO 8601:2004].

$$\left[x(t) = a_{n-1}t^{n-1} + a_{n-2}t^{n-2} + \ldots + a_0 t^0\right]$$

$$\left[y(t) = b_{n-1}t^{n-1} + b_{n-2}t^{n-2} + \ldots + b_0 t^0\right]$$

$$\left[z(t) = c_{n-1}t^{n-1} + c_{n-2}t^{n-2} + \ldots + c_0 t^0\right]$$

JAVASCRIPT

```javascript
{
    "type": "MovingPoint",
    "coordinates": ...,
    "datetimes": ...,
    "interpolations": [
     {
        "coefficients": [[1.0, 3.0, 4.1], [2.0, 2.1, 3.0]],
        // x = 1.0*t^2 + 3.0*t + 4.1 , y = 2.0*t^2 + 2.1*t + 3.0
        "period": {
            "begin": "2011-07-14T22:01:01Z",
            "end" : "2011-07-14T23:01:01Z"
        }
     },
     {
        "coefficients": [[4.0, 2.0], [1.0, 2.0]],
        // x = 4.0*t + 2.0 , y = 1.0*t + 2.0
        "period": {
            "begin": "2011-07-14T23:01:01Z",
            "end" : "2011-07-15T00:01:01Z"
        }
     }
    ]
}
```

<div align="right">JAVASCRIPT</div>

```javascript
{
    "type": "MovingLineString",
    "coordinates": ...,
    "datetimes": ...,
    "interpolations": [
     {
         "coefficients": [[[1.0, 3.0, 4.1], [2.0, 2.1, 3.0]], [[2.0, 1.0, 1.2], [1.0,
0.1, 2.0]]],
         // x1 = 1.0*t^2 + 3.0*t + 4.1 , y1 = 2.0*t^2 + 2.1*t + 3.0
         // x2 = 2.0*t^2 + 1.0*t + 1.2 , y2 = 1.0*t^2 + 0.1*t + 2.0
         "period": {
             "begin": "2011-07-14T22:01:01Z",
             "end" : "2011-07-14T23:01:01Z"
         }
     },
     {
         "coefficients": [[[4.0, 2.0], [1.0, 2.0]], [[2.0, 3.0], [2.0, 1.0]]],
         // x1 = 4.0*t + 2.0 , y1 = 1.0*t + 2.0
         // x2 = 2.0*t + 3.0 , y2 = 2.0*t + 1.0
         "period": {
             "begin": "2011-07-14T23:01:01Z",
             "end" : "2011-07-15T00:01:01Z"
         }
     }
    ]
}
```

6.3.2. Collection of Temporal Geometries

The collection form of temporal geometry instances is a JSON object with two REQUIRED fields:
`"type"` and `"members"`.

<div align="right">JAVASCRIPT</div>

```javascript
{
    ...
    "temporalGeometry": {
      "type": "MultiMovingPoint | MultiMovingLineString | MultiMovingPolygon |
MovingGeometryCollection",
      "members": [
        {
           // Simple temporal geometry instance
           "type": "MovingPoint | MovingLineString | MovingPolygon",
           "coordinates": [...],  // COORDINATES expression
           "datetimes" : [...],   // DATETIMES expression
           "interpolations": [...]   // INTERPOLATIONS expression
        }
      ]
    },
    ...
}
```

- `"type"` : A case-sensitive string that is one of `"MultiMovingPoint"`,
  `"MultiMovingLineString"`, `"MultiMovingPolygon"`, and `"MovingGeometryCollection"`.

- `"members"` : The object is encoded as a JSON array of instances of temporal geometry where each element is encoded in simple form.

  - **MultiMovingPoint**: The elements of the `"members"` object SHALL be instances of type `"MovingPoint"` . The leaf geometry at a time position is an instance of type "MultiPoint", which is the union of each leaf of moving point members at the same time.

  - **MultiMovingLineString**: The elements of the `"members"` object SHALL be instances of type `"MovingLineString"` . The leaf geometry at a time position is an instance of type "MultiLineString", which is the union of each leaf of moving linestring members at the same time.

  - **MultiMovingPolygon**: The element of the `"members"` object SHALL be instances of type `"MovingPolygon"` . The leaf geometry at a time position is an instance of type "MultiPolygon", which is the union of each leaf of moving polygon members at the same time.

  - **MovingGeometryCollection**: Each element of the `"members"` object can be an instance of different moving types. The leaf geometry at a time position is an instance of type `"GeometryCollection"` , which is the union of each leaf of any temporal geometries at the same time.

6.4. Temporal Properties

A moving feature can have more than zero time-varying properties, such as the velocity of vehicles or the wind speed of hurricanes. A temporal property represents a dynamic measure that the result of ascertaining the value of a characteristic of a moving feature that changes over time and/or location. Even though the value of temporal property depends on the spatiotemporal location, this document only considers the temporal dependencies of their changes of value.

> If a property has a static value, it is represented with the "properties" member as same as GeoJSON.

```javascript
{
    ...,
    "temporalProperties": [
      { // a collection of temporal properties which are measured at the same times
        "datetimes": [...],    //(REQUIRED) JSON Array of time instances in order, which
does not allow duplicates.
        "_property0_": {        // _property0_ whose name is defined by an application
          "uom": ...,              //(REQUIRED) a predefined string or URI
          "values": [...],         //(REQUIRED) a JSON Array of values
          "interpolations": [...],//(REQUIRED) a JSON Array of interpolation methods
          "description": "any string"   //(OPTIONAL)
        },
        "_property1_": {        // _property1_ whose name is defined by an application
          "uom": ...,              //(REQUIRED) a predefined string or URI
          "values": [...],         //(REQUIRED) a JSON Array of values
          "interpolations": [...],//(REQUIRED) a JSON Array of interpolation methods
          "description": "any string"   //(OPTIONAL) a JSON string
        },
        ...
      },
      { // another collection of dynamic properties which are measured at the same times
        "datetimes": [...],
        "_property2_": {...},
        "_property3_": {...}
      }
    ],
    ...
}
```

The `temporalProperties` is a JSON array of collections of temporal properties whose results are ascertained at the same times. A collection of temporal properties SHALL have one "datetimes" member and more than one *property* member whose name is defined by an application. However, the value of the *property* member SHALL be a JSON object that has the following fields:

- "uom" : A unit of measure is a quantity adopted as a standard of measurement [ISO 19103:2015]. The unit of a temporal property is represented as a URI denoting a unit-of-measure defined in a web resource or a predefined strings. This practice defines the following unit strings.

Unit strings	Descriptions
print symbols	From the Unified Code for Units of Measure (UCUM)[1]
null	The "values" member contains counting measures.
text	The "values" member contains any strings.
image	The "values" member contains Base64 strings converted from images.

- `"values"` : Each element of values is a string, number, null, or one of the literals: true and false. The number of elements is the same as the "datetimes" ones. There is an one-to-one correspondence between the elements of `"values"` of a *property* object and `"datetimes"` as a temporal sequence of pairs (v, t), where v is a value of measurement and t is its sampling time.

> 🛈   If the values of a temporal property are measured at different times of `"datetimes"` , each SHALL be represented as a new element in the JSON array.

- `"interpolations"` : The temporal property also needs to define an interpolation method like the temporal geometry. The object SHALL be a JSON array of interpolation methods whose instance is a pre-defined interpolation methods of `"Discrete"` , `"Stepwise"` , `"Linear"` (default), and `"Spline"` , or an interpolation formula used for polynomial interpolation in time.

