OGC® DOCUMENT: 22-038R2

External identifier of this OGC® document: http://www.opengis.net/doc/PER/T18-D025

Open Geospatial Consortium

TESTBED-18: REFERENCE FRAME TRANSFORMATION ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2023-01-13
Approval Date: 2023-03-02
Publication Date: 2023-03-09
Editor: Martin Desruisseaux

Note

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

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

CONTENTS

LIST OF TABLES

LIST OF FIGURES

I EXECUTIVE SUMMARY

This Engineering Report assesses the applicability of existing OGC/ISO standards in order to describe objects in orbit around any celestial body or in free flight in our solar system. This report follows up the theoretical groundwork laid by the 3D+ Standards Framework Engineering Report (OGC 22-036). In this report, the ISO 19111 — Referencing by Coordinates Standard and the draft GeoPose can be used for describing a relationship between, for example, a spacecraft and a ground station.

Most concepts defined in GeoPose can also be expressed using existing ISO 19111 constructs and can be shared as an OGC Geography Markup Language (GML) encoding. The GML encoding of coordinate operations is currently more well suited than Well Known Text (WKT) for GeoPose use cases because GML has three features intentionally excluded from WKT (for simplicity reasons) namely: parameter groups, pass-through operations, and x-links. For coordinate *reference systems* (as opposed to coordinate *operations*), WKT is well suited.

A JSON encoding was not considered in this TestBed 18 activity and is therefore not in this report. However, the current PROJ JSON proposal (not yet standardized) closely follows the ISO 19111 model. With JSON and GML both following the same model, the expectation is that most discussions about GML in this ER will apply also to JSON encoding, with few differences other than syntax, such as use of braces instead of `<element>`.

As a proof of concept, the following scenario is defined. A spacecraft (e.g., Voyager 2) has its own Coordinate Reference System (CRS) attached to it. Some astronomical body has been detected by the spacecraft. The location of that astronomical body is expressed in coordinates relative to the spacecraft. The following process chain needs to be expressed:

- perform coordinate transformations from spacecraft CRS to heliocentric CRS;
- then to Earth-Centered Inertial CRS;
- then to classical geographic CRS; and
- finally to the CRS of the platform where there is an observatory in order to obtain the coordinates of the sky region to observe.

This chain of operations should be built using only existing OGC/ISO standards with as few extensions as possible. The result is the `VoyagerToObservatory.xml` file encoded in the Geographic Markup Language (GML) format and published in the Testbed (TB) GitHub repository.

II KEYWORDS

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

testbed, referencing, geopose

III SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV SUBMITTING ORGANIZATIONS

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

- Geomatys

V ABSTRACT

Currently, most OGC standards focus on data that is observed on the ground or directly above planet Earth. Other standards, such as GeoSciML, provide a data model and transfer standard for geological data. Other projects have considered data models and exchange standards for the seas and oceans. Extra-terrestrial space and the exact location of remote spaceborne sensors has been less in focus. This OGC Testbed 18 Engineering Report (ER) starts with an evaluation of current standards and then proposes changes or extensions to those standards in order to describe objects in orbit around any celestial body or in free flight in our solar system with respect to their location, trajectory, and orientation. Finally standard-based mechanisms to transform a location within a reference frame to a location within another reference frame are examined.

1 SCOPE

1 SCOPE

This Engineering Report provides information about the use of Geography Markup Language (GML) and Well-Known Text (WKT) formats with spacecrafts, celestial bodies, and GeoPose frames. This ER is applicable to existing standards and can be used by Standard Working Groups (SWG) as a source of ideas for future evolutions of those standards.

This Engineering Report does not cover International Celestial Reference System (ICRS), its realizations (ICRF), and their relationship to ITRS/ITRF, which is defined by the International Earth Rotation Service (IERS). Those topics are covered in the *3D+ Standards Framework Engineering Report* (OGC 22-036).

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

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

ISO: **Geographic information — Reference model — Part 1: Fundamentals**, 2014. https://www.iso.org/standard/59164.html

ISO: **Geographic information — Conceptual schema language**, 2015. https://www.iso.org/standard/56734.html

ISO: **Geographic information — Spatial referencing by coordinates**, 2019. https://www.iso.org/standard/74039.html

ISO: **Geographic information — Spatial referencing by geographic identifiers**, 2019. https://www.iso.org/standard/70742.html

ISO: **Geographic information — Well-known text representation of coordinate reference systems**, 2019. https://www.iso.org/standard/76496.html

ISO: **Geographic information — Geography Markup Language (GML)**, 2007. https://www.iso.org/standard/32554.html

Adrian Custer: OGC 09-083r4, *GeoAPI 3.0 Implementation Standard with corrigendum*. Open Geospatial Consortium (2018). https://portal.ogc.org/files/?artifact id=71648.

OGC: **GeoPose Specification draft**, 2021. https://github.com/opengeospatial/GeoPose/

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 TERMS, DEFINITIONS AND ABBREVIATED TERMS

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

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

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

3.1. conceptual model

model that defines concepts of a universe of discourse

[**SOURCE:** ISO 19101-1]

3.2. constraint

UML condition or restriction expressed in natural language text or in a machine readable language for the purpose of declaring some of the semantics of an element

[**SOURCE:** ISO 19103]

3.3. coordinate

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

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

[**SOURCE:** ISO 19111]

3.4. coordinate operation

process using a mathematical model, based on a one-to-one relationship, that changes coordinates in a source coordinate reference system to coordinates in a target coordinate reference system, or that changes coordinates at a source coordinate epoch to coordinates at a target coordinate epoch within the same coordinate reference system

[**SOURCE:** ISO 19111]

3.5. coordinate reference system

coordinate system that is related to an object by a datum

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

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

[**SOURCE:** ISO 19111]

3.6. coordinate tuple

tuple composed of coordinates

Note 1 to entry: The number of coordinates in the coordinate tuple equals the dimension of the coordinate system. The order of coordinates in the coordinate tuple is identical to the order of the axes of the coordinate system.

[**SOURCE:** ISO 19111]

3.7. earth-centered inertial

a reference frame having its origin at the center of mass of Earth and which is fixed with respect to the stars

Note 1 to entry: For objects in space, the equations of motion that describe orbital motion are simpler in a non-rotating frame such as ECI.

3.8. **earth-fixed**

a reference frame which remains fixed with respect to Earth's surface in its rotation, and is then rotated with respect to stars

Note 1 to entry: Geodetic CRS defined by ISO 19111 are Earth-Fixed.

3.9. **pose**

representation of a frame transform mapping the space of an outer frame to the space of an inner frame

[**SOURCE:** GeoPose]

3.10. **Minkowski spacetime**

a combination of three-dimensional Euclidean space and time into a four-dimensional manifold where the spacetime interval between any two events is independent of the inertial frame of reference in which they are recorded

3.11. Abbreviated terms

CRS	Coordinate Reference System
CS	Coordinate System
CSV	Comma Separated Values
ECEF	Earth-Centered — Earth-Fixed
ECI	Earth-Centered Inertial

GGXF	Gridded Geodetic data eXchange Format
GML	Geographic Markup Language
WKT	Well-Known Text
1D	One-dimensional
3D	Three-dimensional
4D	Four-dimensional

4

INTRODUCTION

4 INTRODUCTION

A Coordinate Reference System (CRS) is a framework used to precisely measure locations on the surface of the Earth as coordinates. A CRS is the application of the abstract mathematics of coordinate systems and analytic geometry to geographic space. A particular CRS specification for geospatial data comprises a choice of Earth ellipsoid, anchor point(s), map projection (except in geographic CRS), axis directions, and unit of measure. Thousands of reference systems have been specified for use around the world or in specific regions and for various purposes. The multitude of CRSs necessitate transformations between different CRSs. CRSs are now a crucial basis for the sciences and technologies of Geoinformatics, including cartography, geographic information systems, surveying, remote sensing, and civil engineering. This has led to their standardization in international specifications such ISO 19111 — Referencing by Coordinates. This joint OGC/ISO standard is also published by OGC as Abstract Specification, Topic 2: referencing by coordinate. In addition, some software exists that supports the transformation from one CRS to another.

The TestBed-18 activity documented in this ER aims to go beyond the surface of the Earth and enable full location determination, orientation, and trajectory description of objects orbiting celestial bodies or in free flight in our solar system. This ER evaluates current standards with respect to the exact positioning of sensors at any location within the solar system. Then this document proposes extensions to current standards to cover broader needs such as special relativity with Minkowski spacetime. In mathematical physics, Minkowski space (or Minkowski spacetime) is a combination of three-dimensional Euclidean space and time into a four-dimensional manifold where the spacetime interval between any two events is independent of the inertial frame of reference in which they are recorded. [Wikipedia, Nov 8, 2022]

5

CURRENT STANDARDS

5 CURRENT STANDARDS

The ISO 19111 standard, published conjointly by OGC as Topic 2 — *Referencing by Coordinates*, provides the conceptual model for the description of coordinate reference systems (CRS) for locating the position of objects using coordinate tuples. This conceptual model is the foundation for three other standards:

- ISO 19162 — Well-known text representation of coordinate reference systems;
- ISO 19136 — Geography Markup Language (GML); and
- OGC GeoAPI — a set of Java interfaces for allowing different implementations to have a common API.

In an older version published in 2007, the title of ISO 19111 was *Spatial* Referencing by Coordinates. In the revision published in 2019, the *Spatial* word has been removed from the title. This change reflects the incorporation into ISO 19111:2019 of two elements which were previously defined in separated ISO standards:

- **From ISO 19108:2002:** temporal coordinate reference systems which use date & time or quantities that vary monotonically with time; and
- **From ISO 19111-2:2009:** parametric coordinate reference systems which use a non-spatial parameter (e.g., pressure) that vary monotonically with height or depth, not necessarily in a simple manner. The parameter is considered to be a third, usually vertical, axis.

This chapter explains how ISO 19111 objects can be used for representing a chain of coordinate operations from the CRS of a spacecraft to the CRS of an observatory on Earth. In principle it does not matter how those objects are represented (WKT, GML, or Java classes) since all those representations follow the same conceptual model. In practice there are differences in the capabilities of each representation. However, the general concepts stay the same. A geodetic CRS is called `GeodeticCRS` in all representations, and can be associated to an `EllipsoidalCS`, `SphericalCS`, or `CartesianCS` in all representations, and so forth.

The OGC GeoPose draft standard is another way to express a chain of coordinate operations. However, GeoPose uses a different conceptual model that appears to be independent of ISO 19111 (as of November 2022). The GeoPose conceptual model could be seen as a subset of the ISO 19111 conceptual model organized in a different way. GeoPose will be discussed in more detail in Clause 6. The balance of this ER focuses on ISO 19111. That standard is freely available as OGC Topic 2: Referencing by coordinates, which is identical in normative content with the latest edition (2019) of ISO 19111 (only the formatting is different).

5.1. ISO 19111 overview

ISO 19111 supports mixing spatial, parametric, or temporal aspects in a single coordinate reference system. Such a coordinate reference system is called a **compound CRS** and the spatial, parametric, or temporal parts are **components** of the compound CRS.

A coordinate reference system (CRS) may be geodetic and apply to a national or regional scale. Such a coordinate reference system is called a **geodetic CRS** or **projected CRS**.

A coordinate reference system may also apply locally such as for a building or construction site. Such a coordinate reference system is called a **derived CRS** or **engineering CRS**, depending how they are defined.

Finally, a coordinate reference system may be referenced to a moving platform such as a car, a ship, an aircraft, or a spacecraft. Such a coordinate reference system is called an **engineering CRS**. The CRS of a spacecraft can be related to a second coordinate reference system which is referenced to the Earth through a transformation that includes a time element. This is discussed in more details in Clause 5.2.

The definition of a coordinate reference system does not change with time. In object-oriented programming, coordinate reference system objects shall be immutable. However, the defining parameters of a CRS may include a rate of change of the parameter. By this mechanism, and others, ISO 19111 supports the construction of CRS in which the coordinate values may change with time, for example due to tectonic plate motion. A reference frame with such time evolution is called a **dynamic reference frame** and should not be confused with a **moving feature** or **dynamic feature**. Both dynamic reference frame and moving feature describe the motion of points. But in the ISO 19111 context, a dynamic reference frame accounts for the deformations of the object associated to the frame, not for the displacement of that frame relative to another frame, such as the displacement of a moving feature relative to Earth. ISO 19111 currently restricts dynamic datums to geodetic and vertical reference frames. However, if the concept was generalized to spacecrafts, the motions of points in a dynamic datum would describe the deformation of the spacecraft itself. This idea will be explored in Clause 5.9. For now, spacecraft are considered as rigid bodies and ignore dynamic datums. In other words, only static (in ISO 19111 sense) datum are considered.

The following UML is taken from OGC Topic 2 and shows the main (not all) classes. The coordinate reference systems mentioned in the above paragraphs can be seen. The key elements to retain from this UML for the purposes of this ER are as follows.

- There is a base `CRS` class with many sub-classes specialized for different usages.
- All CRSs except `CompoundCRSs` are associated with a datum (or an ensemble of datum) and a coordinate system.
- `DerivedCRSs` are associated with a base CRS and a `Conversion`.

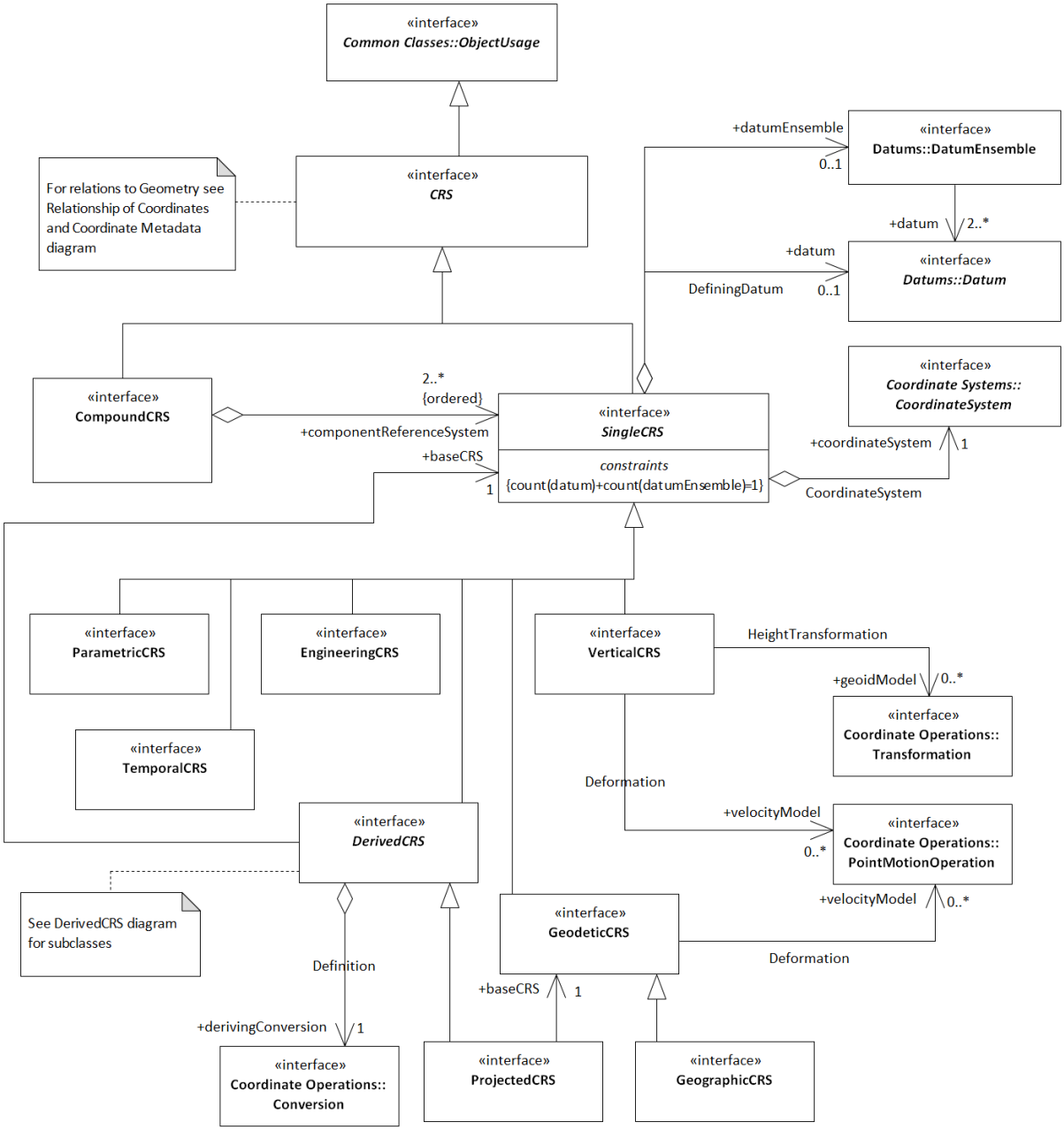

Figure 1 — UML of main ISO 19111 classes (source: OGC Topic 2)

5.2. CRS referenced to a moving platform

For creating a coordinate reference system associated with a moving platform, an engineering datum is first defined. All ISO 19111 data contains an `anchorDefinition` field of type `CharacterString`. That field contains a description of the relationship used to anchor a coordinate system to the object. The anchor may be an identified physical point with the

orientation defined relative to the object. The engineering datum itself is not time-dependent, but any transformations of the associated coordinates to an Earth-fixed or other coordinate reference system shall contain time-dependent parameters. An example is given in Clause 5.7.

In this ER, a CRS referenced to a hypothetical spacecraft as illustrated below is defined. This will be used as the source of a chain of coordinate transformations described step-by-step in subsequent sections. As a scenario, imagine that an astronomical body has been observed by the spacecraft. The body coordinates are relative to the spacecraft, using the CRS illustrated below. The goal is to transform those coordinates to a CRS relative to an observatory on Earth, using only existing OGC/ISO standards with as few extensions as possible.