JAVASCRIPT

```
{
  ...,
  "temporalProperties" : [
    {
      "datetimes" : [ "2017-03-13T01:00:00Z", "2017-03-13T02:00:00Z", "2017-03-
13T03:00:00Z", "2017-03-13T04:00:00Z", "2017-03-13T05:00:00Z", "2017-03-13T06:00:00Z",
"2017-03-13T07:00:00Z", "2017-03-13T08:00:00Z", "2017-03-13T09:00:00Z", "2017-03-
13T10:00:00Z", "2017-03-13T11:00:00Z", "2017-03-13T12:00:00Z", "2017-03-13T13:00:00Z",
"2017-03-13T14:00:00Z", "2017-03-13T15:00:00Z", "2017-03-13T16:00:00Z", "2017-03-
13T17:00:00Z", "2017-03-13T18:00:00Z", "2017-03-13T19:00:00Z", "2017-03-13T20:00:00Z",
"2017-03-13T21:00:00Z", "2017-03-13T22:00:00Z", "2017-03-13T23:00:00Z", "2017-03-
13T24:00:00Z" ],
      "NO2" : {
        "uom" : "ppm",
        "values" : [ 0.018, 0.013, 0.013, 0.014, 0.021, 0.034, 0.036, 0.047, 0.059,
0.052, 0.042, 0.031, 0.024, 0.02, 0.023, 0.022, 0.027, 0.025, 0.029, 0.03, 0.024, 0.02,
0.018, 0.016 ],
        "interpolations" : [ "Stepwise" ]
      },
      "NO" : {
        "uom" : "ppm",
        "values" : [ 0.001, 0.001, 0.001, 0.002, 0.002, 0.006, 0.012, 0.056, 0.085, 0.06,
0.039, 0.024, 0.013, 0.01, 0.009, 0.009, 0.009, 0.007, 0.007, 0.006, 0.005, 0.004, 0.003,
0.003 ],
        "interpolations" : [ "Linear" ]
      }
    },
    {
      "datetimes" : [ "2017-03-13T01:00:00Z", "2017-03-13T03:00:00Z", "2017-03-
13T04:00:00Z", "2017-03-13T05:00:00Z", "2017-03-13T06:00:00Z", "2017-03-13T07:00:00Z",
"2017-03-13T08:00:00Z", "2017-03-13T09:00:00Z", "2017-03-13T10:00:00Z", "2017-03-
13T11:00:00Z", "2017-03-13T12:00:00Z", "2017-03-13T13:00:00Z", "2017-03-13T14:00:00Z",
"2017-03-13T15:00:00Z", "2017-03-13T16:00:00Z", "2017-03-13T17:00:00Z", "2017-03-
13T18:00:00Z", "2017-03-13T19:00:00Z", "2017-03-13T20:00:00Z", "2017-03-13T21:00:00Z",
"2017-03-13T22:00:00Z", "2017-03-13T23:00:00Z", "2017-03-13T24:00:00Z" ],
      "CH4" : {
        "uom" : "ppmC",
        "values" : [ 1.97, 1.98, 1.97, 2.01, 2.19, 2.13, 2.06, 2.21, 2.14, 2.08, 2.04,
1.99, 1.97, 1.96, 1.95, 1.95, 1.96, 1.96, 1.97, 1.96, 1.95, 1.95, 1.95 ],
        "interpolations" : [ "Discrete" ]
      },
      "THC" : {
        "uom" : "ppmC",
        "values" : [ 2.09, 2.05, 2.05, 2.09, 2.33, 2.26, 2.22, 2.45, 2.35, 2.25, 2.18,
2.09, 2.04, 2.04, 2.03, 2.02, 2.03, 2.05, 2.08, 2.06, 2.03, 2.03, 2.03 ],
        "interpolations" : [  // The function is an example, no sense of working
          {
            "coefficients": [1.0, 3.0, 4.1], // v = at^2 + bt + c
            "period": {
                "begin": "2017-03-13T01:00:00Z",
                "end" : "2017-03-13T05:00:00Z"
            }
          },
          {
            "coefficients": [5.0, 8.0], // v = at + b
            "period": {
                "begin": "2017-03-13T05:00:00Z",
```

```
                    "end" : "2017-03-13T24:00:00Z"
                  }
               }
           },
         }
      ],
      ...
   }
```

- `"description"` : A temporal property can have an optional member to describe a short description.

6.5. Spatiotemporal Bounding Box

A moving feature may have a member named `"stBoundedBy"` , which indicates the boundary containing moving features in a spatiotemporal domain. To represent information on the coordinate range for moving features, this MF-JSON format follows GeoJSON's `"bbox"` field. The value of the bbox member is a 2*n array where n is the number of dimensions. The temporal boundary is a temporal period of `"begin"` and `"end"` expressed in ISO 8601:2004.

```javascript
 {
    ...,
    "stBoundedBy": {
        "bbox": [-10.0, -10.0, 10.0, 10.0],
        "period": {
          "begin": "1994-11-05T13:15:30Z",
          "end" : "1994-11-05T13:15:30Z"
        }
    },
    ...
 }
```

6.6. Application Domain Variables (Foreign Members)

MF-JSON uses annotations to represent foreign members which are not described in this document and whose semantics are dependent on a domain or application specific requirement. For this reason, MF-JSON defines foreign elements by extending the foreign member of GeoJSON. For a name/value pair of a foreign member, the name always starts with the at sign (@), such as `"@id"` , `"@context"` , and so on.

6.7. Discussions

Coordinate Reference System

The IETF GeoJSON format (https://www.ietf.org/rfc/rfc7946.txt) recommends a single coordinate reference system based on WGS84[2]. In this version of MF-JSON, CRSs are fixed to WGS84 for space and ISO 8601:2004 for time; however, the coordinate reference systems need to be indicated in the case of an application request. If the application requires an alternative CRS, the CRS of a GeoJSON object can be represented with its "crs" field as described in GeoJSON(2008)[3].

Circular Temporal Geometry

Some applications, such as the predication of hurricanes, need to represent a time-varying circular object. The IETF GeoJSON format (https://www.ietf.org/rfc/rfc7946.txt) excludes the circular types such as Circle or Ellipse. No type for "Circle" and "Ellipse" is defined in this version of MF-JSON.

Geometry Object

A moving feature may have a member named `"geometry"` , which may represent its projection in coordinate space as points, curves, or surfaces. The representation of Geometry objects is same as GeoJSON.

7. RESTful API

This Clause provides a design example of a RESTful API as a form of service interface to handle moving features on the Web with ease of development and access, as well as being robust and scalable in server/client distributed environments. In general, Representational State Transfer (REST) focuses on resources and how to access to these resources over Hypertext Transfer Protocol (HTTP). A few use cases of RESTful approaches for OGC web service interfaces are analyzed in the OGC Testbed 11 REST Interface Engineering Report [OGC 15-052r1]. OGC supports the use of RESTful interfaces and JSON encodings and this practice refers to the OGC SensorThings API [OGC 15-078r6] standard as a good example for RESTful principles.

7.1. General Information

Clients and servers exchange representations of resources via HTTP messages as shown in Figure 3. Clients send a request against the resources to the server in the form of a HTTP verb, Uniform Resource Identifier (URI), protocol version, and request content (header and body). The server replies with a response message consisting of the protocol version followed by a status code and content.
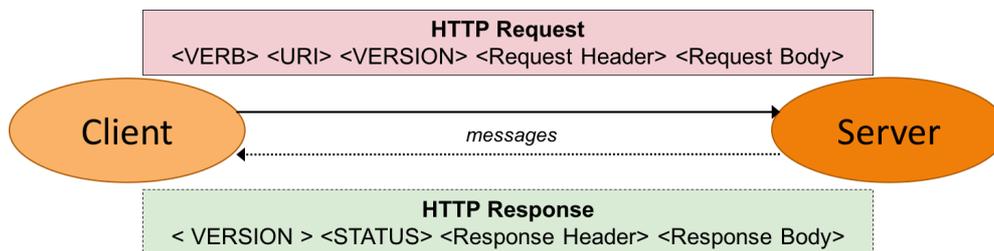
Figure 3. Client/Server Communication based on RESTful API

7.1.1. Verb

Verb is an HTTP method: POST, GET, PUT/PATCH, and DELETE corresponding to create, read, update, and delete (CRUD) operations, respectively. A client make a request with one of the following methods to be applied to the resource.

POST	Creates a new resource.
GET	Retrieves a resource.
PUT	Updates/Replaces an existing resource.
PATCH	Updates/Modifies an existing resource.
DELETE	Deletes a resource.