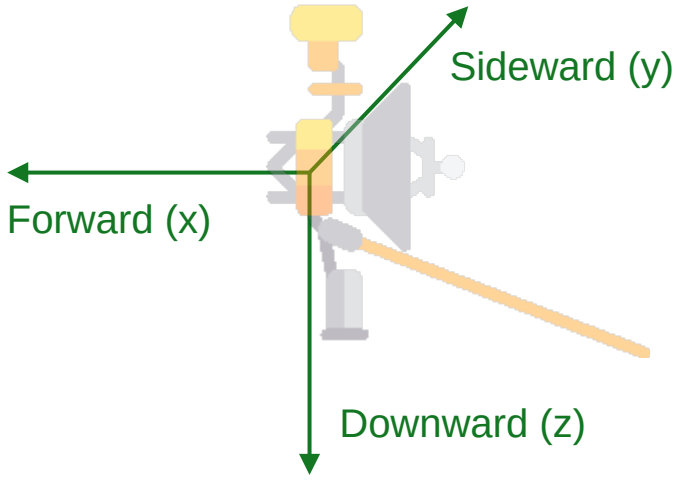

Figure 2 — Engineering CRS for Voyager (illustration)

The above CRS can be encoded in Well Known Text (WKT) format such as in the figure below. The text in the `EngineeringCRS` element can be anything and should be descriptive. The text in the `Anchor` element is mainly for human consumption. But the text in `EngineeringDatum` (i.e., the datum name) is significant. Implementations often use the datum name or identifier for deciding if two `Datum` elements are the same frame, because there is no other property in ISO 19111 datum that could be used. Note that despite a common practice, there is no need for a "D_" prefix or for replacing spaces by underscores.

```
ENGINEERINGCRS["A spacecraft-centred CRS for Voyager",
  ENGINEERINGDATUM["Voyager spacecraft frame",
    ANCHOR["Spacecraft centre of gravity"]],
  CS[Cartesian, 3],
    AXIS["Forward (x)",  forward],
    AXIS["Sideward (y)", starboard],
    AXIS["Downward (z)", down],
    LENGTHUNIT["metre", 1]]
```

Figure 3 — Engineering CRS for Voyager (WKT)

Following is an equivalent snippet in GML. This is a fragment of the `VoyagerToObservatory.xml` file in the GitHub repository. This fragment provides the same content as the above WKT, with the addition of scopes and identifiers (mandatory in GML), more explicit separation of axis names and abbreviations, and explicit name spaces for controlled vocabularies. The issue of GML verbosity compared to WKT is discussed in Annex B.

```
<gml:EngineeringCRS gml:id="VoyagerCRS">
```

```
  <gml:identifier codeSpace="TB18-D025">VoyagerCRS</gml:identifier>
  <gml:name>A spacecraft-centred CRS for Voyager</gml:name>
  <gml:scope>TestBed 18 demonstration.</gml:scope>
  <gml:cartesianCS>
    <gml:CartesianCS gml:id="VoyagerCS">
      <gml:identifier codeSpace="TB18-D025">VoyagerCS</gml:identifier>
      <gml:name>Spacecraft coordinate system</gml:name>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"VoyagerAxisForward">
          <gml:identifier codeSpace="TB18-D025">VoyagerAxisForward</gml:
identifier>
          <gml:name>Forward</gml:name>
          <gml:axisAbbrev>x</gml:axisAbbrev>
          <gml:axisDirection codeSpace="ISO">forward</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"VoyagerAxisStarboard">
          <gml:identifier codeSpace="TB18-D025">VoyagerAxisStarboard</gml:
identifier>
          <gml:name>Sideward</gml:name>
          <gml:axisAbbrev>y</gml:axisAbbrev>
          <gml:axisDirection codeSpace="ISO">starboard</gml:axisDirection> <!--
 Not well defined in this example. -->
        </gml:CoordinateSystemAxis>
      </gml:axis>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"VoyagerAxisDown">
          <gml:identifier codeSpace="TB18-D025">VoyagerAxisDown</gml:
identifier>
          <gml:name>Downward</gml:name>
          <gml:axisAbbrev>z</gml:axisAbbrev>
          <gml:axisDirection codeSpace="OGC">down</gml:axisDirection> <!-- Not
well defined in this example. -->
        </gml:CoordinateSystemAxis>
      </gml:axis>
    </gml:CartesianCS>
  </gml:cartesianCS>
  <gml:engineeringDatum>
    <gml:EngineeringDatum gml:id="CenterOfMass">
      <gml:identifier codeSpace="TB18-D025">VoyagerFrame</gml:identifier>
      <gml:name>Voyager spacecraft frame</gml:name>
      <gml:scope>TestBed 18 demonstration.</gml:scope>
      <gml:anchorDefinition>Spacecraft centre of gravity</gml:anchorDefinition>
    </gml:EngineeringDatum>
  </gml:engineeringDatum>
</gml:EngineeringCRS>
```

Figure 4 — Engineering CRS for Voyager (GML)

A CRS such as the one above does not contain relationship(s) with any other CRSs. The relationship between a spacecraft and the reference frame of a planet is encoded separately. The link uses ISO 19111 `CoordinateOperation` element and is described in Clause 5.6.

5.3. Local tangent plane to ECEF

ISO 19111 supports the definition of new coordinate reference systems as the application of a coordinate conversion from another pre-existing coordinate reference system. Such a coordinate reference system is called a **derived CRS**, and the coordinate reference system from which it was derived is called the **base CRS**. A derived CRS inherits its datum or reference frame from its base CRS. This constraint is enforced by the use of a coordinate *conversion* instead of a coordinate *transformation* (see Clause 5.6) in the derived CRS definition. This implies that if a base CRS is a geodetic CRS referenced to Earth, then the derived CRS is still a geodetic CRS referenced to Earth. This construct cannot be used for transforming coordinates from a geodetic reference frame to the engineering reference frame of a spacecraft. However, derived CRS can be used for the definition of a local coordinate reference system tangent to the geoid or ellipsoid of a planet (among other possibilities).

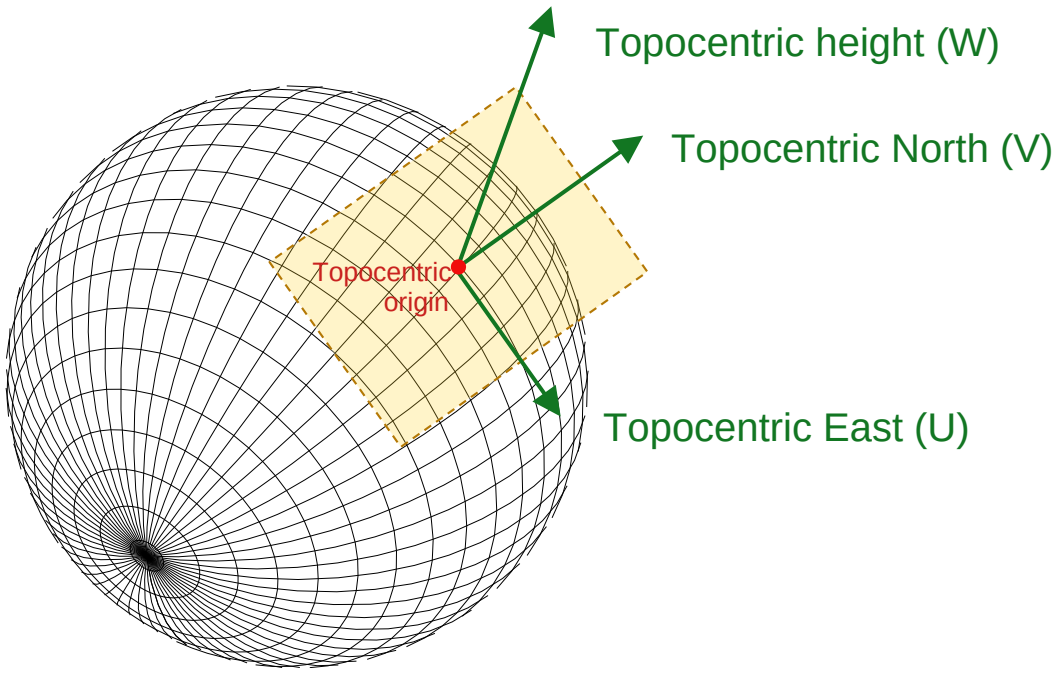

Figure 5 — Derived CRS for launch platform (Illustration)

The following WKT snippet describes the CRS illustrated above. When the `METHOD` element references an operation method defined by some authority like EPSG (which is the case in this example), the CRS definition contains enough information to allow map projection libraries to convert coordinates from Earth to the platform, and conversely. Operation methods are discussed in more details in Clause 5.3.1.

```
GEODETICCRS["Local Tangent Plane - East North Up (LTP-ENU)",
  BASEGEODCRS["WGS 84",
    DATUM["World Geodetic System 1984",
      ELLIPSOID["WGS 84", 6378137, 298.257223563]],
    ID["EPSG",4979]],
  DERIVINGCONVERSION["WGS84 to LTP-ENU",
    METHOD["Geographic/topocentric conversions", ID["EPSG",9837]],
```

```
    PARAMETER["Latitude of topocentric origin", 28.583333, ANGLEUNIT["degree",
0.0174532925199433], ID["EPSG",8834]],
    PARAMETER["Longitude of topocentric origin", -80.583056,
ANGLEUNIT["degree", 0.0174532925199433], ID["EPSG",8835]],
    PARAMETER["Ellipsoidal height of topocentric origin", 100,
LENGTHUNIT["metre", 1], ID["EPSG",8836]]],
  CS[Cartesian, 3],
  AXIS["Topocentric East (U)", east, ORDER[1], LENGTHUNIT["metre", 1]],
  AXIS["Topocentric North (V)", north, ORDER[2], LENGTHUNIT["metre", 1]],
  AXIS["Topocentric height (W)", up, ORDER[3], LENGTHUNIT["metre", 1]],
  USAGE[
    SCOPE["Voyager 2 launch"],
    AREA["Space Launch Complex 41"],
    BBOX[28, -81, 29, -80]]]
```

Figure 6 — Derived CRS for observatory platform (WKT)

5.3.1. Operation method in derived CRS

A derived CRS contains (indirectly) an operation method (`METHOD[…]`). The operation method is a critical element of information which cannot be described fully in WKT or GML. “Method” is rather a controlled vocabulary for identifying a set of formulas published in some external document. A good example of such document is EPSG guidance note #7-2 for map projection formulas. Another example is the Pole rotation method defined by the OGC MetOcean working group. A third, more incomplete example is shown in Annex A.3. This example lists the operation methods derived from GeoPose or needed by `VoyagerToObservatory.xml`. That annex is much less complete than previous examples because it does not describe the formulas. That work would need to be done by an OGC Standard Working Group (SWG).

The formulas described in the above-cited documents need to be hard-coded using some programming language (C/C++, Java, etc.) in map projection libraries. Then the WKT and GML formats can reference those formulas by specifying the operation method name and identifier (*”Geographic/topocentric conversions”* and EPSG:9837 in above example). GML goes one step further than WKT by requiring a more detailed `<gml:OperationMethod>` element, which includes the list of parameters accepted by the operation method (only their descriptions, not the values, which are provided separately). The following UML is taken from OGC Topic 2 and shows the classes for describing an operation method. The key elements to retain from this UML for the purposes of this ER are as follows.

- `OperationMethod` has references to a formula and a set of parameters.
- Formula is either a human-readable text or a citation to some publication; there is no executable code.
- Description of parameters (`GeneralOperationParameter`) are separated from values (`GeneralParameterValue`).

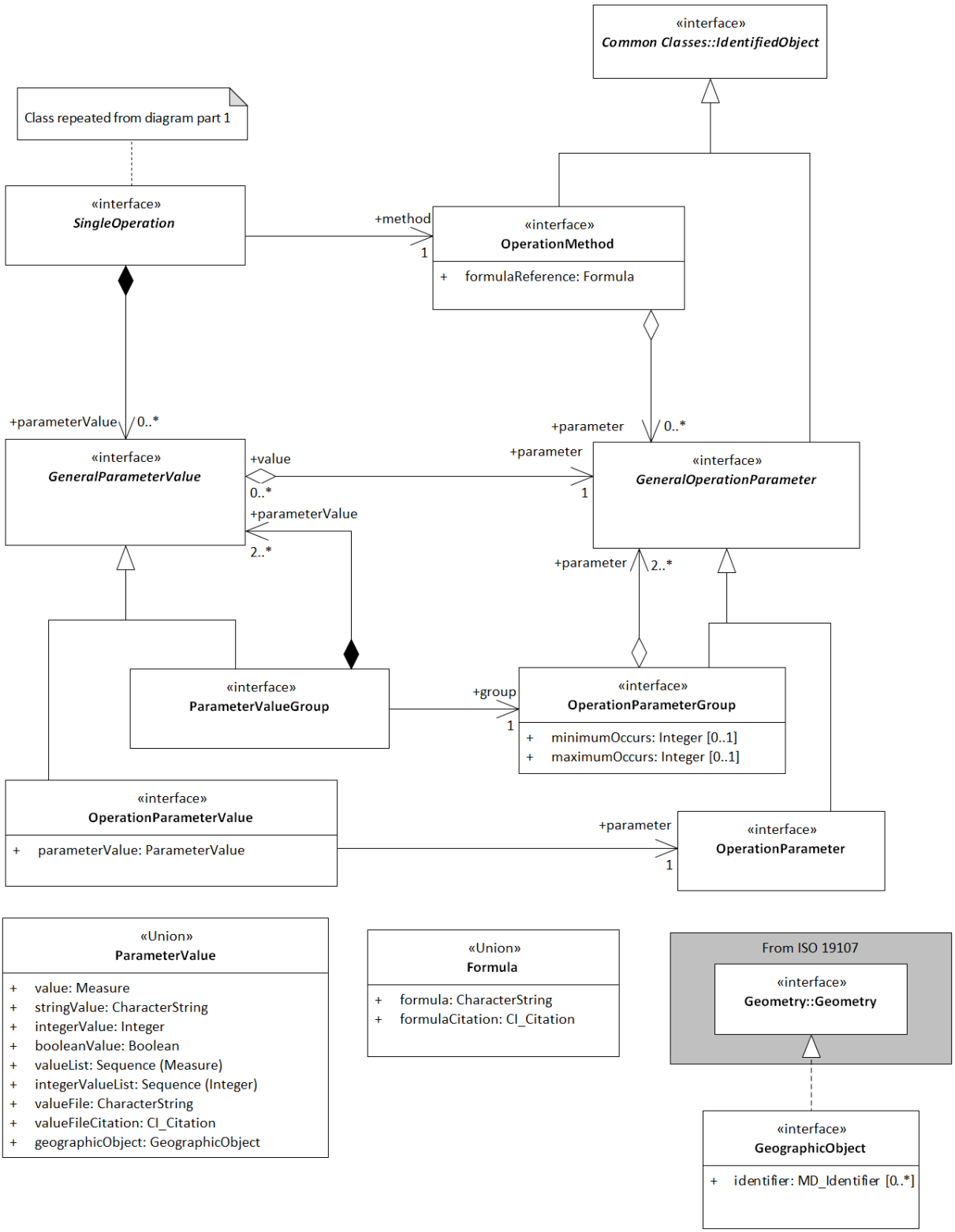

Figure 7 — UML of operation method classes (source: OGC Topic 2)

The `<gml:OperationMethod>` element can be either embedded within the derived CRS definition or be provided elsewhere and referenced using `xlink`. The `VoyagerToObservatory`.

xml file in the TB18 GitHub repository uses the embedded approach for implementing the full "Voyager to Observatory" scenario in a single standalone file. By contrast the examples in this ER use the xlink approach, which separates the operation method from the CRS definition. This separation is desirable because the operation method is typically defined by some authority such as EPSG and shared by an arbitrary number of derived CRS instances. For example, EPSG defines a few tens of operation methods. These methods are shared by thousands of projected CRS definitions in the EPSG database and countless WKT strings outside the EPSG database which rely on EPSG method definitions.