7.1.2. URI

A Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource as being a locator, a name, or both. The general rules of URIs are defined in [IETF RFC 3986]. The RESTful API to support moving feature data uses a Uniform Resource Locator (URL) for location and access of resources via the HTTP protocol. The basic components of URLs can be defined as follows:

> URL = *SERVICE_ROOT* ["/" *RESOURCE_PATH* ["?" *QUERY_OPTIONS*]]

- *SERVICE_ROOT*: the identification of service endpoints for clients, usually formed as `http:""//"host[":"port]["/"version]`. The "http" scheme is used to locate network resources via the HTTP protocol. The host is a domain name of a network host, or its IPv4 address as a set of four decimal digit groups separated by ".". The version is a API version.

- *RESOURCE_PATH*: the representation of a particular resource. By attaching the resource path after the service root URI, clients can address to different types of resources. (sub-clause 7.2).

- *QUERY_OPTIONS*: clients can apply query options after the resource path to further process the addressed resources, such as sorting by properties or filtering with criteria (sub-clause 7.3).

7.1.3. Version

The service implementation shall support a HTTP version. This document assumes the protocol referred to as "HTTP/1.1" [IETF RFC 2616].

7.1.4. Status

The response message should contain a status code of the attempt to understand and satisfy the request. Depending on the first digit of a 3-digit integer code, it is classified into five roles:

1xx	Informational - Request received, continuing process
2xx	Success - The action was successfully received, understood, and accepted
3xx	Redirection - Further action must be taken in order to complete the request
4xx	Client Error - The request contains bad syntax or cannot be fulfilled
5xx	Server Error - The server failed to fulfill an apparently valid request

Full list of status codes are defined in [IETF RFC 2616], Section 10. This document suggests usage of the following specific codes:

- **200 OK** : General success status code.

- **201 CREATED**: Successful creation of resources for POST or PATCH.

- **204 NO CONTENT**: Successful operation for DELETE or PUT. There is nothing in the response body.

- **400 BAD REQUEST**: General error of client request, such as a wrong parameter, missing data, etc.

- **404 NOT FOUND**: The requested resource is not found.

- **406 Not Acceptable**: The media type given in the Content-Type header field is not `"application/geo+json"` or `"application/geo+json-seq"`.

- **500 Internal Service Error**: Unexpected condition during the request operations in a server side.

- **501 Not Implemented**: The server does not support the functionality required to fulfill the request.

7.1.5. Header

A general HTTP request/response message contains header fields consisting of a name, followed by a colon (":") and the field value. Detailed information on HTTP header fields are described in [IETF RFC 2616], Section 4.5 (General Header Fields), Section 5.3 (Request Header Fields), Section 6.2 (Response Header Fields), and Section 7.1 (Entity Header Fields). The RESTful API of moving features in this document concerns only the Content-Type field, and the other fields are left to the implementation of domain applications.

- **Content-Type**: All request/response message SHALL contain the Content-Type field to indicate the media type of the entity-body. The Content-Type field has one of media types of `"application/geo+json"` and `"application/geo+json-seq"` to interpret Moving Features JSON (MF-JSON) entities in the message body.

- **Transfer-Encoding**: A web server serves content as a stream (dynamically-generated content) with the Transfer-Encoding field set by `"chunked"` . The size of each chuck is sent right before the chuck itself, the representation of content is recommended to use the JSON Text Sequence format [IETF RFC 7464] for the streaming transfer. Content transfer is terminated by a final chunk of length zero.

> The Host request-header field specifies the Internet host and port number of the resource being requested. If the service_root omits the request, the client needs to include this field.

7.1.6. Body

The entity-body (if any) sent with an request or response is in the JSON format. If the entities represent moving feature data, it is encoded by MF-JSON. More examples are given the next sub-clause.

7.2. Resources

A request URI is constructed with a resource path for a corresponding resource item. Resources are the fundamental elements of the RESTful API. This API is designed by the resource classes and their relationships as shown in Figure 4.
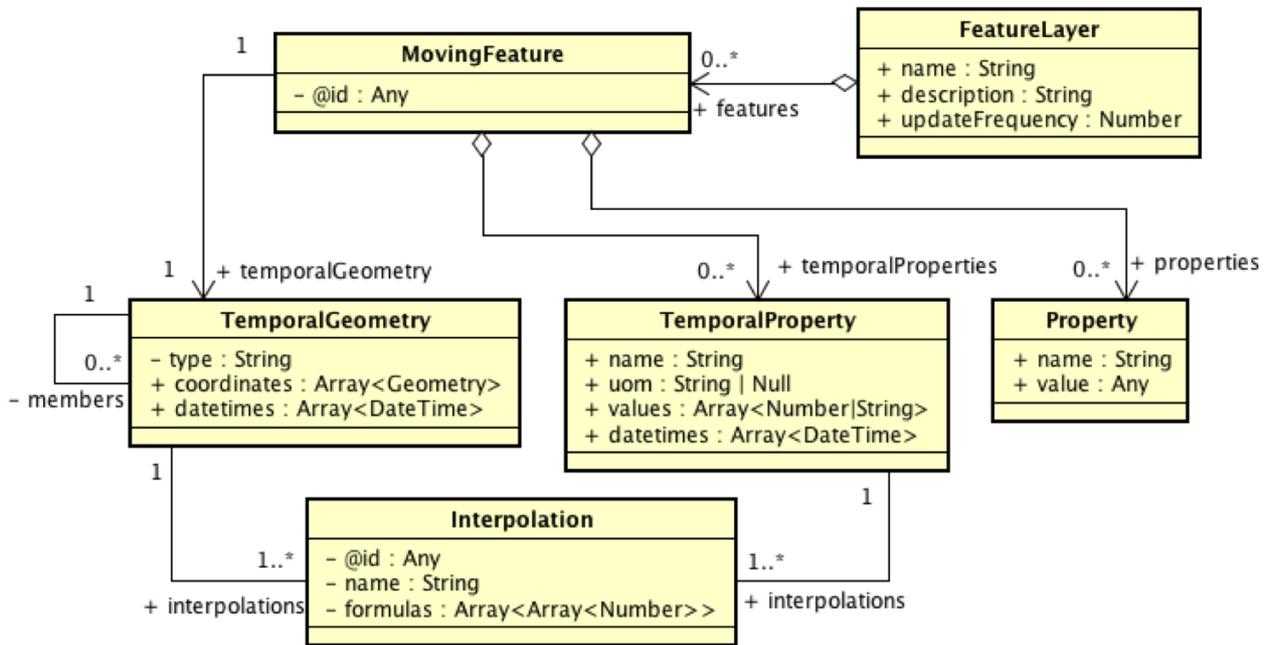
Figure 4. REST Resource Classes of a Moving Feature Service

7.2.1. Resource Classes

- **MovingFeature Class**: Is the root resource to enable the client to access other resources as its properties. A MovingFeature resource is required to be implemented with the following properties in a moving-feature service:

Name	Description	Data type	Multiplicity and use	Editable
@id	It is the unique and system-generated identifier. Clients cannot edit its value.	JSON Any (Null is not allowed)	One (mandatory)	NO

Name	Description	Data type	Multiplicity and use	Editable
temporalGeometry	A navigation property to address the resource containing a temporal geometry.	JSON Object<TemporalGeometry> (Null is not allowed)	One (mandatory)	YES
temporalProperties	A navigation property to address the resource containing temporal properties.	JSON Array<TemporalProperty> or Null	Zero-to-one	YES
properties	A navigation property to address the resource containing static properties as name-value pairs.	JSON Array<Property> or Null	Zero-to-one	YES

JSON Any = False / Null / True / Object / Array / Number / String

A resource instance whose type is specified, such as Geometry, DateTime, TemporalGeometry, TemporalProperty, Property, and so on, is a JSON Object.