The first GML fragment (below) provides an operation method definition which is basically a simplified copy of a definition provided by the EPSG database. Note that formulas are specified only by the *"See IOGP Guidance Note #7-2"* text. A human must read that document for implementing the formulas.

```
<gml:OperationMethod gml:id="ECEF_to_LP">
  <gml:identifier codeSpace="IOGP">urn:ogc:def:method:EPSG::9837</gml:
identifier>
  <gml:name codeSpace="EPSG">Geographic/topocentric conversions</gml:name>
  <gml:formula>See IOGP Guidance Note #7-2.</gml:formula>
  <gml:sourceDimensions>3</gml:sourceDimensions>
  <gml:targetDimensions>3</gml:targetDimensions>
  <gml:parameter>
    <gml:OperationParameter gml:id="epsg-parameter-8834">
      <gml:identifier codeSpace="IOGP">urn:ogc:def:parameter:EPSG::8834</gml:
identifier>
      <gml:name>Latitude of topocentric origin</gml:name>
    </gml:OperationParameter>
  </gml:parameter>
  <gml:parameter>
    <gml:OperationParameter gml:id="epsg-parameter-8835">
      <gml:identifier codeSpace="IOGP">urn:ogc:def:parameter:EPSG::8835</gml:
identifier>
      <gml:name>Longitude of topocentric origin</gml:name>
    </gml:OperationParameter>
  </gml:parameter>
  <gml:parameter>
    <gml:OperationParameter gml:id="epsg-parameter-8836">
      <gml:identifier codeSpace="IOGP">urn:ogc:def:parameter:EPSG::8836</gml:
identifier>
      <gml:name>Ellipsoidal height of topocentric origin</gml:name>
    </gml:OperationParameter>
  </gml:parameter>
</gml:OperationMethod>
```

Figure 8 — Geographic/topocentric conversion method (GML)

The CRS definition can then reference the operation method definition using the xlink attribute. The following fragment assumes that the XML fragment shown in the above figure was in the same GML document. The ECEF_to_LP label is arbitrary as any name is okay because it is local to the document:

```
<gml:method xlink:href="#ECEF_to_LP"/>
```

Referencing definitions in external documents is also possible. For example, the above <gml:OperationMethod> fragment could be completely omitted from the XML document and the above link replaced by the following one:

```
<gml:method xlink:href="https://epsg.org/api/v1/CoordOperationMethod/9837/
export?format=gml"/>
```

The XML fragment shown below uses such an external link for `<gml:baseCRS>` elements in order to keep the fragment smaller. However, the `VoyagerToObservatory.xml` file contains the full definition.

```
<gml:DerivedCRS gml:id="ObservatoryPlatformCRS">
  <gml:identifier codeSpace="TB18-D025">ObservatoryPlatformCRS</gml:identifier>
  <gml:name>Local Tangent Plane - East North Up (LTP-ENU)</gml:name>
  <gml:scope>TestBed 18 demonstration.</gml:scope>
  <gml:conversion>
    <gml:Conversion gml:id="WGS84_to_LP">
      <gml:identifier codeSpace="TB18-D025">WorldToPlatform</gml:identifier>
      <gml:name>WGS84 to LTP-ENU</gml:name>
      <gml:scope>TestBed 18 demonstration.</gml:scope>
      <gml:method xlink:href="#ECEF_to_LP"/>
      <!--
        This is where the parameter values for the conversion from ECEF to the
local tangent plane are provided.
        Note the use of xlink: without them, we would have to repeat
descriptions from a previous GML fragment
        inside the following <gml:operationParameter> elements.
      -->
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:value uom="urn:ogc:def:uom:EPSG::9102">28.583333</gml:value>
          <gml:operationParameter xlink:href="#epsg-parameter-8834"/> <!--
Latitude of topocentric origin -->
        </gml:ParameterValue>
      </gml:parameterValue>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:value uom="urn:ogc:def:uom:EPSG::9102">-80.583056</gml:value>
          <gml:operationParameter xlink:href="#epsg-parameter-8835"/> <!--
Longitude of topocentric origin -->
        </gml:ParameterValue>
      </gml:parameterValue>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:value uom="urn:ogc:def:uom:EPSG::9001">100</gml:value>
          <gml:operationParameter xlink:href="#epsg-parameter-8836"/> <!--
Ellipsoidal height of topocentric origin -->
        </gml:ParameterValue>
      </gml:parameterValue>
    </gml:Conversion>
  </gml:conversion>
  <!--
    The following block contains the Coordinate Reference System (CRS)
associated to the observatory platform.
    In the GeoPose specification, this is called "outer frame." GeoPose
restricts the CRS of outer frame
    to EPSG:4979, which is the CRS used in this example. But in GML any
geodetic CRS could be used below.
    The XML file on the GitHub repository contains a copy of the full
definition, but for this report we make
    the example shorter by using a link to EPSG registry.
  -->
  <gml:baseCRS xlink:href="https://epsg.org/api/v1/CoordRefSystem/4979/export?f
ormat=gml"/>
  <!--
```

```
    The following block defines the axes of the observatory platform as a
Cartesian coordinate system.
    The platform origin is located at 28.583333°N, 80.583056°W and 100 meters
above ellipsoid.
    Those values are specified in the <gml:conversion> block above.
  -->
  <gml:derivedCRSType codeSpace="EPSG">engineering</gml:derivedCRSType>
  <gml:coordinateSystem>
    <gml:CartesianCS gml:id="ObservatoryPlatformCS">
      <gml:identifier codeSpace="TB18-D025">ObservatoryPlatformCS</gml:
identifier>
      <gml:name>Topocentric easting and northing</gml:name>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"PlatformAxisEast">
          <gml:identifier codeSpace="TB18-D025">PlatformAxisEast</gml:
identifier>
          <gml:name>Topocentric East</gml:name>
          <gml:axisAbbrev>U</gml:axisAbbrev>
          <gml:axisDirection codeSpace="ISO">east</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"PlatformAxisNorth">
          <gml:identifier codeSpace="TB18-D025">PlatformAxisNorth</gml:
identifier>
          <gml:name>Topocentric North</gml:name>
          <gml:axisAbbrev>V</gml:axisAbbrev>
          <gml:axisDirection codeSpace="ISO">north</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9001" gml:id=
"PlatformAxisHeight">
          <gml:identifier codeSpace="TB18-D025">PlatformAxisHeight</gml:
identifier>
          <gml:name>Topocentric height</gml:name>
          <gml:axisAbbrev>W</gml:axisAbbrev>
          <gml:axisDirection codeSpace="ISO">up</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
    </gml:CartesianCS>
  </gml:coordinateSystem>
</gml:DerivedCRS>
```

Figure 9 — Derived CRS for observatory platform (GML)

5.4. Earth-centered inertial CRS

An Earth-centered inertial (ECI) coordinate reference system has its origin at the center of mass of the Earth and its axes oriented in directions fixed with respect to the stars. This contrasts with the “Earth-centered — Earth-fixed” (ECEF) frames used in classical geodesy. In geodesy, frames remain fixed with respect to Earth’s surface in its rotation, and then rotated with respect to stars. ECEF coordinates (typically latitude, longitude, and altitude) are convenient

for representing the positions and velocities of terrestrial objects. But for objects in space, the equations of motion that describe orbital motion are simpler in a non-rotating frame such as ECI.

The extent to which an ECI frame is inertial is limited by the non-uniformity of the surrounding gravitational field. For example, the Moon's gravitational influence on a high-Earth orbiting satellite is significantly different than its influence on Earth. Therefore observers in an ECI frame would have to account for this difference in acceleration in their laws of motion. The closer the observed object is to the ECI-origin, the less significant the effect of the gravitational disparity is.

Defining a truly inertial CRS is difficult. Only "quasi-inertial" approximations are used in practice. The *3D+ Standards Framework Engineering Report* (OGC 22-036) discusses the different kinds of quasi-inertial CRS. For the purpose of this ER (Clause 5.4.1, Annex A.3.2, and `VoyagerToObservatory.xml` GML), this complexity is simplified with an hypothetical quasi-inertial CRS defined in such a way that the geodetic CRS is rotating at a constant speed relative to that quasi-inertial CRS.

The WKT below defines a quasi-inertial CRS (ECI) without specifying its relationship with a geodetic CRS (ECEF). The relationship between ECEF and ECI is not specified here because those two CRSs are defined by separated sets of observations (national survey *versus* astronomical observations). ECI is not defined in terms of ECEF, or conversely, ECEF is not defined in terms of ECI. Consequently, saying that one CRS is derived from the other is not appropriate. As such the use of `DerivedCRS` as in Clause 5.3 would not be correct. The relationship between ECI and ECEF is defined separately in Clause 5.4.1.

```
ENGINEERINGCRS["Quasi-inertial (Equatorial CS)",
  ENGINEERINGDATUM["ECI frame",
    ANCHOR["Earth center of mass"]],
  CS[Spherical, 3],
  AXIS["Declination (DEC)", north, ORDER[1], ANGLEUNIT["degree",
0.0174532925199433]],
  AXIS["Right Ascension (RA)", east, ORDER[2], ANGLEUNIT["degree",
0.0174532925199433]],
  AXIS["Radius (R)", up, ORDER[3], LENGTHUNIT["metre", 1]],
  REMARK["North stands for ""north celestial pole"" and East for ""6 hours
from vernal equinox""."]]
```

Figure 10 — Quasi-inertial CRS (WKT)

NOTE The above WKT uses the `EngineeringCRS` type by default. The non-derived CRS types are `GeodeticCRS`, `GeographicCRS`, `ProjectedCRS`, `VerticalCRS`, `TemporalCRS`, `ParametricCRS`, and `EngineeringCRS`. In principle the latter is for contextually local coordinate reference systems such as vehicles, vessels, aircraft, or platforms near the surface of the Earth. However, in practice, an `EngineeringCRS` tends to be used for everything that does not fit within the other CRS types. This section uses `EngineeringCRS` for compatibility with the ISO 19111 standard. However, Clause 8.4 provides an alternative definition using a new `InertialCRS` type proposed as an extension to the ISO 19111 standard.

5.4.1. Transformations between ECI and ECEF

Due to movement of the Australian plate, the transformation parameters between the Geocentric Datum of Australia (GDA) and the International Terrestrial Reference System (ITRS)

change over time. An expansion of the transformation parameters from seven to fourteen parameters has been proposed to address this issue (Dawson and Woods). This expansion adds seven new parameters to express the velocity (first derivative) of the original seven parameters. As a result, a coordinate transformation for a moving reference system becomes possible. The EPSG database contains such transformations.

The proposed solution was developed for low velocities. However, if the time basis for velocity is seconds rather than years, then this same approach can work for any reference system moving at a constant velocity. The simplification used in Testbed 18 (see Clause 5.4) supports using this approach and adapting it to the astronomical context. The EPSG *Time-dependent Coordinate Frame rotation (geographic 3D)* operation has the following parameters which can be used as a source of inspiration.

- X-axis translation
- Y-axis translation
- Z-axis translation
- X-axis rotation
- Y-axis rotation
- Z-axis rotation
- Scale difference
- Rate of change of X-axis translation
- Rate of change of Y-axis translation
- Rate of change of Z-axis translation
- Rate of change of X-axis rotation
- Rate of change of Y-axis rotation
- Rate of change of Z-axis rotation
- Rate of change of scale difference
- Parameter reference epoch

A slight variation of EPSG *Time-dependent Coordinate Frame rotation* methods is needed because the source and target CRSs are different. EPSG methods describe an operation between two `GeodeticCRS` instances, while in our case one instance is a quasi-inertial CRS. Furthermore, EPSG methods associate the source/target CRS with Cartesian or ellipsoidal coordinate systems, while spherical coordinate systems would be more convenient for the transformation described in this section. For those reasons a new operation method named *Time-dependent longitude rotation (equatorial CS)* is used in the WKT below.

```
COORDINATEOPERATION["Earth-Centered Inertial (ECI) to Earth Fixed (ECEF)",
  SOURCECRS[WKT from Clause 5.4 omitted for brevity],
  TARGETCRS[
```

```
    GEODETICCRS["Geocentric (Spherical CS)",
      DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84", 6378137, 298.257223563]],
      CS[Spherical, 3],
      AXIS["Latitude (P)", north, ORDER[1], ANGLEUNIT["degree",
0.0174532925199433]],
      AXIS["Longitude (L)", east, ORDER[2], ANGLEUNIT["degree",
0.0174532925199433]],
      AXIS["Radius (R)", up, ORDER[3], LENGTHUNIT["metre", 1]]]],
  METHOD["Time-dependent longitude rotation (equatorial CS)"],
  PARAMETER["Longitude axis rotation", 0, UNIT["degree", 0.0174532925199433]],
  PARAMETER["Rate of change of longitude axis rotation", 0.985609112046479,
UNIT["degree/day", 1507.9644737231]],
  PARAMETER["Parameter reference epoch", 2000, UNIT["year", 31556925.445]]]
```

Figure 11 — ECI to geodetic CRS (WKT)

The same coordinate operation is expressed in GML format in the `VoyagerToObservatory.xml` file in the `<gml:Transformation gml:id="InertialToEarthFixed">` element. This operation method is included in the list of method proposals in Annex A.3.2. Because this operation is specific to Testbed 18, map projection libraries do not know how to use the specified parameters for performing a coordinate transformation. But the plugin mechanism described in Clause 8.9 (based on Java service provider mechanisms) allows the demo application (Clause 9.1) to register custom code provided by `TimeDependentLongitudeRotation.java`. This code does not need to handle GML parsing. The same code would be invoked by the same plugin mechanism if the transformation was specified in a JSON file, in a WKT encoding, or programmatically.

In real applications, transformations will be more complex than the example in this section. Precession, nutation, polar motion, and other perturbations would need to be taken in account. But this additional complexity does not change how to use the ISO 19111 standard; it only adds more parameters. Clause 5.6 discusses more about coordinate operations.

5.5. Addition of temporal axis

All coordinate reference systems in the above clauses were spatial. ISO 19111 has a mechanism for adding a temporal axis to almost any CRS, but it assumes that the temporal dimension is independent from all other dimensions. This is not true when Einstein's relativity becomes important, but that case will be discussed in Clause 8.5. This section assumes that relativistic effects can be ignored, in which case the process is as below:

- defines a `TemporalCRS` as a one-dimensional CRS, independent from the spatial CRS; and
- defines a `CompoundCRS` containing two components: the spatial CRS and above-cited temporal CRS.

The number of dimensions of the `CompoundCRS` is the sum of the number of dimensions of all its components. In this example the result is a four-dimensional CRS, but more components such as `ParametricCRS` could be added. The following WKT snippet shows a compound CRS using truncated Julian days for the temporal component. Truncated Julian is the time measured as

days since May 24, 1968 at 00:00 UTC and is defined relative to Julian day (JD) as TJD = JD − 2440000.5. This epoch was introduced by NASA for the space program.

```
COMPOUNDCRS["Geocentric (Spherical CS) with time",
  GEODETICCRS[WKT from Clause 5.4.1 omitted for brevity],
  TIMECRS["Truncated Julian days",
    TDATUM["Truncated Julian epoch", TIMEORIGIN[1968-05-24T00:00:00Z]],
    CS[TemporalCount, 1],
    AXIS["Time", future, TIMEUNIT["day", 86400]]]]
```

Figure 12 — Compound CRS with a temporal component (WKT)

The addition of a temporal axis to an existing CRS is a complication when those CRSs need to be used with coordinate operations designed for spatial CRSs (for example map projections). This problem is handled by `PassThroughOperation` (Clause 5.6.3), which is defined in ISO 19111 and GML but not in WKT. Consequently, the WKT format is sufficient for expressing CRS with time axis but may not be sufficient for expressing coordinate operations with a temporal component.

5.5.1. Coordinate epoch

As an alternative to 4-dimensional coordinate tuples, it is possible to keep coordinates as three-dimensional tuples and instead specify the time as a single metadata for the whole set of coordinate tuples. This can be done using the WKT `COORDINATEMETADATA` keyword with an `EPOCH` element. Example:

```
COORDINATEMETADATA[
  ENGINEERINGCRS["A spacecraft-centred CRS for Voyager",
    See Clause 5.2 for the remaining of this WKT],
  EPOCH[2022.47]]
```

Figure 13 — Coordinate metadata (WKT)

The advantage of this approach is to keep CRS and coordinate operation definitions simpler. `CompoundCRS` and `PassThroughOperation` are no longer needed, even `TemporalCRS` may be avoided. Coordinate tuples are also more compact since they do not contain a time coordinate. The inconvenience is that the time coordinate must be the same for all coordinate tuples, until a new `COORDINATEMETADATA` is specified for a new set of coordinate tuples.

`CoordinateMetadata` is a feature available in WKT but not in GML. This is because GML has not yet been updated with respect to the latest ISO 19111 revision. A possible approach would be to use WKT for coordinate metadata (including CRS) and GML for coordinate operations to ingest in an EPSG-like database (Clause 8.8).

The participants in Testbed 18 did not use this approach for two reasons because the library used for the demonstration does not yet support dynamic coordinate reference systems. `CoordinateMetadata` is specified in the ISO 19111:2019 revision and the Apache SIS library has not yet been upgraded to that revision. A more sophisticated GML file is provided from which the simpler GML file can be derived, if desired. The simplification consists in removing all `<gml:CompoundCRS>`, `<gml:TimeCRS>`, and `<gml:PassThroughOperation>` elements.

5.6. Coordinate operations

ISO 19111:2019 distinguishes two kinds of coordinate operations (ignoring dynamic reference frame for now): **conversions** and **transformations**. A conversion can include translation, rotation, change of units, *etc.*, but with a result which is still associated to the same reference frame, for example the same spacecraft. By contrast a transformation involves a change of reference frame, for example from one spacecraft to another one.

The distinction between conversion and transformation types allows type safety in `DerivedCRS` definitions. A conversion (but not a transformation) can be embedded in the definition of a derived CRS. That conversion is theoretically of infinite precision (ignoring rounding errors caused by floating-point arithmetic and numerical approximations caused by Taylor series expansions) because the derived CRS is *by definition* the result of applying that conversion on the base CRS. From the above, at most one conversion (ignoring differences in metadata) can exist between any given pair of CRSs in a self-consistent geodetic dataset. A derived CRS example was given in Clause 5.3.1.

By contrast, transformations between two CRSs are defined externally (for example in a database) and are not part of the CRS definition (ignoring `BoundCRS`, which is a compromise for a common but non-recommended practice). Transformations are determined empirically, for example by astronomical observations, and consequently have stochastic errors. Many transformations may exist between the same pair of CRSs, with different properties such as accuracy, domain of validity, *etc.*

In the context of Testbed 18, the operation for changing a coordinate tuple relative to a spacecraft into a coordinate tuple relative to another spacecraft (for example) would be a *coordinate transformation*, and the transformation parameters would be provided outside the CRS definitions, in an EPSG-like database. For map projection libraries using the EPSG database in a "late-binding" way, the existing software code should be applicable with few changes (Clause 9).

The coordinate operation type (conversion versus transformation) is not specified explicitly in the WKT format. The operation type is rather inferred from whether source and target datum are the same or different. The following example specifies a relationship between a pose and the extremity of a robotic arm. This example uses the quaternions parameters derived from the draft GeoPose standard as shown in Annex A.3.1. This example uses fixed parameter values, but different approaches for time-varying operations are introduced in Clause 5.4.1, Clause 5.6.1.2 and Clause 5.7.

```
COORDINATEOPERATION["Pose to extremity of robotic arm",
  SOURCECRS[GEOGRAPHICCRS[... complete definition omitted for brevity ...]],
  TARGETCRS[ENGINEERINGCRS[... complete definition omitted for brevity ...]],
  METHOD["Pose location and orientation by quaternion"],
  PARAMETER["Latitude of pose", 28.583333, ANGLEUNIT["degree",
0.0174532925199433]],
  PARAMETER["Longitude of pose", -80.583056, ANGLEUNIT["degree",
0.0174532925199433]],
  PARAMETER["Ellipsoidal height of pose", 100, LENGTHUNIT["metre", 1]],
  PARAMETER["Quaternion X",  0.20056154657066608, SCALEUNIT["unity", 1]],
  PARAMETER["Quaternion Y", -0.08111602541464237, SCALEUNIT["unity", 1]],
  PARAMETER["Quaternion Z",  0.36606032744426537, SCALEUNIT["unity", 1]],
```

```
PARAMETER["Quaternion W", -0.90509396922613010, SCALEUNIT["unity", 1]],
OPERATIONACCURACY[0.1]]
```

Figure 14 — Coordinate transformation from a pose to the extremity of a robotic arm

The above WKT does not define a CRS. It defines a *relationship* between two coordinate reference systems. An arbitrary number of relationships can exist and can be stored in an EPSG-like database. When a coordinate operation between the two CRSs is requested, map projection libraries such as PROJ or Apache SIS iterate over all coordinate operations found in the database for that pair of CRSs. Then an operation is selected using some heuristic rules, for example:

- preference given to the largest intersection between the *temporal* domain of validity of the coordinate operation and the *time of interest* specified by the user;
- preference given to the largest intersection between the *geographic* domain of validity of the coordinate operation and the *area of interest* specified by the user; and
- preference given to the operation having the smallest `OPERATIONACCURACY` value.

5.6.1. GML encoding

The coordinate operation shown above can also be encoded in GML. While GML is generally more verbose, GML also has some advantages. One of them is that any element, for example the `EngineeringCRS` of Voyager spacecraft, can be defined only once in the document and referenced in other places using `xlink:href`. By contrast, if many coordinate operations for Voyager are defined using the WKT format, the full `ENGINEERINGCRS ["A spacecraft-centred CRS for Voyager", …]` element would need to be repeated every time. A proposal for avoiding this problem is given in Clause 8.7, but that proposal is not standard. By contrast, the current GML standard already supports this compactness. See for example the use of `xlink:href` in Clause 5.3.

Another reason for using GML is that time-varying parameters may require the ISO 19111 `OperationParameterGroup` construct, which for simplicity reasons has been excluded from the current ISO 19162 WKT standard. Both parameter groups and pass-through operations are available in GML. The main ISO 19111 features missing in GML 3.2.1 compared to WKT are coordinate metadata (Clause 5.5.1), dynamic datum (Clause 5.9), datum ensemble, and point motion operations. Those missing features could be added by upgrading GML schema from ISO 19111:2007 to ISO 19111:2019.

5.6.1.1. Source/target CRS in `<gml:Conversion>`

GML 3.2.1 has one issue which may possibly be a bug in the GML schema. According to ISO 19111, `sourceCRS` and `targetCRS` properties are mandatory in `Transformation` and optional in `Conversion`. However in the GML schema, source/target CRSs are not allowed at all in the `<gml:Conversion>` element.

Forbidding source/target CRS is acceptable when the `Conversion` is used as a **defining conversion**, for example in the definition of a `ProjectedCRS`. In that case, the source CRS and target CRS are inferred from the context: the `baseCRS` property and the `ProjectedCRS`

instance respectively. However, `<gml:Conversion>` can also be used as a standalone coordinate operation, outside any CRS definition. In that latter case, not being able to specify the source and target CRS is a problem. The `VoyagerToObservatory.xml` file works around this problem by using `<gml:Transformation>` for an element which should have been `<gml:Conversion>`.

5.6.1.2. Piecewise function

ISO 19111:2019 provides a mechanism for repeating a group of parameters many times with different values. This mechanism can be used for building a piecewise function, such as different sets of values for different time steps. This feature is currently available only in GML. The transformation shown in Clause 5.6 is repeated in the figure below, but in GML instead of WKT encoding and augmented with time-varying parameters. First, the operation method is defined. The definition below contains the parameters listed in Annex A.3.1 for the GeoPose *"Pose irregular series"* transformation.

```
<gml:OperationMethod gml:id="PoseIrregularSeries">
  <gml:identifier codeSpace="TB18-D025">PoseIrregularSeries</gml:identifier>
  <gml:name>Pose irregular series</gml:name>
  <gml:formula>See TB18 D025.</gml:formula>
  <gml:sourceDimensions>3</gml:sourceDimensions>
  <gml:targetDimensions>3</gml:targetDimensions>
  <gml:parameter>
    <gml:OperationParameter gml:id="start-instant">
      <gml:identifier codeSpace="TB18-D025">:start-instant</gml:identifier>
      <gml:name>Start instant</gml:name>
    </gml:OperationParameter>
  </gml:parameter>
  <gml:parameter>
    <gml:OperationParameter gml:id="end-instant">
      <gml:identifier codeSpace="TB18-D025">:end-instant</gml:identifier>
      <gml:name>End instant</gml:name>
    </gml:OperationParameter>
  </gml:parameter>
  <!--
    Above "Start instant" and "End instant" parameters will have exactly one
value.
    The parameters defined below are inside a group which can be repeated as
many
    times as desired with different parameter values.
  -->
  <gml:parameter>
    <gml:OperationParameterGroup gml:id="pose-irregular-frame">
      <gml:identifier codeSpace="TB18-D025">pose-irregular-frame</gml:
identifier>
      <gml:name>Frame specification (irregular)</gml:name>
      <gml:minimumOccurs>1</gml:minimumOccurs>
      <gml:maximumOccurs>1000</gml:maximumOccurs> <!-- Arbitrary limit in
number of steps. -->
      <gml:parameter>
        <gml:OperationParameter gml:id="valid-time">
          <gml:identifier codeSpace="TB18-D025">valid-time</gml:identifier>
          <gml:name>Valid time</gml:name> <!-- As ISO-8601 or duration since
Unix epoch. -->
        </gml:OperationParameter>
      </gml:parameter>
      <gml:parameter>
        <gml:OperationParameter gml:id="pose-location">
          <gml:identifier codeSpace="TB18-D025">pose-location</gml:identifier>
```

```
          <gml:name>Pose location</gml:name> <!-- As a sequence of 3
coordinate values. -->
        </gml:OperationParameter>
      </gml:parameter>
      <gml:parameter>
        <gml:OperationParameter gml:id="yaw">
          <gml:identifier codeSpace="TB18-D025">yaw</gml:identifier>
          <gml:name>Yaw</gml:name>
        </gml:OperationParameter>
      </gml:parameter>
      <gml:parameter>
        <gml:OperationParameter gml:id="pitch">
          <gml:identifier codeSpace="TB18-D025">pitch</gml:identifier>
          <gml:name>Pitch</gml:name>
        </gml:OperationParameter>
      </gml:parameter>
      <gml:parameter>
        <gml:OperationParameter gml:id="roll">
          <gml:identifier codeSpace="TB18-D025">roll</gml:identifier>
          <gml:name>Roll</gml:name>
        </gml:OperationParameter>
      </gml:parameter>
    </gml:OperationParameterGroup>
  </gml:parameter>
</gml:OperationMethod>
```

Figure 15 — Definition of Pose irregular series transformation (GML)

The following fragment assigns parameter values to the operation. The important thing to note is that yaw/pitch/roll parameter values are wrapped in a group which can be repeated as many times as desired. A documentation for the *"Pose irregular series"* operation method should be provided by an OGC standard working group with a description of all parameters, and that documentation should specify how the interpolations are done.

```
<gml:Conversion gml:id = "PoseToRoboticArm">
  <gml:identifier codeSpace="TB18-D025">PoseToRoboticArm</gml:identifier>
  <gml:name>Pose to extremity of a robotic arm</gml:name>
  <gml:scope>TestBed 18 demonstration.</gml:scope>
  <gml:method xlink:href="#PoseIrregularSeries"/> <!-- Reference to above <gml:
OperationMethod>. -->
  <gml:parameterValue>
    <gml:ParameterValue>
      <gml:stringValue>2022-10-20T06:00:00Z</gml:stringValue>
      <gml:operationParameter xlink:href="#start-instant"/>
    </gml:ParameterValue>
  </gml:parameterValue>
  <gml:parameterValue>
    <gml:ParameterValue>
      <gml:stringValue>2022-10-21T18:00:00Z</gml:stringValue>
      <gml:operationParameter xlink:href="#end-instant"/>
    </gml:ParameterValue>
  </gml:parameterValue>
  <gml:parameterValue>
    <!--
      Yaw/pitch/roll parameters for the first time step.
    -->
    <gml:ParameterValueGroup>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:stringValue>2022-10-20T06:00:00Z</gml:stringValue>
          <gml:operationParameter xlink:href="#valid-time"/>
        </gml:ParameterValue>
```

```
      </gml:parameterValue>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:valueList uom="urn:ogc:def:uom:EPSG::9001">4 6 2</gml:valueList>
 <!-- UoM is metres. Values are dummy. -->
          <gml:operationParameter xlink:href="#pose-location"/>
        </gml:ParameterValue>
      </gml:parameterValue>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:value uom="urn:ogc:def:uom:EPSG::9122">40</gml:value> <!-- UoM
is degrees. Value is dummy. -->
          <gml:operationParameter xlink:href="#yaw"/>
        </gml:ParameterValue>
      </gml:parameterValue>
      <!-- Pitch and Roll parameters omitted for brevity. -->
      <gml:group xlink:href="#pose-irregular-frame"/>
    </gml:ParameterValueGroup>
  </gml:parameterValue>
  <gml:parameterValue>
    <!--
      Yaw/pitch/roll parameters for the second time step.
      This is the same group than above, repeated with different parameter
values.
    -->
    <gml:ParameterValueGroup>
      <gml:parameterValue>
        <gml:ParameterValue>
          <gml:stringValue>2022-10-20T17:00:00Z</gml:stringValue>
          <gml:operationParameter xlink:href="#valid-time"/>
        </gml:ParameterValue>
      </gml:parameterValue>
      <!-- Pose location, Yaw, Pitch and Roll parameters omitted for brevity. -
->
      <gml:group xlink:href="#pose-irregular-frame"/>
    </gml:ParameterValueGroup>
  </gml:parameterValue>
  <!--
    More time steps could be added by continuing to repeat <gml:
ParameterValueGroup>.
  -->
</gml:Conversion>
```

Figure 16 — Piecewise function using parameter groups (GML)

See the `PoseToRoboticArm.xml` file in Testbed 18 GitHub repository for a complete GML document.

5.6.2. Chain of operations

Coordinate transformations from a source reference frame to a target reference frame may require transformations to intermediate frames. For example in the scenario defined in this ER, coordinates relative to the spacecraft are transformed to heliocentric coordinates, then to ECI coordinates, then to ECEF coordinates, and finally to coordinates relative to an observatory. For each step, more than one transformation may exist, which may cause combinatorial explosion. The capability to find such an indirect path between two CRSs is highly implementation dependent, and two libraries may choose different paths. To avoid those problems, it is desirable to declare explicitly one or many preferred paths between pairs of CRSs that are otherwise not

directly connected to each other. This is done in ISO 19111 using the `ConcatenatedOperation`, and is used notably in the EPSG database.

The following WKT defines a "Spacecraft → Heliocentric → ECI → ECEF → Platform" chain of transforms. For each step, the source CRS of that step is the target CRS of the previous step. For brevity, only CRS names are given. The rest of CRS definitions are omitted. Temporal components (Clause 5.5) and operation parameters are also omitted. A complete `CONCATENATEDOPERATION` definition in WKT format would have a lot of repetitions, which is why GML is currently preferred for this scenario (Clause 8.7). See `VoyagerToObservatory.xml` file for a complete definition in GML format.

```
CONCATENATEDOPERATION["Voyager spacecraft to Observatory platform",
  SOURCECRS[ENGINEERINGCRS["A spacecraft-centred CRS for Voyager", …]],
  TARGETCRS[GEODETICCRS["Local Tangent Plane - East North Up (LTP-ENU)", …]],
  STEP[COORDINATEOPERATION[
    SOURCECRS[ENGINEERINGCRS["A spacecraft-centred CRS for Voyager", …]],
    TARGETCRS[INERTIALCRS["Ecliptic heliocentric CRS (Spherical)", …]],
    METHOD["Trajectory to Conventional frame"], …]],
  STEP[COORDINATEOPERATION[
    SOURCECRS[INERTIALCRS["Ecliptic heliocentric CRS (Spherical)", …]],
    TARGETCRS[INERTIALCRS["Quasi-inertial (Equatorial CS)", …]],
    METHOD["To circular orbit (Spherical domain)"], …]],
  STEP[COORDINATEOPERATION[
    SOURCECRS[INERTIALCRS["Quasi-inertial (Equatorial CS)", …]],
    TARGETCRS[GEODETICCRS["WGS 84 (Spherical)", …]],
    METHOD["Longitude axis rotation"], …]],
  STEP[COORDINATEOPERATION[
    SOURCECRS[GEODETICCRS["WGS 84 (Spherical)", …]],
    TARGETCRS[GEOGRAPHICCRS["WGS 84", …]],
    METHOD["Geographic/spherical conversions"], …]],
  STEP[COORDINATEOPERATION[
    SOURCECRS[GEOGRAPHICCRS["WGS 84", …]],
    TARGETCRS[GEODETICCRS["Local Tangent Plane - East North Up (LTP-ENU)", …]],
    METHOD["Geographic/topocentric conversions"], …]]
```

Figure 17 — "Voyager spacecraft to Observatory platform" chain of transforms (WKT)

The above concatenated operation can be inserted into an EPSG-like database in the same way that any other (more direct) coordinate operations can be. They can be handled by map projection libraries in the same way as direct operations, as described in Clause 5.6.

5.6.3. Temporal axis in coordinate operations

Each coordinate transformation depends on an operation method, and each operation method assumes specific kinds of source and target reference systems. For example, an operation method may be defined from a three-dimensional geographic CRS to a three-dimensional projected CRS. In order to execute a three-dimensional coordinate operation on a four-dimensional coordinate tuple, the encoding needs to specify on which coordinate values the three-dimensional operation shall work. This is the purpose of ISO 19111 `PassThroughOperation`. Pass-through operations can be encoded in GML but have no WKT counter part.

Below is a fragment derived from the `VoyagerToObservatory.xml` file. This pass-through operation wraps an *"Earth Fixed (ECEF) spherical to geographic"* conversion which does not depend on time and works only on three-dimensional coordinates. For applying that conversion

on (c_1, c_2, c_3, c_4) coordinate tuples where c_4 is time, the encoding needs to instruct the wrapped conversion to operate on (c_1, c_2, c_3) coordinates and then to let all remaining coordinates (only c_4 in this example) pass through unchanged.

```
<gml:PassThroughOperation gml:id="SphericalToGeographic4D">
  <gml:identifier codeSpace="TB18-D025">SphericalToGeographic4D</gml:
identifier>
  <gml:name>Earth Fixed (ECEF) spherical to geographic with time</gml:name>
  <gml:scope>TestBed 18 demonstration.</gml:scope>
  <gml:sourceCRS xlink:href="#GeodeticAndTimeCRS"/>
  <gml:targetCRS xlink:href="#GeographicAndTimeCRS"/>
  <gml:modifiedCoordinate>1</gml:modifiedCoordinate> <!-- Index values start
at 1. -->
  <gml:modifiedCoordinate>2</gml:modifiedCoordinate>
  <gml:modifiedCoordinate>3</gml:modifiedCoordinate>
  <gml:coordOperation>
    <gml:Conversion gml:id="SphericalToGeographic">
      <gml:identifier codeSpace="TB18-D025">SphericalToGeographic</gml:
identifier>
      <gml:name>Earth Fixed (ECEF) spherical to geographic</gml:name>
      <gml:scope>TestBed 18 demonstration.</gml:scope>
      <gml:method>
        <gml:OperationMethod gml:id="GeographicSphericalConversion">
          <gml:identifier codeSpace="TB18-D025">GeographicSphericalConversion</
gml:identifier>
          <gml:name>Geographic/spherical conversions</gml:name>
          <gml:formula>See TB18 D025.</gml:formula>
          <gml:sourceDimensions>3</gml:sourceDimensions>
          <gml:targetDimensions>3</gml:targetDimensions>
        </gml:OperationMethod>
      </gml:method>
    </gml:Conversion>
  </gml:coordOperation>
</gml:PassThroughOperation>
```

Figure 18 — Pass-through operation (GML)

5.7. Spacecraft trajectory

The position of a spacecraft may be too complex for a description using simple formulas with a few parameters. Instead, providing the spacecraft trajectory in a file may be more convenient. This approach is similar to the use of datum shift grids (NADCON, NTv2, etc.) but with vector data instead of gridded data. In Testbed 18, the participants invented a new operation method named *"Trajectory to Conventional frame"* with a single parameter, the name of a file encoded in Moving Feature CSV format (OGC 14-084r2). The following constraints shall hold:

- operation source CRS = CRS attached to the moving feature (Voyager in our scenario); and
- operation target CRS = CRS of the Moving Feature CSV file (Heliocentric in our scenario).

Spacecraft yaw, roll, and pitch can be provided as attributes of the moving features. Using the spacecraft location relative to the Sun together with yaw/roll/pitch information, the coordinates of an object relative to Voyager can be transformed to heliocentric coordinates.

Below is an example of moving features CSV files. The first line declares the CRS, bounding boxes (in CRS units), start time, and end time. The second line declares the columns, with yaw/roll/pitch attributes as the last three columns (the corresponding values in this example are random). The `HeliocentricCRS` identifier can be used for looking up the complete CRS definition in the `VoyagerToObservatory.xml` file. CRS axes are Ecliptic latitude (degrees), Ecliptic longitude (degrees), and Distance from Sun (km). This example contains only one feature — the Voyager spacecraft — but an arbitrary number of features could be stored in the same file.

```
@stboundedby, urn:ogc:def:crs:TB18-D025::HeliocentricCRS, 3D, -37.524474
289.911611 19695916300, -37.524474 289.911611 19695917500, 2022-10-
31T18:52:00Z, 2022-10-31T19:00:00Z, absolute
@columns, mfidref, trajectory, yaw,xsd:decimal, pitch,xsd:decimal, roll,xsd:
decimal
Voyager, 2022-10-31T18:53:00Z, 2022-10-31T18:54:00Z, -37.524474 289.911611
19695916300, 10, 5, 20
Voyager, 2022-10-31T18:54:00Z, 2022-10-31T18:55:00Z, -37.524474 289.911611
19695916800, 10, 4, 21
Voyager, 2022-10-31T18:55:00Z, 2022-10-31T18:56:00Z, -37.524474 289.911611
19695917500, 10, 5, 20
```

Figure 19 — Spacecraft coordinates (Moving Features CSV)

The demonstration Java application (Clause 9.1) defines a *"Trajectory to Conventional frame"* operation method capable to read and use this CSV file. The coordinate operation is implemented by the `TrajectoryToConventionalFrame.java` file and added to the map projection library using the plugin mechanism described in Clause 8.9.