- **TemporalGeometry Class**: The class for accessing the temporal geometry of moving features. A TemporalGeometry resource is required to be implemented with the following properties in a moving-feature service:

Name	Description	Data type	Multiplicity and use	Editable
type	Its value is one of "MovingPoint", "MovingLineString", "MovingPolygon", "MultiMovingPoint", "MultiMovingLineString", "MultiMovingPolygon", and "MovingGeometryCollection".	JSON String	One (mandatory)	NO
coordinates	A collection of geometries represented by lists of sampled positions.	JSON Array<Geometry> (or Null for collection types)	Zero-to-one (mandatory for simple types)	YES
datetimes	A collection of datetimes when the coordinates are sampled.	JSON Array<DateTime> (or Null for collection types)	Zero-to-one (mandatory for simple types)	YES
members	A navigation property to address temporal-geometry elements of the collection types such as "MultiMovingPoint", "MultiMovingLineString", "MultiMovingPolygon", and "MovingGeometryCollection".	TemporalGeometry (or Null for simple types)	Zero-to-many	NO
interpolations	A navigation property to address the interpolation methods.	Interpolation (or Null for collection types)	Zero-to-many (mandatory for simple types)	YES

- **TemporalProperty Class**: The class for accessing the temporal properties of moving features. A TemporalProperty resource is optionally to be implemented in a moving-feature service. The minimum properties for the class are defined by:

Name	Description	Data type	Multiplicity and use	Editable
name	A name of dynamic attribute of feature.	JSON String	One (mandatory)	YES (but, no duplication within a moving feature.)
uom	A symbol or URI to address the unit of measurement.	JSON String or Null	Zero-to-one	YES
values	A collection of sampled values to represent dynamic changes of feature attribute.	JSON Array<Number/String>	One (mandatory)	YES
datetimes	A collection of datetimes when the values are sampled.	JSON Array<DateTime>	One (mandatory)	YES
interpolations	A navigation property to address the interpolation methods.	Interpolation	One-to-many (mandatory)	YES

- **Interpolation Class**: The class for accessing an interpolation method. An Interpolation resource is required to be implemented with the following properties in a moving-feature service:

Name	Description	Data type	Multiplicity and use	Editable

Name	Description	Data type	Multiplicity and use	Editable
@id	It is the unique and system-generated identifier.	JSON Any	One (mandatory)	NO
name	A name of interpolation.	JSON String	One (mandatory)	NO
formulas	A collection of formulas to estimate arbitrary values at time.	JSON Any	Zero-to-one	NO

For pre-defined methods such as `"Discrete"`, `"Stepwise"`, `"Linear"`, and `"Spline"`, the client cannot edit the names of methods. This practice assumes that the name of a user-defined interpolation formula is automatically assigned by the service when moving features are inserted.

- **Property Class**: The class for accessing the static properties of moving features. A Property resource is optionally to be implemented in a moving-feature service. The minimum properties for the class are defined by:

Name	Description	Data type	Multiplicity and use	Editable
name	A name of feature attribute.	JSON String	One (mandatory)	YES (but, no duplication within a moving feature.)
value	A value of feature attribute.	JSON Any	One (mandatory)	YES

- **FeatureLayer Class**: The class for a collection of moving features to manage data in a distinct (physical or logical) space; this is an optional resource and can be replaced by an Application-dependent feature. When the service implements this class, the following properties may be required:

Name	Description	Data type	Multiplicity and use	Editable

Name	Description	Data type	Multiplicity and use	Editable
name	A property indicates the label of a FeatureLayer resource	JSON String	One (mandatory)	YES (but, no duplication.)
description	A property describes a short comment about the layer	JSON String	One (mandatory)	YES
updateFrequency	A property provides the connectivity of movement within a time interval (second)	Integer	One (mandatory)	YES (but, the previous data are not affected.)
features	A navigation property to address each moving feature	MovingFeature	Zero-to-many	YES

The moving-feature service needs to manage the connectivity of features' movement by using the updateFrequency interval. If the updateFrequency interval is 0, the server does not manage the connectivity of movements.

7.2.2. Resource Path Patterns

Each resource in the service has at least one URL. The recommended pattern of resource URL paths to address a collection of MovingFeature entities, a MovingFeature entity, and their properties may be formed as:

URL patterns without a FeatureLayer resource

```
- SERVICE_ROOT/MovingFeatures
- SERVICE_ROOT/MovingFeatures(@id)
- SERVICE_ROOT/MovingFeatures(@id)/temporalGeometry
- SERVICE_ROOT/MovingFeatures(@id)/temporalProperties
- SERVICE_ROOT/MovingFeatures(@id)/temporalProperties($NAME)
- SERVICE_ROOT/MovingFeatures(@id)/properties
- SERVICE_ROOT/MovingFeatures(@id)/properties($NAME)
```

> 🛈 @id is the identifier of moving feature; therefore, the server needs to return its identifier when a client inserts a new feature or provides a way to access each identifier.

If the server implements the FeatureLayer resource, it is recommend that the path pattern of resource be formed as:

URL patterns with FeatureLayer resources

```
- SERVICE_ROOT/FeatureLayers/
- SERVICE_ROOT/FeatureLayers($NAME)/features
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)/temporalGeometry
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)/temporalProperties
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)/temporalProperties($NAME)
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)/properties
- SERVICE_ROOT/FeatureLayers($NAME)/features(@id)/properties($NAME)
```

> 🛈 $NAME is replaced by a value of property `name`.

> ⚠️ For the root type of resources, a server should take one resource type of MovingFeatures or FeatureLayers. Both a MovingFeatures and FeatureLayers resource cannot exist at the same level.

7.2.3. Resource Path Examples

A client can make a request against the resources via URL using various HTTP methods: POST, GET, PUT/PATCH, and DELETE.

Example 7.1: To insert a moving feature data into the service.

```
>>> Request
POST  SERVICE_ROOT/MovingFeatures    HTTP1.1
Content-Type: application/geo+json
{
    "type": "MovingFeature",
    "temporalGeometry": {
      "type": "MovingPoint", // a geometry type to represent a trajectory object
      "coordinates": [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0] ],
      "datetimes": ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z", "2011-07-15T01:01:01Z"],
      "interpolations": ["Linear"]
    },
    "temporalProperties": [   //(optional)
      {
        "datetimes" : ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z"],
        "length": {
          "uom": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length", // a URL denoting
a unit-of-measure
          "values": [1.0, 2.4, 1.0],
          "interpolations": ["Stepwise"]
        }
      }
    ]
}


<<< Response
201 CREATED
{
    "@id": "mf0001",
    "@created": "2012-07-14T22:01:01" // Application-defined property
}
```

*Example 7.2: To insert a list of moving feature data into the service with the
"application/geo+json-seq" content type.*

```
>>> Request
POST  SERVICE_ROOT/MovingFeatures    HTTP1.1
Content-Type: application/geo+json-seq
{ "type": "MovingFeature", "temporalGeometry": {...}, "temporalProperties": [...]}
{ "type": "MovingFeature", "temporalGeometry": {...}, "temporalProperties": [...]}
{ "type": "MovingFeature", "temporalGeometry": {...}, "temporalProperties": [...]}

<<< Response
201 CREATED
{
    "@id": ["mf0001", "mf0002", "mf0003"]
    "@created": "2012-07-14T22:01:01" // Application-defined property
}
```

Example 7.3: To add new trajectory data into a stored moving feature.

```
>>> Request
POST  SERVICE_ROOT/MovingFeatures('mf0001')/temporalGeometry   HTTP1.1
Content-Type: application/geo+json
{
    "type": "MovingPoint", // a geometry type to represent a trajectory object
    "coordinates": [ [100.0, 0.0], [101.0, 0.0]],
    "datetimes": ["2011-07-16T02:01:01Z", "2011-07-16T06:01:01Z"],
    "interpolation": ["Linear"]
}

<<< Response
201 CREATED
{
    "@modified": "2012-07-14T22:01:01" // Application-defined property
}
```

Example 7.4: To get the list of all the temporal property information.

```
>>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalProperties   HTTP1.1

<<< Response
200 OK
[
  {
    "datetimes" : ["2011-07-14T22:01:01Z", "2011-07-14T23:01:01Z", "2011-07-
15T00:01:01Z"],
    "length": {
      "uom": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length",
      "values": [1.0, 2.4, 1.0],
      "interpolations": ["Stepwise"]
    }
  }
]
```

Example 7.5: To retrieve a temporal property information whose name is 'dose'.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalProperties('dose')   HTTP1.1

<<< Response
200 OK
{
    "datetimes" : ["2011-07-14T22:01:01Z", "2011-07-15T12:01:01Z"],
    "dose": {
      "uom": "http://www.qudt.org/qudt/owl/1.0.0/quantity/AbsorbedDose",
      "values": [0.003, 0.003],
      "interpolations": ["Discrete"]
    }
}
```

Example 7.6: To modify the name of a property.

```
>>> Request
PUT  SERVICE_ROOT/MovingFeatures('mf0001')/properties#name='title'   HTTP1.1

<<< Response
204 NO CONTENT
```

7.3. Access Interfaces

The OGC Moving Features Access [OGC 16-120r3] document requires three types of operations for accessing moving feature data: Type A, Type B, and Type C as follows:

- Type A: Retrieval of feature attribute
  For example, these operations retrieve positions, trajectories, and velocities of a moving feature such as a car, a person, a vessel, an aircraft, and a hurricane.

- Type B: Operations between one trajectory object and one or more geometry objects
  An example of these operations is "intersection" between a geometry object like an administrative boundary and a trajectory of a moving feature like a car, a person, a vessel, an aircraft, and a hurricane.

- Type C: Operations between two trajectory objects
  An example of these operations is to calculate a distance of the nearest approach of a trajectory to another trajectory. The case studies are distance between a criminal agent and a police agent for predicting crime patterns or distance between soccer players for making proper tactics.

OGC Moving Feature Access provides a guideline for implementing interfaces to support moving feature data calculations in a database, data service, or an application using various programming languages or protocols (e.g., SQL functions, Java APIs, and Web APIs). This sub-clause exemplifies how to realize those operations using RESTful API based on a resource URL, followed by the query option parts. The client is able to retrieve a resource representation using a HTTP GET request.

> **"GET"** *SERVICE_ROOT* **"/"** *RESOURCE_PATH* **"?"** *QUERY_OPTIONS* **"HTTP/1.1"**

This document basically follows the usage of query options ($filter, $count, $orderby, $skip, $top, $select, and $search) as used in the OGC SensorThings API standard [OGC 15-078r6]. The OGC SensorThings API provides a good example of adaptation in OData protocols [OData-Part1] and extension of geospatial query functions. This practice employs the `$select`, `$filter`, and `$search` options in moving feature operations. In order to make a simple rule for the query options, the abstract operations of moving features as identified in [OGC 16-120r3] are re-

categorized into three interfaces as shown in Figure 5: **Harvest**, **Relation**, and **Analysis**.
Interface **GeoSpatial** and **Temporal** are additionally defined to access derived properties from a
TemporalGeometry and TemporalProperty resource.

> The Figure 5 omits the parameters and return types of Type A, B, and C
> operations. The detailed specification of each operation in Type A, B, and C (gray
> color boxes) is described by the OGC Moving Features Access standard. The other
> interfaces (yellow color boxes) are not related to the OGC Moving Features Access
> standard.
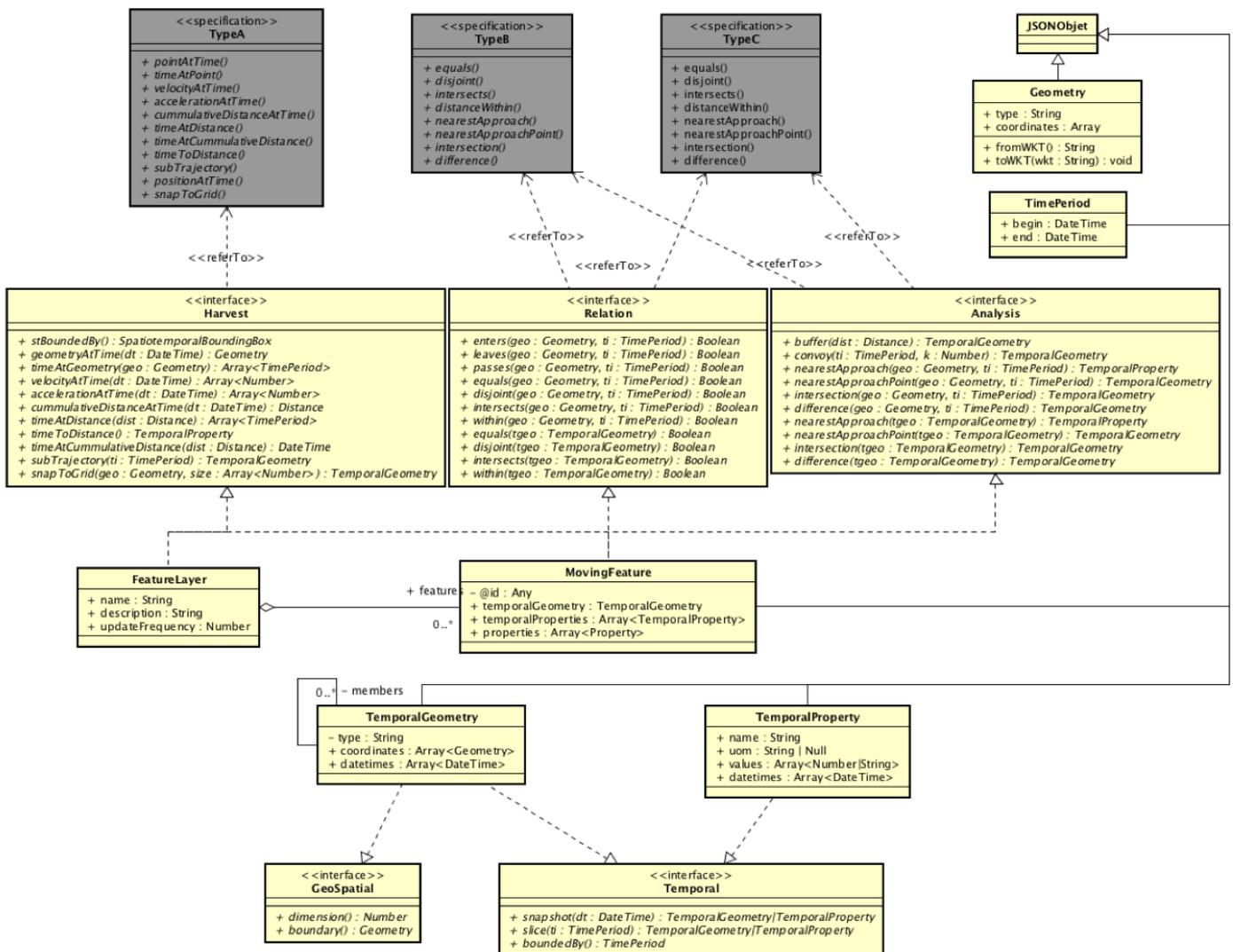
Figure 5. Interface Types of RESTful API

- **Harvest Interface**: This interface mostly implements the Type A operations to retrieve
  feature attributes. The practice uses `geometryAtTime` and `timeAtGeometry` to extend the
  geometry types not only 0-dimensional geometry objects but also 1- and 2-dimensional
  geometry objects instead of *pointAtTime* and *timeAtPoint* in the Moving Features Access
  document. In addition, operation `stBoundedBy` is added to return the boundary object
  containing moving features in a spatiotemporal domain. This interface is realized with Query
  Option `$select`.

- **Relation Interface**: This interface implements the topological relationship operations in Type B (between trajectory and geometry objects) and Type C (between two trajectory objects), such as *disjoint* and *intersects*. The practice changes the name of *distanceWithin* to `within` to exclude the distance parameter. Moreover, new relations of `enters`, `leaves`, and `passes` are defined to clarify its temporal order of topological changes of relation from *intersects*. Figure 6 shows the examples of each relation. This interface is realized with Query Option `$filter`.
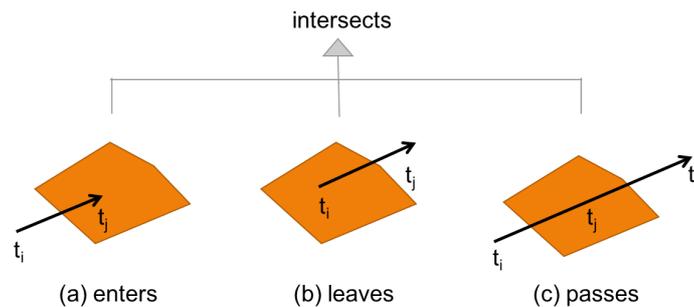

Figure 6. Specialized Relations of Moving Features from intersects ($t_i < t_j < t_k$ for time instant t_i, t_j, t_k)

- **Analysis Interface**: This interface implements the analysis operations in Type B (between trajectory and geometry objects) and Type C (between two trajectory objects), such as *nearestApproach* and *difference*. The operations return a new object as a computation result. For example, operation "nearestApproach" with geometric object calculates the distance between a trajectory and the geometry or between two trajectories, and returns a distance in time. This practice additionally defines two more operations: `buffer` and `convoy`. The `buffer` operation returns a temporal geometry (simple or collection) covering all points within a given distance from a target temporal geometry (simple or collection). The `convoy` operation returns the minimum bounding temporal geometry enclosing a group of moving features such that these features are consecutively close to each other (the k-nearest neighbors from a target) during a given time period. Figure 7 shows the result examples of `buffer` and `convoy` operations. This interface is realized with Query Option `$search`.
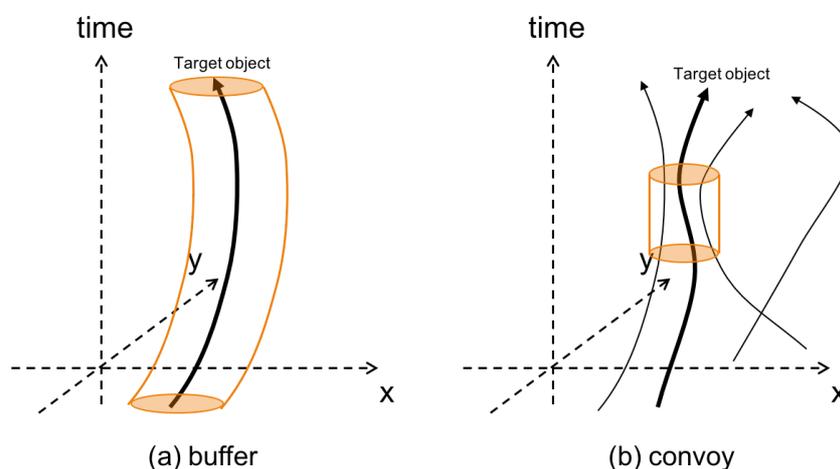

Figure 7. Examples of Analysis Operations

- **GeoSpatial Interface**: This interface provides the function `dimension` that returns the maximum dimension of leaf geometry objects and `boundary` that returns the closure of the combinatorial geospatial boundary of all leaves (a foliation) for a temporal geometry resource. This interface is realized with Query Option `$select`.

- **Temporal Interface**: This interface provides the functions `snapshot` and `slice` that returns a sub-object of a temporal geometry/property at a given time instant and time period, respectively. The `boundedBy` function returns the temporal range covering its prism. This interface is realized with Query Option `$select`.

7.3.1. Query Option `$select`

The `$select` query option requests the service to harvest the properties or return a value derived from properties. The operations of **Harvest**, **GeoSpatial**, and **Temporal** are realized with the `$select` query option corresponding to a MovingFeature or FeatureLayer resource.

Example 7.7: geometryAtTime of a moving feature with ID.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?$select=geometryAtTime(2008-02-04T00:00:00Z)
HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "geometryAtTime" : {
        "type" : "POINT",
        "coordinates" : [ 116.35072, 39.96354 ]
    }
}
```