5.7.1. Gridded Geodetic data eXchange Format

Map projection libraries are faced with a variety of formats for handling datum shifts: NADCON, NTv2, text formats, *etc*. The Gridded Geodetic data eXchange Format (GGXF) is an ongoing OGC effort for defining a single format for all datum shifts. The proposal in the previous section, using Moving Feature CSV encoding, would force map projection libraries to know how to read that file format. If the GGXF format is used instead, then libraries would be able to use trajectory *almost* (see below) out of the box assuming that they already support that format for classical datum shift operations. The main characteristic of a GGXF file for trajectory would be:

- Source CRS = Voyager 3D CRS in our scenario;
- Target CRS = Heliocentric 3D CRS in our scenario;
- Interpolation CRS = The temporal 1D CRS; and
- Content type = Trajectory (a new content type to be defined).

The only extension to theexisting standard draft would be to define the `trajectory` content type. Some existing content types that may be used as a source of inspiration are `Cartesian3dOffsets` and `velocityGrid`. Note that this is not a proposal to turn GGXF into a new Moving Features format, but only to use it as a "Moving Reference Frame" format.

The use of GGXF would add one more constraint compared to Moving Feature CSV encoding: data must be gridded in the dimension of the interpolation CRS. In our scenario, it means that

data must be provided at a constant time interval. However, this constraint is highly desirable anyway for performance reasons: it makes it possible to very quickly find the data to use for a given time coordinate. Without that constraint, map projection libraries would have to use a temporal index, which may be prohibitively expensive if a search using that index must be performed for every coordinate tuple to transform.

5.7.2. Relativist moving feature

In the context of general relativity, gravitational wells (planets, *etc.*) distort spacetime into a rolling countryside of hills and valleys. GGXF should be able to model this gravitational terrain in the same way that GGXF can model a terrestrial geoid. This terrain changes over time (because planets are moving), but those changes can be modeled with the addition of a temporal axis.

Clause 5.7.1 proposed a non-relativist GGXF file where the dimension of the interpolation CRS is completely independent from the dimensions of source and target CRS. For using a GGXF file in the context of Einstein's relativity, all three CRSs (source, target, and interpolation) would need to be four-dimensional CRSs. A new GGXF content type would need to be defined, for example `relativistTransformation`. But there is no technical difficulty known at this time. This type would be the same process as classical NADCON/NTv2 datum shifts, only in 4 dimensions instead of 2. The GGXF binary file format can efficiently handle that number of dimensions since it is based on HDF5. There is possibly a performance issue for *reverse* operation because it may require an iterative algorithm, but the algorithm described in Clause 7.3 can mitigate that cost.

In addition to general relativity, transformations between the moving spacecraft and an observer are also subject to Lorentz contractions caused by special relativity (Clause 5.8.1). Those effects can be stored in a GGXF file either as a separated content type or added to the same content type as the one for general relativity.

Metric tensors are fundamental objects of study in general relativity. They are properties of a reference frame, not necessarily inertial. Temporal (e.g., separation of the future and the past) and geometric (distance, volume, curvature, angle) notions in a given reference frame are derived from the metric tensors. Those tensors can be represented as 4×4 symmetric matrices, thus having 10 independent components. While not explicitly stated in the current draft GGXF specification, GGXF can support the storage of metric tensors at each node of a grid. However, a `metricTensor` content type would differ from the usual GGXF content types because it would be a property of a *Coordinate Reference System* (CRS) rather than a property of a *Coordinate Operation* between a pair of CRSs. Storing this information may require a slight amendment of GGXF attributes:

- when describing a transformation property, specify `sourceCRS`, `targetCRS` and `interpolationCRS` attributes (this is the current specification); and
- when describing a CRS property, specify only a `CRS` attribute (which would be a change compared to current specifications).

5.8. Relativist CRS

The proposals in all previous sections (except Clause 5.7.2) ignored Einstein's relativity. In all previous CRS definitions, the temporal dimension was independent from the spatial dimensions. Those dimensions were defined in separated `SingleCRS` components, then those components were assembled in a `CompoundCRS`. In the ISO 19111 model, `CompoundCRS` is not associated with a coordinate system (Figure 1). Only the CRS components (the spatial and temporal CRSs) are associated with a coordinate system. Consequently, there is no 4-dimensional coordinate system. Only a 3-dimensional `CartesianCS` (or other type) followed by a 1-dimensional `TemporalCRS`.

For objects moving at sufficient speeds relative to an observer, the above specified separation is not desirable. The rate of change in the three spatial dimensions (the velocity) has an impact on the rate of change in the fourth dimension (the frequency of clock ticks). For doing coordinate transformations in the context of Einstein's relativity, the two coordinate systems (3D + 1D) need to be replaced by a single coordinate system (4D). A convenient 4D coordinate system for relativist calculations is the Minkowski space. However, there is no "Minkowski coordinate system" in the current ISO 19111. Adding that coordinate system would be an extension, which is proposed in Clause 8.5.

Given two coordinate reference systems named "A" and "B," if A and B are in different gravity fields and/or are moving fast relative to each other, then (*x*, *y*, *z*, *t*) coordinate tuples can be transformed from A to B in two different ways:

By formula: If the "gravity terrain" is simple enough, an application could do the mathematics by itself. A and B would need to be `SingleCRS` instances associated to a Minkowski coordinate system. A new `OperationMethod` instance would need to be defined in a way similar to Clause 5.3.1, then used in the definition of a coordinate operation similar to Clause 5.6.

By gridded data: If the "gravity terrain" is too complex, then some data producer would need to run a model and save the output in a "datum shift grid." This is similar to NADCON grids or NTv2, but in 4 dimensions instead of 2. The Gridded Geodetic data eXchange Format (GGXF) could be used (Clause 5.7.2).

5.8.1. Lorentz contractions

If a reference system is traveling at a sufficient velocity relative to an observer, distance along the path of travel appears to contract and time appears to dilate. Those changes are functions of the relative velocity between the observer and the observed. Because they are functions of *relative* velocity, Lorentz contractions are considered in this ER as properties of the coordinate *transformation* between a pair of CRS, not a property of any particular CRS (contrarily to metric tensors discussed in Clause 5.7.2). There can be different Lorentz contractions and time dilatations for each different pair of reference frames.

Consider an inertial reference frame named *A* in which a feature of interest is stationary. This may be a CRS attached to the feature (like the `EngineeringCRS` in Clause 5.2), but this does not necessarily need to be the case. Any inertial reference frame where the feature appears

stationary is suitable. Lengths measured on this feature are called **proper lengths**, and time measured by a clock attached to this feature is called **proper time**.

Consider a reference frame named *B* moving at a constant velocity $\vec{v}$ relative to reference frame *A*. Let's also assume that both frames have their axes oriented in the same direction, and that the velocity vector $\vec{v}$ is in the direction of *x* axis. Then consider the following quantities:

- Δt_0 is a *proper time* measured in a clock stationary relative to frame *A*;
- Δx_0 is a *proper length* measured in frame *A* in the same direction as $\vec{v}$; and
- Δy_0 and Δz_0 are lengths measured in directions perpendicular to $\vec{v}$.

Then the length and time measurements of the same feature, but measured in reference frame *B* instead of *A*, become:

- $\Delta t = \Delta t_0 \cdot \gamma$;
- $\Delta x = \Delta x_0 / \gamma$; and
- $\Delta y = \Delta y_0$ and $\Delta z = \Delta z_0$ (lengths in directions perpendicular to the motion are unchanged)

where γ is the Lorentz factor, which depends only on the relative velocity $\vec{v}$ (the speed of light *c* is a constant):

$$\gamma(v) = 1 \Big/ \sqrt{1 - (v/c)^2} \tag{1}$$

This relationship can be represented in matrix form as shown below. This matrix assumes that velocity $\vec{v}$ is in the direction of *x* axis. If the velocity was instead in the direction of *y* axis (for example), then the diagonal would contain the 1/γ term on the second row instead of the first row. If the velocity was toward an arbitrary direction (not a single axis), then the matrix would contain a mix of terms similar to a rotation matrix.

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t \end{bmatrix} = \begin{bmatrix} 1/\gamma & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \gamma \end{bmatrix} \times \begin{bmatrix} \Delta x_0 \\ \Delta y_0 \\ \Delta z_0 \\ \Delta t_0 \end{bmatrix} \tag{2}$$

The matrix form is convenient because it can express Lorentz contractions between two reference frames having a relative velocity $\vec{v}$ in an arbitrary direction. The matrix shown in the above equation can be interpreted as a Jacobian matrix (Clause 7). In the context of special relativity, this matrix does not depend on location in space and time. However, in the context of general relativity where there is acceleration (non-constant $\vec{v}$ vector) and/or gravity fields, the matrix would have different values for each spatiotemporal position.

The above matrix form fits nicely with the ISO 19111 conceptual model, which works on coordinate tuples such as (*x*, *y*, *z*, *t*). This matrix form is closely related to an affine transform, which is a common operation in map projection libraries. EPSG defines an operation method (2D only) named *"Affine parametric transformation"* while OGC 01-009 defines an operation method (in any number of dimensions) named *"Affine"*. An affine transformation can be represented with

an equation like the one above but augmented with an additional row, an additional column for translation terms, and with the Δ symbol removed.

$$\begin{bmatrix} x \\ y \\ z \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\gamma & 0 & 0 & 0 & T_x \\ 0 & 1 & 0 & 0 & T_y \\ 0 & 0 & 1 & 0 & T_z \\ 0 & 0 & 0 & \gamma & T_t \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ t_0 \\ 1 \end{bmatrix} \tag{3}$$

Remarks about the Jacobian matrix apply also to this affine transform matrix: this matrix assumes special (not general) relativity with a velocity $\vec{v}$ in the direction of *x* axis. For velocity in an arbitrary direction, the affine transformation is still applicable but looks more like a rotation matrix. In general relativity, the transformation is not affine anymore and the above matrix form does not apply (except locally). For the latter case, GGXF (Clause 5.7.2) may be an alternative.

5.8.2. Feature property values

A feature may contain properties such as lengths, surfaces, volumes, or angles measured on the feature. For example, a “Bridge” feature may contain a property named “length of the bridge” with a value in meters. Clarifying in which CRS such geometric properties are measured is important.

A common practice is to group features in collections having common characteristics. For example, each feature may be a row in a database table, and a common CRS may be applied to all features in the table. However, if each feature has a different velocity relative to the common CRS, then Lorentz contractions (Clause 5.8.1) will change the lengths differently for each feature. There are two opposite ways to handle this problem.

- Adopt a convention saying that all length properties in feature instances shall be *proper lengths*. This is equivalent to defining an implicit local CRS for each feature instance.
- Or conversely, adopt a convention saying that length properties in all feature instances are relative to the reference frame of the common CRS shared by all features in the collection.

Note that the same problem exists with map projections in classical geodesy. For example, the Mercator projection exaggerates the lengths of objects close to a pole. Given a collection of features located at different geographic latitudes, the Mercator projection will inflate the lengths differently for each feature. Should feature properties be real lengths or lengths measured in the Mercator projection? Both approaches have advantages and disadvantages. “Real length” or “proper length” are better descriptions of the features but having lengths in units of the common CRS is more consistent with the coordinates of feature geometries (because those coordinates are usually expressed in the common CRS). The latter convention may also prevent the need to transform property values when doing calculations between two or more features, which usually requires that the features to be in a common CRS.

In the scenario involving Mercator deformations, storing real lengths instead of lengths in a common CRS may be only a matter of preference since transformations need only the geographic latitudes. Latitude is an information contained (directly or indirectly) in coordinate tuples. But in the scenario involving Lorentz contractions, storing proper lengths introduces a

new difficulty: transformations to the common CRS require the feature velocity, which is not information contained in the coordinate tuples given to transform methods. Consequently, transforming proper lengths would require one of the following:

- six-dimensional coordinate systems with (*x*, *y*, *z*, *Vx*, *Vy*, *Vz*) axes where *V* is the velocity of the feature (Clause 8.6.3); or
- a new API for specifying the velocity together with the position in transform methods (Clause 8.6.2). For example, instead of `transform(DirectPosition)` something like `transform(DirectPosition, Velocity)` may instead be required.

Storing proper lengths is equivalent to defining an implicit CRS with each feature. If an application needs a common CRS, for example for rendering all features on the same map, then it may be more convenient to store lengths in units of that common CRS. Regardless, both conventions are practically equivalent if feature velocities *relative to the common CRS* are low enough for ignoring relativist effects. This does not mean that relativist effects are completely ignored. They are still taken into account during transforms between two CRSs (Clause 8.6.1).

5.9. Spacecraft deformations

The OGC Temporal Domain Working Group (DWG) has begun development of a discussion paper on dynamic features. Dynamic features are moving features that can change location, shape, and state as a function of time. A complete description of the spacecraft shape and state is out of scope for this Engineering Report. However, the consequence of those changes on coordinate values is considered. For example, if an `EngineeringCRS` is attached to the spacecraft (Clause 5.2) and if all measurements on the spacecraft surface are relative to an anchor point, then the coordinates of another fixed point may change with time because, for example, of thermal dilatation. In this scenario, the change in coordinate values is not caused by an “active” displacement of a point. The location is still the same point on the spacecraft surface. The coordinate change is rather caused by a deformation of the support. In ISO 19111, this is called “point motion operation” and is a third kind of operation in addition to conversion and transformation (Clause 5.6). A reference frame subject to point motions is called a **dynamic reference frame**. By contrast, a reference frame which is not dynamic is called a **static reference frame**.

The difference between a dynamic reference frame and a dynamic feature is that the deformations of a dynamic feature can be expressed in a static reference frame, which is external to the feature. For example, the changing shape of a hurricane may be described in a static geographic CRS. A reference frame can become dynamic when it is attached to the shape of the dynamic feature.

Point motion operations are supported by the GGXF format (Clause 5.7.1). While it was designed primarily for tectonic plate movements, handling thermal dilatations of spacecraft should be possible as a problem similar to tectonic plate movements on Earth.

6

GEOPOSE DRAFT STANDARD

6 GEOPOSE DRAFT STANDARD

The draft OGC GeoPose standard defines requirements for the interoperable exchange of the location and orientation of real or virtual geometric objects (poses) within reference frames anchored to the Earth's surface or to other bodies. A pose is a representation of a transform mapping the space of an **outer reference frame** to the space of an **inner reference frame**. The outer frame can be referenced to the Earth, another astronomical body, or a spacecraft (non-exclusive list). The inner frame can be referenced to (for example) a spacecraft orbiting around the outer frame. Then the GeoPose standard specifies the packaging of sequenced (or linked) frame transforms. This is done with a **frame graph**, which is a directed acyclic graph representation of the transformational relationships between reference frames.

There may be zero, one, or many paths between two distinct frames. Coordinate transformations between the same frames but following different paths may not agree. A transformation (pose) may be associated with additional non-geometrical properties such as time of observation or validity. In GeoPose version 1.0, coordinate transformations can only involve translations and rotations.

The characteristics of a frame transform in GeoPose are very similar to an ISO 19111 Coordinate *Transformation* (Clause 5.6). Both are defined by parameters. Both have a source and target associated to different frames, while ISO 19111 makes no distinction between inner and outer frames. In both cases, many transformation paths may exist and produce slightly different results, while in ISO 19111 there is an expectation that the differences are within the declared operation accuracy. Both have a domain of validity, which in ISO 19111 may be spatial and/or temporal. Both can express a chain of transforms, which is called **concatenated transform** in ISO 19111 (Clause 5.6.2). While GeoPose defines classes such as `Chain`, `Graph`, `Quaternion`, `TangentPointPosition`, `YawPitchRollAngles`, *etc.*, many of them can be retrofitted into existing ISO 19111 constructs and encoded with the current GML schema.

The following sub-sections propose ways to retrofit GeoPose objects into the ISO 19111 model. In some cases, this implies relaxing GeoPose restrictions. The WKT and GML encoding examples given in Clause 5 can apply here. The GeoPose encoding in JSON can be more compact than ISO 19111 encoding in GML or WKT (not necessarily because of JSON versus XML syntax) but is not yet (as of November 2022) as flexible.

6.1. Outer frame

GeoPose uses an outer frame as the starting point of a frame graph. In GeoPose 1.0, that outer frame is fixed to a three-dimensional "World Geodetic System 1984" CRS with latitude, longitude, and ellipsoidal height axes (EPSG:4979). In the ISO 19111 model, there is no distinction between outer and inner frames. Any frame can be associated to the source CRS of a coordinate operation. Consequently, the restriction to EPSG:4979 does not apply.

GeoPose allows the definition of a Local Tangent Plane (LTP), which is tangential to the geoid or ellipsoid at a given latitude and longitude. This plane can be oriented in different ways such as

(yaw, pitch, roll) angles or by quaternion, and the orientation can be time dependent. If defined, this tangent plane is an inner frame derived from the outer frame in the frame graph. However, as previously stated, in ISO 19111 there is no concept of inner and outer frames. Instead, the important criterion is whether the "inner frame" is referenced to the same object than the "outer frame". For example, if the "inner frame" is a land area (or maybe a non-moving platform fixed to Earth), then the coordinate system is changed for convenience but the CRS is still associated (even if indirectly) to the same datum, for example WGS84. In such case there is a derived CRS as defined in Clause 5.3. Otherwise there is a coordinate transformation as defined later in this section.

If a local tangent plane is defined by a `DerivedCRS`, then there does not need to be an explicit part of the frame graph. Since ISO 19111 has no restriction about the source CRS of a coordinate operation, a `DerivedCRS` can be the "outer frame" of a frame graph. Derived CRSs and their relationship with their base CRSs have well-defined encoding in WKT (Clause 5.3) and GML.