Example 7.8: geometryAtTime of all moving features.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?$select=geometryAtTime(2008-02-04T00:00:00Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id" : "mf0001",
        "geometryAtTime" : {
            "type" : "POINT",
            "coordinates" : [ 116.35072, 39.96354 ]
        }
    },
    {
        "@id" : "mf0002",
        "geometryAtTime" : {
            "type" : "POINT",
            "coordinates" : [ 116.01843751281389, 39.909385232047136 ]
        }
    }
]
```

Example 7.9: timeAtGeometry of a moving feature with ID.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?$select=timeAtGeometry(POINT(116.4%2039.8))
HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "timeAtGeometry" : [ "2008-02-07T13:00:37Z", "2008-02-07T13:00:51Z", "2008-02-
08T13:35:57Z", "2008-02-08T13:36:06Z", "2008-02-06T13:08:32Z", "2008-02-06T13:08:29Z",
"2008-02-07T23:04:51Z", "2008-02-07T23:04:55Z", "2008-02-06T23:09:41Z", "2008-02-
06T23:09:56Z", "2008-02-04T00:36:14Z", "2008-02-04T00:36:23Z", "2008-02-05T13:17:28Z",
"2008-02-05T13:17:39Z" ]
}
```

Example 7.10: timeAtGeometry of all moving features.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?$select=timeAtGeometry(POINT(116.4%2039.8))  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id" : "mf0001",
        "timeAtGeometry" : [ "2008-02-07T13:00:37Z", "2008-02-07T13:00:51Z", "2008-02-
08T13:35:57Z", "2008-02-08T13:36:06Z", "2008-02-06T13:08:32Z", "2008-02-06T13:08:29Z",
"2008-02-07T23:04:51Z", "2008-02-07T23:04:55Z", "2008-02-06T23:09:41Z", "2008-02-
06T23:09:56Z", "2008-02-04T00:36:14Z", "2008-02-04T00:36:23Z", "2008-02-05T13:17:28Z",
"2008-02-05T13:17:39Z" ]
    },
    {
        "@id": "mf0002",
        "timeAtGeometry": ["2013-05-01T10:33:45Z"]
    }
]
```

Example 7.11: velocity of the all moving feature.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?$select=velocity(2008-02-04T00:00:00Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id" : "mf0001",
        "velocity" : [ 0.0, 0.0 ]
    },
    {
        "@id" : "mf0002",
        "velocity" : [ -5.42125853725347E-4, 1.0906301588750913E-4 ]
    }
]
```

Example 7.12: timeAtCummulativeDistance of a moving feature with ID.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?$select=timeAtCummulativeDistance(1,%22km%22)
HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "timeAtCummulativeDistance" : "2008-02-02T22:35:04Z"
}
```

Example 7.13: stBoundedBy of moving features.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?$select=stBoundedBy()  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
  {
    "@id" : "mf0001",
    "stBoundedBy" : {
      "bbox": [-10.0, -10.0, 10.0, 10.0],
      "period": {
        "begin": "1994-11-05T13:15:30Z",
        "end" : "1994-11-05T13:15:30Z"
      }
    }
  },
  {
    "@id" : "mf0002",
    "stBoundedBy" : {
      "bbox": ...,
      "period": ...
    }
  }
]
```