Another way to decide whether to use a `DerivedCRS` is to look how its relationship with the base CRS was defined. As a rule of thumb (not an absolute criterion), if the relationship has been determined by observations, is subject to stochastic errors, and can be revised at any time, then there probably is a coordinate transformation. Conversely if the relationship has been established *by definition* (potentially after observation, before an authority decided to fix some parameter values as the official ones), then there probably is a coordinate conversion. In the former case (transformation), `DerivedCRS` cannot be used. In the latter case (conversion), a `DerivedCRS` can be used, but it still optional. In any case, the distinction between conversion and transformation carries semantic information which is absent from GeoPose.

6.2. Inner frame

An inner frame is defined by the coordinate operation from another (possibly outer) frame to the inner frame. This is similar to a ISO 19111 derived CRS, except that the coordinate operation can be a transformation. GeoPose defines various classes (`YawPitchRollAngles`, `AtAndUpVector`, *etc.*) for storing the parameters required for the definition of coordinate operations. Each class is specialized for some specific parameters, for example rotation angles. This is in contrast with ISO 19111, which has an approach more similar to ISO 19109 `FeatureType`. In ISO 19109, a `FeatureType` is a *meta-class*: a class used for defining other classes. This is similar to reflection in the Java programming language, or to the elements used for defining a schema in XML or JSON. In ISO 19111, the `OperationParameter` and `OperationParameterGroup` types (Figure 7) act like meta-classes: they define the set of expected parameters for a given coordinate operation, much as a JSON schema defines the set of properties for a given JSON element. When applied to strongly typed programming languages such as Java, meta-classes are flexible because they do not require the definition of new classes in the programming language for each new kind of operation. Instead, new operations are described by instances of ISO 19111 `OperationMethod` and `OperationParameter` types, which in turn can be materialized by a few rows in the EPSG database. Those definitions need to be provided only once and can be referenced by `xlink` in a GML document or by authority code in a WKT definition.

GeoPose defines many sets of parameters, from basic to more advanced use cases. But they are all different ways to specify translations and rotations. This is similar to EPSG providing different ways for defining the same map projection. For example, EPSG provides three different ways to define a Mercator projection:

- **Mercator (variant A)** which accepts a *Scale factor at natural origin* parameter (among others);
- **Mercator (variant B)** which accepts a *Latitude of 1st standard parallel* parameter (among others); and
- **Mercator (variant C)** which accepts *Latitude of 1st standard parallel* and *Latitude of false origin* parameters (among others).

All those variants are only different ways to process the parameters. After parameter processing, those variants can be implemented by the same code. The same remark applies to operations defined by GeoPose. This is demonstrated in Annex A.3.1, which lists GeoPose operations reworded as definitions using ISO 19111 objects. The following GeoPose operation methods are defined:

- Basic-YPR (yaw, pitch, roll) ⟶ "Pose rotation by YPR;" and
- Basic-Quaternion ⟶ "Pose location and orientation by quaternion."

The Advanced GeoPose method is excluded because it can be done by using a `DerivedCRS` as the outer frame (Clause 6.1), and then using the Basic-Quaternion operation method (Clause 5.6). The `GEODETICCRS["Local Tangent Plane - East North Up (LTP-ENU)"]` element in Clause 5.3 corresponds to a first inner frame in GeoPose. Then the `COORDINATEOPERATION["Pose to extremity of robotic arm"]` element in Clause 5.6 corresponds to a second inner frame in GeoPose.

6.3. Chain and frame graph

A chain is an outer frame and a sequence of transformations to a final innermost frame. A frame graph is a structured modeling of the pose relationship between frames (nodes) and transforms (edges) in a graph structure. Those two constructs can be modeled in ISO 19111 by `CoordinateOperation` and `ConcatenatedOperation`. See Clause 5.6.2 for an example.

6.4. Time-varying parameters

GeoPose provides two ways to define a coordinate operation with time-dependent parameters. One way is termed "regular" and is built with a sequence of frames with a constant inter-pose duration. The second way is termed "irregular" and supports variable inter-pose durations.

In both cases, there is a block of parameters which is repeated with different values for each GeoPose frame. In the ISO 19111 model, a repeating set of parameters can be represented by a `OperationParameterGroup` (Figure 7). A group can contain for example the "valid time", "translation," and "rotation" parameters. Parameter groups derived from the GeoPose standard are given in Annex A.2. Then a coordinate operation can be created with a set of parameters containing a `OperationParameterGroup` instance for each valid time. Clause 5.6.1.2 shows an example in GML (the parameter group construct is not available in WKT) using the *"Pose irregular series"* transformation. A complete GML document is provided by the `PoseToRoboticArm.xml` file in the Testbed 18 GitHub repository.

The following are two differences in the ISO 19111 model compared to the GeoPose model.

- Translation and rotation are not parts of the frame definition. They are parts of the *coordinate operation* between a pair of frames.
- The set of parameters for all valid times are grouped in a single operation. The above example does not define a new frame for each valid time.

Note that all examples shown in this document up to this point use existing ISO 19111 and GML standards. No extension to existing standards has been required yet.

6.5. GeoPose encoding of CRS

GeoPose encodes coordinate reference systems (CRS) in JSON documents with a small block containing an authority, an identifier, and a list of parameters. This block and its properties are specific to the GeoPose draft standard. An issue with the current GeoPose approach is that it is implementation dependent. The authority property can be interpreted as the language in which parameters are encoded. If the authority is PROJ, then the parameters are a PROJ string and only PROJ (or other software having some compatibility layer) can understand those parameters. The same issue exists with EPSG codes as well. If a software does not have a connection or a copy of EPSG geodetic dataset, it cannot understand the CRS.

The GeoPose structure could be replaced by a ISO 19111 structure encoded in JSON. There is no OGC standard yet for JSON encoding of ISO 19111, but a proposal based on PROJ-JSON is on the table. The ISO 19111 structure contains all information currently found in GeoPose structure, plus an implementation neutral description of the CRS. For example, no matter the authority, all CRSs have a set of axes. The ISO 19111 structures enumerate those axes together with other information such as datum. Those structures do not necessarily contain all metadata provided by the authority, but they contain sufficient information for allowing coordinate operations even if the software does not know the authority. CRS encoding derived from ISO 19111 such as GML, WKT, or PROJ-JSON make possible to:

- use the authority code if the software understands the authority;
- or otherwise fallback on information embedded in the CRS definition.

7

JACOBIAN MATRIX

7 JACOBIAN MATRIX

The legacy OGC 01-009 *Coordinate Transformation Services* specification defined a `derivative` method as a member of the coordinate operation interface (that interface was named differently in OGC 01-009). That method receives a position in input argument and returns a matrix in output. That method is described in 01-009 clause §12.4.5.6 as:

> The derivative of this transform at a point. (...snip...). The derivative is the matrix of the non-translating portion of the approximate affine map at the point. The matrix will have dimensions corresponding to the source and target coordinate systems. If the input dimension is *M*, and the output dimension is *N*, then the matrix will have size [M][N]. The elements of the matrix {elt[n][m] : n=0..(N-1)} form a vector in the output space which is parallel to the displacement caused by a small change in the m'th ordinate in the input space.
>
> — OGC, OGC 01-009, Section 12.4.5.6

While the OGC 01-009 document does not contain the word "Jacobian," the above definition matches a Jacobian matrix. In the particular case of a two-dimensional map projection *P* where input coordinates are (φ, λ) and output coordinates are (*x*, *y*), the Jacobian matrix contains the partial derivatives of P(φ,λ):

$$J_P(\phi,\lambda) = \begin{bmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \lambda} \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \lambda} \end{bmatrix} \tag{4}$$

For example if *P* is a spherical Mercator projection, then the matrix is:

$$J_P(\phi,\lambda) = \begin{bmatrix} 0 & R \\ \frac{R}{cos(\phi)} & 0 \end{bmatrix} \tag{5}$$

NOTE The above matrix would be diagonal if the input axis order was (*longitude*, *latitude*). The (*latitude*, *longitude*) axis order may look like a source of confusion in this context, but the systematic use of matrices in map projection libraries makes the axis order issue largely irrelevant. Switching axes order causes some terms to appear at different locations in the matrix, but the math is the same. Map projection libraries can do matrix operations (multiplication, inversion) without any concern about axis order.

In the figure below, the first column in the Jacobian matrix gives the $\vec{V}$ vector and the second column gives the $\vec{U}$ vector (in that order because we used latitude, longitude axis order as inputs). The matrix values, and consequently the $\vec{U}$ and $\vec{V}$ vectors, vary at each point. The figure illustrates the vectors of the Jacobian matrix evaluated at two locations, P_1 and P_2.

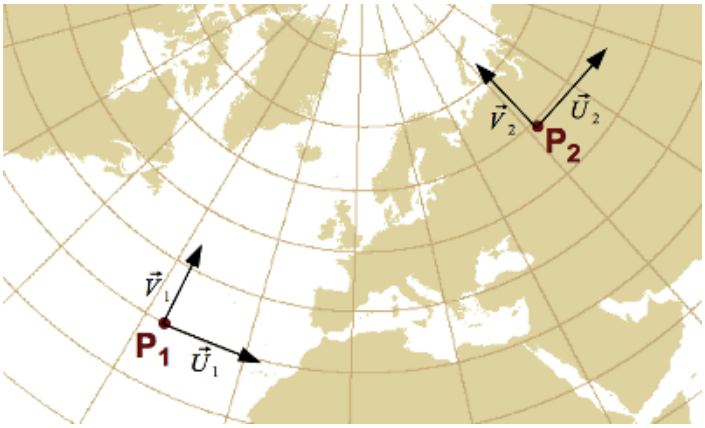

Figure 20 — Derivatives of a map projection

The above example was for a map projection, but more complex chains of operation can be built. The Jacobian matrix of the whole chain is the product of the Jacobian matrices of each step. Another useful property is that for obtaining the Jacobian of the reverse projection, the equations do not need to be derived. The Jacobian of the forward projection can be computed, then the matrix inverted:

$$J_{P^{-1}} = J_P^{-1} \tag{6}$$

While it is true for any function, this property is especially useful for map projections because the reverse projection is often much more complicated than the forward projection. This has the amazing consequence that an exact Jacobian matrix can exist even for projections having no exact formulas, providing that exact formulas exist in the opposite direction.

7.1. Getting matrix values

If the coordinate operation is an affine transform, then the Jacobian matrix is the same everywhere. But for any non-linear operation, the Jacobian matrix can be different at every spatiotemporal location. Consequently, Jacobian matrices must be computed in a way similar to the computations done for transforming positions. ISO 19111 currently defines the following method on `CoordinateOperation`:

```
CoordinateSet transform(CoordinateSet positions);
```

In addition to above method, the following method would also be needed. This method already exists in GeoAPI `MathTransform` interface, inherited from OGC 01-009:

```
Matrix derivative(DirectPosition position)
```

Those two methods exist for ISO 19111 and OGC 01-009, respectively. A possible extension to those APIs is presented in Clause 8.6.2.

7.2. Example: transforming a BBOX

Some numerical problems, such as the search for minimum value, are solved by iterative computations of a function. Often, the search converges faster when the function derivative is known (e.g., *Gradient descent* algorithm). This is also true for the transformation of a bounding box from one CRS to another CRS. The algorithm needs to find the minimum and maximum values of the transformed shape. If the transformation is non-linear, then those minimums and maximums are not necessarily the corner coordinates. For example, the blue shape in the figure below is the result of transforming a bounding box. As one can see, the lowest part of the blue shape is lower than all corners. The transform algorithm needs to find the minimal *y* coordinate, which appears somewhere between the corners. A brute force approach would be to sample a large number of points between the corners and retain the smallest value, with the hope that the sampling is dense enough for including a point close to the real minimum. But a more efficient approach is to use the derivatives at the two lower corners for approximating a cubic polynomial ($y = A + B{\cdot}\lambda + C{\cdot}\lambda^2 + D{\cdot}\lambda^3$), illustrated by a red curve in the figure below.

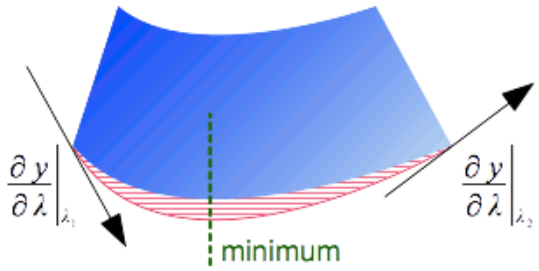

Figure 21 — Use of derivatives for BBOX transformations

The cubic polynomial can easily be solved for finding the λ coordinate of the minimal *y* value according that approximation. That minimum computed form cubic approximation is usually located close to the real minimum of the transformed shape. If desired the Jacobian matrix can be evaluated recursively at the new λ coordinates for getting even closer. Then the real *y* value can be computed at the λ coordinate of the minimum.

The performance advantage of using Jacobian matrices for BBOX transformations increases quickly with the number of dimensions. A brute force approach sampling 100 points on each side would need to sample 4×100 points for a two-dimensional rectangle, 100×100×6 points for a three-dimensional cube, and millions of points for a four-dimensional hypercube (3D + time). In practice, the number of points can be greatly reduced when considering some dimensions as independent (e.g., time), or if the transformation is linear in at least some dimensions. But in a relativist system the 4 dimensions are not independent and the transformations are not always linear.

7.3. Example: datum shift

Some transformations are too complex for expression by a mathematical formula. Instead, a grid is provided with displacement vectors from coordinates in the source CRS to coordinates in the target CRS. This grid can be encoded in a GGXF file (Clause 5.7.1), for example.

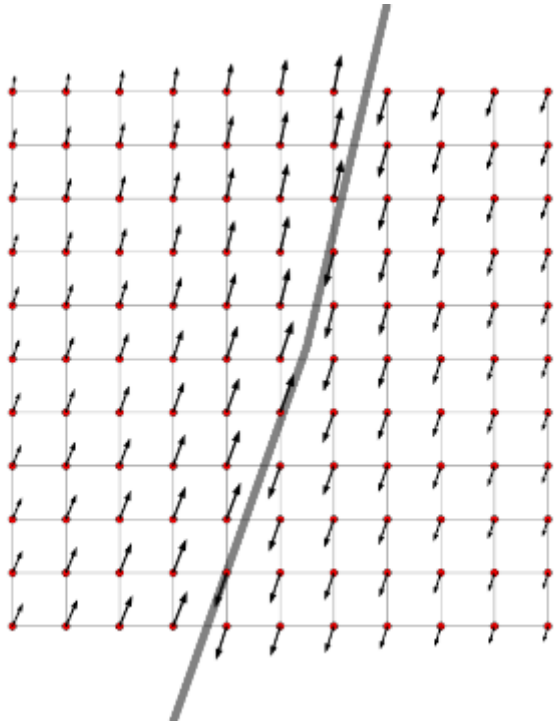

Figure 22 — Displacement vectors (source: GGXF group)

Transforming coordinates from a source CRS to a target CRS is a straightforward process as follows.

1. Get the displacement vectors at source coordinates.
2. Compute *target coordinates* = *source coordinates* + *displacement vector*.

However, the reverse operation (i.e., from target to source) is much more complicated. A displacement vector cannot be directly used for given target coordinates because the domain of the grid is in the *source* CRS, not the target CRS. The coordinates in source CRS (i.e., the location where to fetch a displacement vector) are not known since they are precisely the values we want to compute. For solving this problem, NADCON (North American Datum Conversion) uses an iterative approach based on the assumption that source coordinates are approximately equal to target coordinates, with only small displacement vectors between them. This assumption allows the following algorithm (using vector arithmetic).

1. Use target coordinates as initial *interim source coordinates*.
2. Get the displacement vector at interim source coordinates.

3. Compute *interim source coordinates = target coordinates − displacement vector*.

4. Repeat from step 2 until the change between two iterations is smaller than the desired accuracy.

However, the above algorithm does not converge if there are relatively large variations in vector magnitudes and directions. The interim source coordinates may move chaotically around the true source coordinates and can even go further and further at each iteration step. A more stable (but still not perfect) algorithm is as shown below.

1. Use target coordinates as initial *interim source coordinates*.

2. Get the displacement vector at interim source coordinates.

3. Compute *interim source coordinates = target coordinates − displacement vector*.

4. Get new displacement vector at new interim source coordinates.

5. Compute *error in target* = (*new interim source coordinates + new displacement vector*) − *target coordinates*.

6. Compute *error in source* = J^{-1} × *error in target* where *J* is defined below.

7. Translate the interim source coordinates by a vector in the opposite direction of *error in source*.

8. Repeat from step 3 until the change between two iterations is smaller than the desired accuracy.

In a nutshell, compute at each iteration the error between the estimated position and the desired position. Then correct the position by moving in the direction opposite to the error. The problem is that the error is measured in *target coordinates* while we need to correct the *source coordinates*. Consequently, the *error in target* vector needs to be converted to an *error in source* vector. If the Jacobian matrix at the position of interim source coordinates is known, then the conversion can be done as below where (φ,λ) are source coordinates, (*x*, *y*) are target coordinates and Δ are differences between estimated and actual values:

$$\begin{bmatrix} \Delta\phi \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \phi} & \frac{\partial x}{\partial \lambda} \\ \frac{\partial y}{\partial \phi} & \frac{\partial y}{\partial \lambda} \end{bmatrix}^{-1} \times \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{7}$$

Note that if target coordinates are approximately equal to source coordinates, then the Jacobian matrix is close to an identity matrix. Consequently $(\Delta\varphi, \Delta\lambda) \approx (\Delta x, \Delta y)$ and step 7 in above algorithm simplifies as shown below.

- New interim source coordinates defined in step 7: $(\varphi,\lambda) - (\Delta\varphi, \Delta\lambda) \approx (\varphi,\lambda) - (\Delta x, \Delta y)$.
- From step 5: $(\Delta x, \Delta y) = (\varphi,\lambda) + (Dx, Dy) - (x, y)$ where *D* is the displacement vector.
- After substitution and simplification, *interim source coordinates* = $(x, y) - (Dx, Dy)$.

The result of the above simplification is exactly step 3 in the NADCON algorithm. From the point of view of the algorithm using Jacobian matrix, datum shift algorithms such as NADCON have an implicit assumption that the Jacobian matrix is close to identity everywhere. This is true for most geodetic datum shifts, but it is not true for transformations based on grids having stronger curvatures or deformations.

In general relativity, gravity fields change the spatial and temporal coordinates all together. The gravity field may be too complex to be described by a few equations, as seen for example with Earth geoids. Given two reference frames in different gravity fields, transformations from one frame to the other frame could be described empirically by a grid similar to datum shift grids used in geodesy (Clause 5.7.2). If curvature is strong, classical algorithms used in geodesy may not converge. Jacobian matrices can resolve the convergence problem in some cases (not always, it also has its limits). Even when convergences are possible without them, as seen in previous section, Jacobian matrices can speed calculations by reducing the number of iterations, especially in spaces of more than two dimensions.

However, computing the Jacobian matrix has a cost. In a datum shift grid, it requires computing finite differences with neighbor nodes and the results are not exact representations of the derivatives at the desired location. Better accuracy, and possibly better performance, can be achieved if the Jacobian matrices are computed by the data producer and stored in the grid. The Gridded Geodetic data eXchange Format (GGXF), in development at OGC, is technically capable to store matrices.

7.4. Lorentz transformations

In the context of special relativity, Lorentz transformations can be represented by a Jacobian matrix as discussed in Clause 5.8.1. In the context of general relativity, the Jacobian matrix is valid only locally and may change at every spatiotemporal position. But this is similar to the map projection case discussed in Clause 7, and the same API as Clause 7.1 (i.e., the `derivative(…)` method) can apply if the following conditions hold.

- The relative velocity between source CRS and target CRS is known (it may be a parameter of the coordinate operation).
- Coordinates and property values to transform are in units of the source CRS, not proper lengths (Clause 5.8.2), so the velocity of individual feature instances can be ignored.

The Jacobian matrix obtained under those conditions allows transformation of lengths observed in source CRSs to lengths observed in target CRSs. None of those lengths need to be proper lengths. If transformation from/to proper lengths is desired, then the transformation methods need an additional API discussed in Clause 8.6.2.

8

PROPOSED EXTENSIONS TO STANDARDS

8 PROPOSED EXTENSIONS TO STANDARDS

This section proposes backward-compatible extensions to existing standards. The problem of GML verbosity is not described in this section, because fixing that issue would probably break backward compatibility (see Annex B for more information). A summary of proposed extensions is in Clause 8.11.

8.1. Celestial body information

Technically, geodetic frames could be associated to any celestial body (if we ignore the etymology of "Geo" prefix in class names — see Clause 8.3). In practice, there is often an implicit assumption that geodetic frames are associated to the Earth. There is currently no way in the ISO 19111 standard to unambiguously identify the celestial body of a reference frame. A process could be to parse the reference frame name or the anchor, but such an approach is fragile because those attributes are free text strings. A more systematic approach would be to introduce a new class, for example named `CelestialBody` and add an association from `GeodeticReferenceFrame` to `CelestialBody`. Expressed in WKT format, this proposal would be the element in bold characters below:

```
GEODCRS["Mars (2015) / planetographic",
  DATUM["Mars (2015)",
    CELESTIALBODY["Mars"],
    ELLIPSOID["Mars", 3396190, 169.8944472236, LENGTHUNIT["metre", 1]],
    ANCHOR["Viking 1 lander : 47.95137 W"],
    PRIMEM["Airy", 0, ANGLEUNIT["degree", 0.01745329251994329]]],
  CS[ellipsoidal, 2],
    AXIS["Latitude (B)", north, ORDER[1]],
    AXIS["Longitude (L)", east, ORDER[2]],
    ANGLEUNIT["degree", 0.01745329251994329],
  SCOPE["For horizontal positioning on Mars."]]
```

Figure 23 — Celestial body proposal in a WKT definition

The PROJ 6+ library (Clause 9) already implements something similar. That library stores EPSG data in its own database schema, which is slightly different than the EPSG schema. The PROJ database schema contains a field for celestial body information.

In the ISO 19111 abstract model, the `CelestialBody` could be one of the following types.

- A subclass of ISO 19111 `IdentifiedObject` class. The advantage is to introduce no new dependency to another standard, which is inconvenient because the celestial body would be identified only by a single name, which may be insufficient.
- A subclass of ISO 19112 `Location` class (from *Referencing by identifiers*). The advantage is to enable the use of structured identifiers like addresses. For example, the celestial body could be identified by a "Galaxy → Star → Planet" hierarchy of names, however an inconvenience would be to introduce a dependency from ISO 19111 to ISO 19112.

Like ISO 19111, the terminology used in ISO 19112 assumes a location on Earth. The use of ISO 19112 depends on whether the use of GeoXXX objects is acceptable for bodies other than Earth. This issue is discussed in Clause 8.3.

8.1.1. Celestial body in geographic extent

The ISO 19112 Location class has an association to ISO 19115 GeographicExtent, which can describe a geographic area with a textual description, a polygon, or a bounding box. In practice, GeographicBoundingBox is the most frequently used specialization. However, geographic bounding boxes in ISO 19115 do not specify a CRS. This is because they are intended to describe only an approximated region. For that reason the latitudes and longitudes are specified with only two fraction digits. But even if the exact CRS is not specified, the celestial body needs to be specified. An association from GeographicBoundingBox to CelestialBody may be needed.

8.2. Non-geographic domain of validity

The BBOX element in the WKT format accepts only Earth latitudes and longitudes. Other ways to specify a domain of validity are needed, for example as a distance from an arbitrary celestial body. This is important for allowing map projection libraries to select the most appropriate coordinate operation for a pair of CRSs when more than one operation is available in the database (see Clause 5.6).

For simplicity reasons the latitude and longitude restriction is specific to the WKT format. The ISO 19111 abstract model does not use specifically the GeographicBoundingBox or GeographicExtent class. Instead, the model uses the more generic Extent class, which allows custom Extent subclasses. The steps for extending the domain of validity to spatial environments could be as follows.

- Create a new class as a specialization of ISO 19115 Extent.
- Create a new WKT keyword for representing the new extent class.

8.3. Use of geocentric term

ISO 19111 uses “geographic” and “geocentric” terms in places where it could apply to any celestial bodies. For example, the AxisDirection code list contains geocentricX, geocentricY, and geocentricZ values. There are at least three possible ways to extend ISO 19111 axis directions to planetary CRS.

- Define new code list values such as heliocentricX, heliocentricY, and heliocentricZ for other astronomical bodies.

- Accept the use of `geocentricX`, `geocentricY`, and `geocentricZ` for any astronomical bodies, despite the "geo" semantic.
- Do a compromise with a single set of new axis directions for a whole category of astronomical bodies other than Earth. For example `planetocentricX`, `planetocentricY` and `planetocentricZ`.

The first alternative would make implementations more complicated because it would hide the fact that those axes are used with the same formulas (such as spherical to Cartesian coordinate system conversions) regardless of the astronomical body. The second alternative is the opposite extreme. The third alternative is between the two "extremes".

The above discussion was only about one code list. But the "geo" prefix is spread across OGC/ISO standards. "Geo" appears in class names of ISO 19111 (`GeographicCRS`, `GeodeticCRS`, `GeodeticReferenceFrame`), of ISO 19115 (`GeographicExtent`, `GeographicBoundingBox`, `GeographicDescription`, `GeolocationInformation`, `Georectified`, `Georeferenceable`), and of ISO 19107 (`Geodesic`) to cite just a few. Earth-related prefixes also appear in attributes or associations (`geographicIdentifier`, `alternativeGeographicIdentifier`, `geographicExtent`, `geographicElement`, `geolocationInformation`, `geoidModel`, `greenwichLongitude`, *etc*). Duplicating all above-cited classes with different names, properties, and code list values would be significant work not only for the planetary working group, but also for the working groups of all impacted standards and for the implementers.

8.4. Inertial CRS type

Clause 5.4 defined an Earth-centered quasi-inertial CRS using the `EngineeringCRS` type. This is because no other ISO 19111 type would have been acceptable. However, `EngineeringCRS` is still not quite right because this CRS is intended for local uses or vessels. A new CRS type is desirable, with a structure (class, associations, and attributes) similar to `GeodeticCRS`. The new CRS type could be named `InertialCRS` and its datum class could be named `InertialDatum`. Whether `InertialDatum` should contain an ellipsoid (the existing `Ellipsoid` class could be reused) is an open question. An ellipsoid may not be needed if the CRS accepts only three-dimensional Cartesian and spherical coordinate systems and is not intended for referencing objects on planet surface.

Using this `InertialCRS` proposal, the definition shown in *Quasi-inertial CRS (WKT)* figure of Clause 5.4 would become as below. This proposal also uses the `CelestialBody` proposal presented in Clause 8.1, and keep an ellipsoid inside the datum for this version.

```
INERTIALCRS["Quasi-inertial (Equatorial CS)",
  INERTIALDATUM["ECI frame",
    CELESTIALBODY["Earth"],
    ELLIPSOID["WGS 84", 6378137, 298.257223563]],
  CS[Spherical, 3],
  AXIS["Declination (DEC)", north, ORDER[1], ANGLEUNIT["degree",
0.0174532925199433]],
  AXIS["Right Ascension (RA)", east, ORDER[2], ANGLEUNIT["degree",
0.0174532925199433]],
  AXIS["Radius (R)", up, ORDER[3], LENGTHUNIT["metre", 1]],
```

```
  REMARK["North stands for ""north celestial pole"" and East for ""6 hours
from vernal equinox""."]]
```

Figure 24 — Quasi-inertial CRS (proposed WKT extension)

8.5. Minkowski coordinate system

Each Coordinate Reference System (CRS) is associated with a Coordinate System (CS). The subtypes defined by ISO 19111 such as `CartesianCS` and `SphericalCS` can be at most three-dimensional. The `TemporalCS` component, if desired, must be added separately as a dimension independent from the spatial ones (Clause 5.5).

For coordinate operations involving special or general relativity, the mathematical formulas require that space and time are handled together. To represent this requirement in the ISO 19111 model, a new `CoordinateSystem` subtype is required. The proposal is to define `MinkowskiCS` class as a coordinate system with the following constraints.

- The number of dimensions must be exactly 4.
- All dimensions must have the same unit of measurement (UoM). Temporal coordinates shall be converted to length unit by multiplying by the speed of light.

For each type of coordinate system, there is a different set of formulas for computing geometric properties such as distance. For example, the distance between two points is computed using the Pythagorean formula if the coordinate system is Cartesian, or a different formula if the coordinate system is polar. Like any coordinate system, the use of a Minkowski coordinate system also implies a particular formula for computing interval between two spacetime events:

$(\Delta s)^2 = (\Delta ct)^2 - (\Delta x)^2 - (\Delta y)^2 - (\Delta z)^2$

For the purpose of this Engineering Report, the key information to retain is that the use of `MinkowskiCS` implies not only a 4-dimensional space, but also homogeneity in axis units and a set of mathematical formulas that apply in this space.

8.5.1. Association to a CRS

The next step is to associate `MinkowskiCS` to a CRS. A specialized CRS subtype (for example `MinkowskiCRS`) could be created, but this is not necessarily desirable. It may be desirable to be able to substitute `CartesianCS` by `MinkowskiCS` when relativistic effects are significant in any CRS. If allowing such substitution anywhere is considered too questionable, a compromise is to allow substitution only for `InertialCRS` and `EngineeringCRS` in the first version of a new standard. This association can easily be done for `InertialCRS` (Clause 8.4) if the two classes are defined in the same document, but is more problematic for `EngineeringCRS`.

ISO 19111 `EngineeringCRS` accepts only coordinate systems of type `EngineeringCS`, which is defined as the union of `AffineCS`, `CartesianCS`, `CylindricalCS`, `LinearCS`, `PolarCS`,

SphericalCS, and OrdinalCS classes. There are two ways to support an association with another coordinate system type:

- add MinkowskiCS to the list of types contained in EngineeringCS union; or
- completely remove the EngineeringCS union and allow EngineeringCRS to be associated to any CoordinateSystem type.

Both approaches require a modification of ISO 19111. But the first approach also requires either defining MinkowskiCS in ISO 19111, or introducing a dependency from ISO 19111 to another ISO standard where MinkowskiCS would be defined. Furthermore, that approach would fix only the case of MinkowskiCS. The process would have to be repeated every time that a new type of coordinate system needs to be associated to EngineeringCRS. By contrast the second approach avoids any explicit dependency from ISO 19111 to MinkowskiCS and allows the easy addition of any new coordinate system types in the future without a revision to the ISO 19111 standard. The inconvenience is that using an association with CoordinateSystem would accept all coordinate system types, including TemporalCS which was intentionally omitted from EngineeringCS union. But the TemporalCS exclusion could be documented as a constraint instead of being enforced with EngineeringCS union.

8.5.2. Why a specific type

An alternative to introducing a new MinkowskiCS type is to use a standard CompoundCRS made of a spatial component with a temporal component and then consider the substitution of such compound CRS by Minkowski spaces as an implementation detail. However, an explicit MinkowskiCS class would serve four purposes:

- allow constraints on the unit of measurement by stating that all dimensions, including the temporal one, must have the same unit;
- declare that coordinates in this space should be handled using the formulas valid in Minkowski space;
- declare that the four coordinates are inseparable and should be handled together during coordinate operations (Note: a separation is still possible by explicitly converting the coordinate tuples to another CRS. However, doing so implies that a different set of geometric properties and formulas is desired); and
- declare that the 4 coordinate values in a tuple use the same reference frame. (By contrast, a compound CRS implies the existence of 2 or more reference frames (e.g., a vertical frame distinct from the horizontal one)).

A similar approach exists in the current ISO 19111 standard. A three-dimensional CRS with any vertical measurement *except ellipsoidal height* is expressed by a compound CRS made of a spatial component with a vertical or parametric component. Each component has its own reference frame, which can be chosen almost independently from the other (provided that their domains of validity are compatible). For example, various vertical frames such as Mean Sea Level (MSL), Lowest Astronomical Tide (LAT), gravity-related height, barometric pressure, *etc.* can be used with various horizontal frames. The sole exception to this pattern is the ellipsoidal height. The

ISO 19111 standard prohibits the construction of vertical CRS for ellipsoidal height, which in turn disallows the construction of a compound CRS with an ellipsoidal height component. Any use of ellipsoidal height must be done through a **single CRS** associated to a three-dimensional ellipsoidal (or Cartesian in map projection case) coordinate system. This design introduces an apparent complication for handling vertical measurements, because libraries have to implement at least one special case. However, this approach has three justifications as follows.

- An ellipsoidal height coordinate is meaningless if not accompanied by the horizontal coordinates, contrarily to the other kinds of height which still have physical meanings.
- Ellipsoidal heights share the same reference frame than the horizontal components, contrarily to other kinds of height which have their own reference frame.
- Transformation from a 3D CRS to another 3D CRS may require transformation all kinds of heights to ellipsoidal heights before transforming from one geodetic frame to another, and that latter transformation should operate on the three coordinates together because the ellipsoidal height has an impact on the result of transforming the horizontal components.

The last point in the above list is a frequent cause of error. In practice, when users build a chain of coordinate operations, what frequently happens is that the *z* value is either lost or not transmitted to the step performing a transformation between two geodetic reference frames. A height of zero is then assumed, which results in different horizontal coordinates than would have been calculated with a non-zero ellipsoidal height. Sometimes the non-transmitted *z* value is re-injected *as-is* into the final result, which gives the illusion that a truly three-dimensional coordinate operation has been applied but produced the wrong result.

The ISO 19111 model tries to avoid this error by forcing users to handle (*latitude*, *longitude*, *ellipsoidal height*) coordinates as a whole. This is important for geospatial data integrity. Some may argue that it should be the responsibility of implementations, but implementers themselves (especially in small projects) may lack this expertise. Furthermore, except for the most trivial cases or for the cases listed in the EPSG database, finding a transformation path between two arbitrary CRSs involves heuristic rules which sometimes look like black magic. This issue can be seen on the mailing lists of open-source map projection libraries, with endless flow of questions from users surprised by some transformation results. For this reason, advanced users sometimes prefer to build their transformation chains themselves using the EPSG codes of coordinate operations (instead of EPSG codes of CRS), at the cost of exposing themselves to the above-cited error.

For the same reasons that `EllipsoidalCS` forces users to handle (*latitude*, *longitude*, *ellipsoidal height*) coordinates as an indivisible tuple, it is desirable to have a `MinkowskiCS` class forcing users to handle (*x*, *y*, *z*, *t*) coordinates as an indivisible tuple when relativistic effects are expected to be significant. Note that "indivisible tuple" here actually means "hardly divisible when using the ISO 19111 model." Implementations can always find their way if they really want to.

8.5.3. Temporal epoch

The use of a temporal axis usually requires the definition of a temporal epoch provided in a `TemporalDatum`. But the CRS associated to `MinkowskiCS` may be already associated to another datum such as `EngineeringDatum`, and a CRS can not be associated to two datums. However,

this issue may not apply because there is no absolute time in the context of special relativity. Specifying a time is done by declaring that an event happened simultaneously with the clock showing some numbers. But two events that are simultaneous according one observer may not be simultaneous according another observer. Considering reference frames as observers, the use of epoch date in temporal datum depends on the source and target CRS. Consequently transformation of time coordinates between two relativistic CRSs may need to be specified in an EPSG-like database for each pair of CRSs for reasons more fundamental than only unit conversion or epoch shift. Given that transformations can not be inferred from only epoch dates, there is perhpas no need to specify those epochs in datums of relativistic CRS.

8.6. Relativist coordinate operations

Lorentz contractions (Clause 5.8.1) depend on relative velocity. Consequently, the velocity information needs to be provided somewhere. There are three possible approaches, each with different tradeoffs.

8.6.1. Velocity as operation parameter

The first approach is to consider only the relative velocity between two reference frames. This velocity can be high, so relativist effects *between the two frames* are important. However, inside each reference frame, the features would be considered almost stationary (moving slowly enough for ignoring relativist effects). With this approach, coordinate transformations require only the feature location in the source CRS because the feature velocity *relative to the source CRS* is considered negligible. Even if the feature velocity relative to the *target* CRS is not negligible, the relativist effects are handled by the transformation from source CRS to target CRS. Put another way, the velocity parameter needed for computing Lorentz contractions is considered part of the coordinate operation (i.e., depends only on the source CRS, target CRS, and optionally an epoch), and the feature velocity is considered to add only a negligible change to that. This approach requires no change to existing ISO 19111 standard.