Example 7.14: subTrajectory of a moving feature with ID.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?$select=subTrajectory(2008-02-
03T23:00:00Z,2008-02-03T23:05:00Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "subTrajectory" : {
        "type" : "MovingPoint",
        "datetimes" : [ "2008-02-03T23:00:00Z", "2008-02-03T23:04:34Z", "2008-02-
03T23:03:49Z", "2008-02-03T23:03:04Z", "2008-02-03T23:02:19Z", "2008-02-03T23:01:29Z",
"2008-02-03T23:00:44Z", "2008-02-03T23:00:04Z", "2008-02-03T23:04:39Z", "2008-02-
03T23:03:54Z", "2008-02-03T23:03:09Z", "2008-02-03T23:02:24Z", "2008-02-03T23:01:34Z",
"2008-02-03T23:00:49Z", "2008-02-03T23:00:09Z", "2008-02-03T23:04:44Z", "2008-02-
03T23:03:59Z", "2008-02-03T23:03:14Z", "2008-02-03T23:02:29Z", "2008-02-03T23:01:39Z",
"2008-02-03T23:00:54Z", "2008-02-03T23:00:14Z", "2008-02-03T23:04:49Z", "2008-02-
03T23:04:04Z", "2008-02-03T23:03:19Z", "2008-02-03T23:02:34Z", "2008-02-03T23:01:49Z",
"2008-02-03T23:00:59Z", "2008-02-03T23:00:19Z", "2008-02-03T23:04:56Z", "2008-02-
03T23:04:09Z", "2008-02-03T23:03:24Z", "2008-02-03T23:02:39Z", "2008-02-03T23:01:54Z",
"2008-02-03T23:01:04Z", "2008-02-03T23:00:24Z", "2008-02-03T23:04:14Z", "2008-02-
03T23:03:29Z", "2008-02-03T23:02:49Z", "2008-02-03T23:01:59Z", "2008-02-03T23:01:09Z",
"2008-02-03T23:00:29Z", "2008-02-03T23:04:19Z", "2008-02-03T23:03:34Z", "2008-02-
03T23:02:54Z", "2008-02-03T23:02:04Z", "2008-02-03T23:01:19Z", "2008-02-03T23:00:34Z",
"2008-02-03T23:04:29Z", "2008-02-03T23:03:44Z", "2008-02-03T23:02:59Z", "2008-02-
03T23:02:09Z", "2008-02-03T23:01:24Z", "2008-02-03T23:00:39Z", "2008-02-03T23:05:00Z" ],
        "coordinates" : [ [ 116.35079, 39.96372 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96422 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96372 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96422 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96421 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35107, 39.96425 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35106, 39.96425 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35105, 39.96424 ], [ 116.35079, 39.96373 ], [
116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35104, 39.96424 ], [ 116.35079, 39.96374 ], [ 116.35079, 39.96373 ], [
116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35104,
39.96423 ], [ 116.35079, 39.96374 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ] ],
        "interpolations" : ["Linear"]
    }
}
```

The subTrajectory operation can also be realized by using the slice interface of temporal
geometry as follows:

Example 7.15: slice of the temporal geometry of the moving feature whose identifier is 'mf0001'.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalGeometry?$select=slice(2008-02-
03T23:00:00Z,2008-02-03T23:05:00Z)   HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "slice": {
        "type" : "MovingPoint",
        "datetimes" : [ "2008-02-03T23:00:00Z", "2008-02-03T23:04:34Z", "2008-02-
03T23:03:49Z", "2008-02-03T23:03:04Z", "2008-02-03T23:02:19Z", "2008-02-03T23:01:29Z",
"2008-02-03T23:00:44Z", "2008-02-03T23:00:04Z", "2008-02-03T23:04:39Z", "2008-02-
03T23:03:54Z", "2008-02-03T23:03:09Z", "2008-02-03T23:02:24Z", "2008-02-03T23:01:34Z",
"2008-02-03T23:00:49Z", "2008-02-03T23:00:09Z", "2008-02-03T23:04:44Z", "2008-02-
03T23:03:59Z", "2008-02-03T23:03:14Z", "2008-02-03T23:02:29Z", "2008-02-03T23:01:39Z",
"2008-02-03T23:00:54Z", "2008-02-03T23:00:14Z", "2008-02-03T23:04:49Z", "2008-02-
03T23:04:04Z", "2008-02-03T23:03:19Z", "2008-02-03T23:02:34Z", "2008-02-03T23:01:49Z",
"2008-02-03T23:00:59Z", "2008-02-03T23:00:19Z", "2008-02-03T23:04:56Z", "2008-02-
03T23:04:09Z", "2008-02-03T23:03:24Z", "2008-02-03T23:02:39Z", "2008-02-03T23:01:54Z",
"2008-02-03T23:01:04Z", "2008-02-03T23:00:24Z", "2008-02-03T23:04:14Z", "2008-02-
03T23:03:29Z", "2008-02-03T23:02:49Z", "2008-02-03T23:01:59Z", "2008-02-03T23:01:09Z",
"2008-02-03T23:00:29Z", "2008-02-03T23:04:19Z", "2008-02-03T23:03:34Z", "2008-02-
03T23:02:54Z", "2008-02-03T23:02:04Z", "2008-02-03T23:01:19Z", "2008-02-03T23:00:34Z",
"2008-02-03T23:04:29Z", "2008-02-03T23:03:44Z", "2008-02-03T23:02:59Z", "2008-02-
03T23:02:09Z", "2008-02-03T23:01:24Z", "2008-02-03T23:00:39Z", "2008-02-03T23:05:00Z" ],
        "coordinates" : [ [ 116.35079, 39.96372 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96422 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96372 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96422 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35102, 39.96421 ], [ 116.35079, 39.96374 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35107, 39.96425 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35106, 39.96425 ], [
116.35079, 39.96373 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35076, 39.96383 ], [ 116.35105, 39.96424 ], [ 116.35079, 39.96373 ], [
116.35079, 39.96373 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076,
39.96383 ], [ 116.35104, 39.96424 ], [ 116.35079, 39.96374 ], [ 116.35079, 39.96373 ], [
116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35076, 39.96383 ], [ 116.35104,
39.96423 ], [ 116.35079, 39.96374 ], [ 116.35079, 39.96373 ], [ 116.35076, 39.96383 ] ],
        "interpolations" : ["Linear"]
    }
}
```

However, the slice operation is for only one MovingFeature resource. This practice restricts the
use of the GeoSpatial and Temporal interfaces to only TemporalGeometry or/and
TemporalProperty resource types. The following examples show the use cases of each operation.

*Example 7.16: snapshot of the temporal property whose name is 'length' of the moving feature
whose identifier is 'mf0001'.*

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalProperties('length')?
$select=snapshot(2013-05-01T10:33:41Z)   HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "snapshot": {
      "datetimes": ["2013-05-01T10:33:41Z"],
      "length": {
        "uom": "http://www.qudt.org/qudt/owl/1.0.0/quantity/Length",
        "values": [1.2],
        "interpolations": ["Discrete"]
    }
  }
}
```

Example 7.17: boundedBy of the temporal geometry of the moving feature whose identifier is 'mf0001'.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalGeometry?$select=boundedBy()   HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "boundedBy": {
        "begin" : "2008-02-03T23:00:00Z",
        "end" : "2008-02-03T23:05:00Z"
    }
}
```

Example 7.18: boundary of the temporal geometry of the moving feature whose identifier is 'mf0001'.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalGeometry?$select=boundary()   HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
  "boundary": {
    "type": "LineString",
    "coordinates": [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0] ]
  }
}
```

Example 7.19: to select a part of a temporal property of the moving feature whose identifier is 'mf0001'.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')/temporalProperties('length')?select=values gt
10 HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
  "temporalProperties": [{
      "datetimes": ["2017-03-13T01:00:00Z","2017-03-13T21:00:00Z", "2017-03-
13T23:00:00Z"],
      "length": {
        "uom" : "m",
        "values": [10, 15, 20],
        "interpolations": "Linear"
  }]
}
```

7.3.2. Query Option `$filter`

The `$filter` query option allows clients to filter a collection of moving features that are
addressed by a request URL. It can be applied to a specific moving feature or a collection of
moving features. The `$filter` option is evaluated for each moving feature in the collection, and
returns the features where the expression of the Relation interface is `true` in the response. For a
specific feature, it returns `true` or `false`. The **Relation** interface is realized with the `$filter`
query option.

Example 7.20: The disjoint operation returns true or false corresponding to a moving feature.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?
$filter=disjoint(LINESTRING(1%202,3%204,5%206),2008-02-02T22:31:00Z,2008-02-02T22:40:00Z)
HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "disjoint": true
}
```

*Example 7.21: The intersects operation returns the identifiers of moving features whose trajectory
intersects with the parameter geometry object for a particular period of time in the collection.*

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?
$filter=intersects(POLYGON((30%2010%2C40%2040%2C20%2040%2C10%2020%2C30%2010)),2013-05-
01T10:33:50Z,2013-05-01T10:36:41Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "intersects": [ "mf0001", "mf0003"]
}
```

Example 7.22: The distanceWithin operation returns the identifiers of moving features which are located within 100km from the given position during the parameter time period.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?$filter=distanceWithin(POINT(103%201.0),2013-05-
01T10:33:50Z,2013-05-01T10:36:41Z,100;km) HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "distanceWithin ": null
}
```

> The operations of Type C between two trajectory objects provide spatiotemporal relations (e.g., intersects). The current API tries to realize the Type C operation by a temporary method until an expression is developed for the request URL, such as the Well-Known Text (WKT) format for a geometry object.
>
> *(1) Create a temporal resource of FeatureLayer as follows:*
>
> ```
> >>> Request
> POST  SERVICE_ROOT/FeatureLayers   HTTP1.1
> Content-Type: application/geo+json
> {
>     "name": "queryfeatures",
>     "description": "temporal feature layer for query parameters"
>     "updateFrequency": 0
> }
>
> <<< Response
> 201 CREATED
> {
>     "@id": "queryfeatures",
>     "@created": "2012-07-14T22:01:01"
> }
> ```
>
> *(2) Insert a query object into the feature layer as follows:*

```
>>> Request
POST  SERVICE_ROOT/FeatureLayers('queryfeatures')  HTTP1.1
Content-Type: application/geo+json
{
    "type": "MovingFeature",
    "temporalGeometry": {
    ....
    }
}
<<< Response
201 CREATED
{
    "@id": "mf9999999",
    "@created": "2012-07-14T23:01:01"
}
```

(3) Search intersected moving features from the bus layer with the query feature.

```
>>> Request
GET  SERVICE_ROOT/FeatureLayers('bus')/features?
$filter=intersects(@id)&&@id='mf9999999'  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "intersects": [ "mf0001", "mf0003"]
}
```

7.3.3. Query Option `$search`

The `$search` query option allows clients to analyze moving features or a specific moving feature, and returns a new object as a computation result. The **Analysis** interface is realized with the `$search` query option.

Example 7.23: To search the nearest approach point with ID.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('mf0001')?
$search=nearestApproachPoint(LINESTRING(116.35%2039.8,116.36%2040),2008-02-
04T00:00:00Z,2008-02-04T00:10:00Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
      "nearestApproachPoint" : {
          "type" : "MovingPoint",
          "datetimes" : [ "2008-02-04T00:06:14Z", "2008-02-04T00:06:19Z" ],
          "coordinates" : [ [ 116.35089, 39.96329 ], [ 116.35814635910225,
39.962927182044886 ] ],
          "interpolations" : ["Linear"]
      }
}
```

Example 7.24: To search the nearest approach distance for all moving features.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?
$search=nearestApproach(LINESTRING(116.35%2039.8,116.36%2040),2008-02-04T00:00:00Z,2008-
02-04T00:10:00Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id" : "mf0001",
        "nearestApproach" : {
          "datetimes" : [ "2008-02-04T00:06:14Z", "2008-02-04T00:06:19Z" ],
          "distance" : {
            "uom" : "m",
            "values" : [ 7.265423889165578, 7.265423889165578 ],
            "interpolations" : ["Linear"]
          }
        }
    }
  ]
```

Example 7.25: To compute each intersection object of the temporal geometry from a collection of moving features with a parameter geometry object for a particular period of time.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures?
$search=intersection(POLYGON((30%2010%2C40%2040%2C20%2040%2C10%2020%2C30%2010)),2013-05-
01T10:33:50Z,2013-05-01T10:36:41Z)  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id": "mf001",
        "intersection": {
            "type": "MovingPoint,
            "coordinates": [...],
            "datetimes" : [...],
            "interpolations": [...]
        }
    },
    {
        "@id": "mf003",
        "intersection": {
            ...
        }
    }
]
```

The `$search` query option can be used for the full text search of properties/temporalProperties of moving features with a set of keywords. For example,

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures/properties?$search=bike  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id": "mf001",
        "properties": {
            "name": "bike",
            "description": "fast bike"
        }
    }
]
```

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures/properties('name')?$search=bike  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id": "mf001",
        "name": "bike"
    }
]



>>> Request
GET  SERVICE_ROOT/MovingFeatures/temporalProperties?$search=(mountain OR
bike) AND NOT clothing  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id": "mf003",
        "temporalProperties": [{
            "datetimes": ["2017-03-13T01:00:00Z","2017-03-13T21:00:00Z",
"2017-03-13T23:00:00Z"],
            "messages": {
              "uom" : "text",
              "values": ["mountain bike", "mountain", "bike"],
              "interpolations": "Discrete"
            },
            "annotations": {
              "uom" : "text",
              "values": ["a mountain bike", "a mountain", "a bike"],
              "interpolations": "Discrete"
            }
        }]
    }
]
```

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures/temporalProperties('messages')?$search=
(mountain OR bike) AND NOT clothing  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
[
    {
        "@id": "mf003",
        "messages": {
            "datetimes": ["2017-03-13T01:00:00Z","2017-03-13T21:00:00Z",
"2017-03-13T23:00:00Z"],
            "values": ["mountain bike", "mountain", "bike"]
        }
    }
]
```

7.3.4. Addressing Entities: `$ref`, `$value`

This practice uses the symbolic resource `$ref`, located at the service root or a FeatureLayer entity, to get the URL for addressing all entities. The symbolic resource `$value` is allowed to resolve to a single property value or to get a list of values of TemporalProperty and Property elements of a MovingFeatures resource in text/plain form.

Example 7.26: To get the URLs of all MovingFeatures entities if the service does not have any resource of FeatureLayer.

```
>>> Request
GET  SERVICE_ROOT/$ref  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "url": ["MovingFeatures('m0001')", "MovingFeatures('m0002')"]
}
```

Example 7.27: To get the URLs of all FeatureLayer entities if the service manages resources of FeatureLayer.

```
>>> Request
GET  SERVICE_ROOT/$ref  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "url": ["FeatureLayers('bus')", "FeatureLayers('typhoon')"]
}
```

Example 7.28: To get the URLs of all MovingFeatures entities in a FeatureLayer entity.

```
>>> Request
GET  SERVICE_ROOT/FeatureLayers('typhoon')/$ref  HTTP1.1

<<< Response
200 OK
Content_Type: application/geo+json
{
    "url": ["features('m0001')", "features('m0002')"]
}
```

Example 7.29: To get a list of names of FeatureLayer.

```
>>> Request
GET  SERVICE_ROOT/FeatureLayers/name/$value  HTTP1.1

<<< Response
200 OK
Content_Type: text/plain
bus, typhoon
```

Example 7.30: To get a list of static properties' name.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('m0001')/properties/name/$value  HTTP1.1

<<< Response
200 OK
Content_Type: text/plain
title, author
```

Example 7.31: To get a value of a static properties.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('m0001')/properties('title')/$value  HTTP1.1

<<< Response
200 OK
Content_Type: text/plain
Test Title
```

Example 7.32: To get a list of temporal properties' name.

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('m0001')/temporalProperties/name/$value  HTTP1.1

<<< Response
200 OK
Content_Type: text/plain
length, dose, picture
```

Example 7.33: To get a unit value of a temporal property

```
>>> Request
GET  SERVICE_ROOT/MovingFeatures('m0001')/temporalProperties('length')/uom/$value
HTTP1.1

<<< Response
200 OK
Content_Type: text/plain
http://www.qudt.org/qudt/owl/1.0.0/quantity/Length
```

Bibliography

- [1] Schadow, G. and McDonald, C. J.: Unified Code for Units of Measure (UCUM), unitsofmeasure.org

- [2] EPSG: EPSG Geodetic Parameter Dataset, www.epsg.org

- [3] IETF: [RFC 7946] The GeoJSON Format, 2016

Appendix A: Revision History

Date	Release	Author	Paragraph modified	Description
2016-08-29		Kyoung-Sook KIM, Hirotaka OGAWA	All	Created
2016-09-27		Kyoung-Sook KIM	RESTful APIs: Resource types and interfaces	Revised
2016-10-23		Kyoung-Sook KIM	Interpolation formulas, RESTful APIs	Revised
2017-01-15		Kyoung-Sook KIM	RESTful APIs- Addressing Entities	Added
2017-03-15		Kyoung-Sook KIM	Temporal Properties	Revised
2017-06-12		Scott Simmons	All	prepare for publication

Last updated 2017-06-28 10:21:35 EDT