8.6.2. Position with derivatives

The second approach is to consider that not only the source and target CRSs may have high relative velocities, but in addition features may also have high velocities relative to the reference frame of their CRS. In this situation, the relative velocity between the two CRSs is not sufficient. Coordinate operations also need the relative velocity between the feature to transform and the source CRS. This information could be provided by expanding the `CoordinateOperation` class with methods shown below. The methods without the `Velocity` argument already exist (in slightly different forms) in existing standards. The methods with the `Velocity` argument are proposed additions. Note that those additions are sometimes useful even in non-relativist transformations, for example for computing a trajectory.

```
/**
 * Transforms a stationary point from source CRS to target CRS.
 */
```

```
DirectPosition transform(DirectPosition p);

/**
 * Transforms a point moving at the given velocity relative to the source CRS.
 */
DirectPosition transform(DirectPosition p, Velocity v);

/**
 * Gets the derivative of this transform at a stationary point.
 */
Matrix derivative(DirectPosition p);

/**
 * Gets the derivative of this transform at a point moving at the given
velocity.
 */
Matrix derivative(DirectPosition p, Velocity v);
```

Figure 25 — Coordinate operation methods

8.6.3. Six-dimensional CRS

The third approach is to specify velocities as in Clause 8.6.2, but using six-dimensional CRS with (*x*, *y*, *z*, *Vx*, *Vy*, *Vz*) axes instead of adding methods with `Velocity` argument. This approach avoids the need to modify the API. However, its compatibility with current standards is still uncertain because *x* and *Vx* (for example) would both have the same `AxisDirection` code list value, and some map projection libraries do not allow the same axis direction to appear twice in the same CRS (except temporal directions `future` and `past` because meteorological models can have two time axes). The use of six independent axes also hides (at least in ISO 19111 model) the *Vx*=∂x/∂t, *Vy*=∂y/∂t and *Vz*=∂z/∂t relationships.

Clause 8.6.1 can coexist with the two other approaches. This is an approximation that is often good enough, and compatible with existing standards. The two other approaches are more accurate, but require extensions (Clause 8.6.1) or at least clarification in the interpretation (Clause 8.6.3) of existing standards. For consistency, it may be preferable to choose only one of the two latter approaches.

8.7. Compact WKT

The Well-Known Text (WKT) format can be verbose because of repetitions. For example, if a file provides many CRS definitions, the same `DATUM` element may be repeated inside many of `GEODETICCRS` elements. The expansion becomes bigger when the file provides `COORDINATEOPERATION` definitions, because the same source and target CRS can be repeated many times and each repetition consumes tens of lines. Consider for example a concatenated operation:

```
CONCATENATEDOPERATION["A to D",
  SOURCECRS[A[…]],
  TARGETCRS[D[…]],
  STEP[COORDINATEOPERATION[SOURCECRS[A[…]], TARGETCRS[B[…]], …]],
  STEP[COORDINATEOPERATION[SOURCECRS[B[…]], TARGETCRS[C[…]], …]],
```

```
STEP[COORDINATEOPERATION[SOURCECRS[C[…]], TARGETCRS[D[…]], …]]]
```

Figure 26 — CRS repetitions in concatenated operations

Each CRS (A, B, C, and D in above example) is defined twice. If some of those CRSs are derived or projected CRSs, then in addition to a CRS repetition, the operation parameters inside the CRS definitions are also susceptible to repetition in the `COORDINATEOPERATION` body. This repetition problem does not occur in the definition of a single CRS, which is the main purpose of WKT format. But it occurs and can become quite acute when defining a coordinate operation, or when defining more than one CRS in the same file.

GML can avoid this problem by defining each element only once, then referring to previous definitions using `xlink`. An example applied to `CoordinateOperation` was shown in Clause 5.6.1. Links exist also in YAML, but the WKT format and JSON (when no extension is added) lack an equivalent mechanism.

For reducing verbosity, the WKT format could be extended with aliases. This is not exactly the linking mechanism described in above paragraph, but it achieves similar results for the purpose of compactness. The WKT format could be extended by giving special meanings to the `=` and `$` characters when they appear outside quoted text. Those characters are not valid in Unicode identifiers and cannot appear outside quoted texts according to the current WKT specification, so this proposal would not create ambiguity. The extension would introduce the following syntax:

```
<Unicode identifier> = <WKT definition fragment>
!! Restriction: the same Unicode identifier cannot be assigned twice in the
same file.
```

Figure 27 — Definition of an alias

The WKT fragment can be anything provided that brackets or parenthesis are balanced. The occurrence of last closing bracket or parenthesis identifies the end of the WKT fragment. Then, the `$<Unicode identifier>` would be allowed to appear anywhere outside quoted text in the same file. The effect would be as if the `<WKT definition fragment>` associated to the Unicode identifier was copied inline.

The following example creates `DEG`, `METRE`, `WGS84`, `LATLON`, and `WORLD` aliases. The alias names have no special meaning and can be anything. An alias can use previously defined aliases (e.g. `DEG` in this example). The WKT element which is not an alias (when there is no `=` sign) is the main definition, which is `GEODCRS["WGS 84", …]` in this example:

```
DEG    = ANGLEUNIT["degree", 0.0174532925199433]
METRE  = LENGTHUNIT["metre", 1]
WGS84  = DATUM["World Geodetic System 1984",
           ELLIPSOID["WGS 84", 6378137.0, 298.257223563, $METRE]],
           PRIMEM["Greenwich", 0.0, $DEG]

LATLON = CS[ellipsoidal, 2],
           AXIS["Latitude (B)", north, ORDER[1]],
           AXIS["Longitude (L)", east, ORDER[2]],
           $DEG

WORLD  = AREA["World."],
         BBOX[-90.00, -180.00, 90.00, 180.00]

GEODCRS["WGS 84", $WGS84, $LATLON,
```

```
SCOPE["Horizontal component of 3D system (…snip…)."],
$WORLD, ID["EPSG", 4326]]
```

Figure 28 — WKT alias examples

The `WGS84`, `LATLON`, and `WORLD` aliases are not useful when defining a single CRS as shown above, but become more useful if EPSG:4326 is followed by the definitions of many other CRSs in the same file. Aliases can also be used for full CRS definitions, which is very handy for the `CONCATENATEDOPERATION` case mentioned at the beginning of this section.

This extension has existed for 7 years in Apache SIS (Clause 9). That implementation shows that this extension proposal can work, but it does not mean that an OGC standard would need to be identical.

8.8. User-defined transformation registry

The EPSG database contains not only the definitions of thousands of CRSs but also contains transformations between various pairs of CRSs. Those transformation definitions are used by *late-binding* implementations of map projection libraries. A consequence of this architecture is that if a CRS has the following characteristics:

- is defined outside the EPSG database;
- is not a `ProjectedCRS`, `DerivedCRS`, or `BoundCRS`or is one of those types not defined in the EPSG database; and
- the CRS definitions are the only information (no `COORDINATEOPERATION[…] supplied),

then the software may not be able to perform coordinate transformations. Software will need some mechanism for completing the EPSG database with a user-defined registry of transformations.

A simple approach could be to encourage data producers to write `COORDINATEOPERATION` elements in Well Known Text (WKT) or GML format in a single text file. A single file could contain an arbitrary number of elements. Using the compact WKT proposal (Clause 8.7), the WKT file should be reasonably small and efficient to parse. Data producers would publish that file at some stable URL on their web site. CRS definitions would contain a link to this transformation registry. Software could then load that registry when first needed.

One issue is to decide how to specify the URL to a geodetic registry where map projection libraries can download coordinate operations. A possible location using current ISO 19111 and ISO 19115 models could be at the following path:

CRS.name → Identifier.authority → Citation.onlineResource

Where the `OnlineResource` would have the following property values:

- `linkage` = URL to the WKT file containing coordinate operations;

- `applicationProfile` = (to be determined); and
- `function` = `coordinateOperationRegistry` (a new code list value added in `OnLineFunction`). Alternatively, the function could be the existing `download` standard value if there is another way to specify that the purpose is to provide coordinate operations in WKT format.

The above proposal can be done without changing the ISO 19111 model. However, it would require a new keyword in the WKT format.

8.9. User-defined operation methods

The definitions of Coordinate Transformations are restricted to the set of Operation Methods known to the implementation software. Each operation method needs to be associated to code in a programming language such as Java or C/C++; there is currently no way to define an executable `OperationMethod` in an implementation independent way. The best that can be done is to standardize the *names* of methods and parameters, then describe their use in a human-readable document such as EPSG guidance notes or TestBed 18 D023 Engineering Report. This approach works well as long as the standardized methods are sufficient. However, if a user requires formulas which are not included in the set of predefined operation methods, current OGC standards provide no help.

Implementation independent (in a given programming language) operation methods are technically feasible. Some programming languages have a concept of *Service Providers* which allows users to augment some aspects (services) of a library. For example ,in the Java language, ISO 19111 operation methods are represented by instances of the standard `org.opengis.referencing.operation.OperationMethod` Java interface (defined by OGC GeoAPI 3.0.2). If a GeoAPI implementation chooses to accept user supplied `OperationMethod` instances, then that implementation would put the following line in its `module-info.java` file:

```
module com.mylib {
    requires org.opengis.geoapi;

    uses org.opengis.referencing.operation.OperationMethod;
}
```

Figure 29 — Java implementation accepting user supplied operation methods

If a user named Alice defines her own operation method in a class named `people.alice.MyTrajectoryConverter`, then she can plugin that operation in the `com.mylib` library with the following lines in her `module-info.java` file:

```
module people.alice {
    requires com.mylib;
    requires org.opengis.geoapi;

    provides org.opengis.referencing.operation.OperationMethod
        with people.alice.MyTrajectoryConverter;
```

```
}
```

Figure 30 — Custom operation method added to a Java library

However, for making the above code useful, the `OperationMethod` Java interface currently defined by GeoAPI 3.0.2 would need to be extended. A key goal of the GeoAPI development work is to translate existing OGC/ISO abstract models to Java interfaces. GeoAPI does not add classes or properties that are not defined by other OGC/ISO standards, except for better integration with the target programming language. The properties defined by ISO 19111 `OperationMethod` class provide metadata about the coordinate operation but provide nothing for materializing the method in executable code. For the use cases described in this Engineering Report, the following method would need to be added to GeoAPI `OperationMethod` interface:

```
/**
 * Creates a math transform from the specified group of parameter values.
 *
 * @param  factory     the factory to use if this constructor needs to create
other math transforms.
 * @param  parameters  the parameter values that define the transform to
create.
 * @return the math transform created from the given parameters.
 * @throws InvalidParameterNameException if the given parameter group contains
an unknown parameter.
 * @throws ParameterNotFoundException if a required parameter was not found.
 * @throws InvalidParameterValueException if a parameter has an invalid value.
 * @throws FactoryException if the math transform cannot be created for some
other reason
 *         (for example a required file was not found).
 */
MathTransform createMathTransform(MathTransformFactory factory,
ParameterValueGroup parameters)
        throws InvalidParameterNameException, ParameterNotFoundException,
               InvalidParameterValueException, FactoryException;
```

Figure 31 — Materializing `OperationMethod` to executable code in GeoAPI

This is demonstrated in the `org.opengis.testbed.T18D025` Java demo in the OGC GitHub repository using an interface specific to the Apache SIS project (Clause 9.1) as a workaround for the absence of the above method in GeoAPI interface.

An alternative approach would be to specify formulas in CQL2, which supports trigonometric functions, among others. CQL2 may be sufficient for relatively simple coordinate operations, but more advanced coordinate operations may still need to be encoded in a programming language. For example, reverse map projections often use iterative algorithms, which require loops with non-trivial stop conditions. Coordinate operations may also need to load files, for example the trajectory in Clause 5.7. Coordinate operations are often complex enough to require the definitions of private methods, use of exception handling, *etc.*

8.10. Reverse operation method

A coordinate operation sometimes needs to use the reverse of a coordinate operation defined in a geodetic registry. For example, “Mercator” is defined in the EPSG database as an operation method from geographic CRS to projected CRS. However, there is no registered operation for doing the coordinate operation in the reverse direction. In some cases, the reverse operation can be executed by using the forward operation with parameter values of opposite sign. But this is not the case of every operation method.

The legacy OGC 01-009 specification resolved this ambiguity by introducing an `INVERSE_MT` keyword in WKT format. This keyword means that the coordinate operation to apply is the reverse of the coordinate operation contained inside the `INVERSE_MT[...]` element. There is no equivalent feature in ISO 19111 (abstract model), ISO 19162 (WKT), or in GML format.

ISO 19162 discusses this issue in the *WKT representation of concatenated coordinate operations* section. The ISO approach is based on conventions on the CRS declaration order. For example, if the target CRS of a `ConcatenatedOperation` is the source CRS of the last step, the last step must be executed in reverse direction. This approach works in the context of concatenated operations. For the reverse direction of a standalone operation, a workaround may be to wrap the operation in a concatenated operation having a single step.

The Testbed 18 participants invented a scenario where coordinates are transformed from Voyage CRS to observatory CRS. The scenario was defined in that direction rather than the opposite direction in order to avoid the difficulty of expressing the reverse of EPSG:9837 (Geographic/topocentric conversions) operation method. For allowing scenarios in both directions, one of the following actions should be considered:

- introduce something equivalent to OGC 01-009 `INVERSE_MT` keyword; or
- document in a more prominent place how implementations should infer the direction of an operation method in the general case (not only in concatenated operations).

8.11. Summary of proposed extensions

Extensions that could be done in a separate standard:

- add a `CelestialBody` class (Clause 8.1);
- new `Extent` subclass for regions at a distance from a celestial body (Clause 8.2);
- add new axis direction codes for declination and right ascension;
- if desired (it may not be the case), new code list values for axis directions such as `planetocentricX`, `planetocentricY`, and `planetocentricZ` (Clause 8.3);

- add an `InertialCRS` class and its `InertialDatum` dependency (Clause 8.4);
- add a `MinkowskiCS` coordinate system (Clause 8.5);
- compact WKT for definitions of complex transformation chains (Clause 8.7);
- standard location in the CRS metadata where to specify the URL to a transformation registry (Clause 8.8);
- add `trajectory` content type to GGXF format (Clause 5.7.1); and
- add `relativistTransformation` content type to GGXF format (Clause 5.7.2).

Extensions requiring modifications to existing standards:

- add an association from ISO 19111 `GeodeticReferenceFrame` to `CelestialBody` (Clause 8.1);
- add an association from ISO 19115 `GeodeticExtent` to `CelestialBody` (Clause 8.1.1);
- remove the `EngineeringCS` union from ISO 19111 standard (Clause 8.5.1);
- new WKT keyword in ISO 19162 for the URL to a transformation registry (Clause 8.8);
- materialize user supplied operation methods as executable code (Clause 8.9).
- allow source and target CRS in `<gml:Conversion>` (Clause 5.6.1.1) — *possibly a GML bug*;
- upgrade GML schema from ISO 19111:2007 to ISO 19111:2019 (Clause 5.6.1) — maybe already done in ISO 19136:2020;
- allow some GGXF content types to have a single `CRS` attribute instead of `sourceCRS`, `targetCRS` and `interpolationCRS` (Clause 5.7.2); and
- add a `CoordinateOperation.transform(DirectPosition, Velocity)` method (Clause 8.6.2).

Potentially useful features derived from previous standards:

- coordinate operation derivative as a Jacobian matrix (Clause 7.1); and
- WKT keyword for reverse operations (Clause 8.10).

9

IMPLEMENTATIONS

9 IMPLEMENTATIONS

ISO 19111 is an abstract model. It defines data structures with UML diagrams. Those data structures that can be implemented as file formats, database tables, or classes in object-oriented languages are applicable. Some known implementations are as follows.

- **As software:**
- Apache SIS (all versions)
- PROJ (since version 6)
- **As API:**
- OGC GeoAPI (OGC 09-083r4)
- **As database schema:**
- EPSG geodetic dataset
- **As data format:**
- Well-Known Text (WKT) — join ISO 19162 and OGC 18-010r7 standard
- Geographic Markup Language (GML) — join ISO 19136 and OGC 07-036r1 standard
- PROJ-JSON

The OGC GeoAPI standard translates the ISO 19111 model to interfaces in Java and Python programming languages. At least two implementations are known to follow the ISO 19111 model closely, either as a direct implementation of GeoAPI or by taking inspiration from it:

- Apache Spatial Information System (SIS) — implements GeoAPI 3.0.1 directly; and
- PROJ 6 and later — implements GeoAPI 3.0.1 through PROJ-JNI.

Those two projects differ in many aspects (programming language, *etc.*), but one difference relevant to this discussion is management of a geodetic registry as described below.

- Apache SIS connects to a database software of user-choice (PostgreSQL, MySQL, Derby, *etc.*), searches for an EPSG schema, and uses that schema *as-is*. This approach gives users more control on their geodetic dataset. An inconvenience is that SIS inherits EPSG schema limitations such as the absence of explicit information about celestial bodies.
- PROJ 6+ defines its own database schema, which is inspired by the EPSG schema. EPSG data need to be imported into the PROJ database on SQLite before they can be used. This approach gives less flexibility to users for connecting to their own geodetic dataset, but more flexibility to PROJ for modifying the schema. Those modifications include

the addition of a field for identifying the celestial body. In some ways, PROJ already implements the proposal described in Clause 8.1.

Given a software using an EPSG-like database, if that database content is editable, then custom `EngineeringCRS` and `CoordinateTransformation` instances can be inserted as new records. Custom records may require the following modifications to implementation schema compared to EPSG schema:

- addition of a field for identifying the celestial body (already done by PROJ 6); and
- addition of a codespace (authority) for each CRS code used as a primary key.

With information available in an EPSG-like database, software should be able to perform coordinate transformations between different celestial bodies, orbiting objects, or objects in free flight, using the existing code with few (if any) changes.

The `EngineeringCRS` and `CoordinateTransformation` instances to add could be specified by a GML or WKT file as proposed in Clause 8.8. Implementors are free to decide if they want to modify their EPSG-like database or provide the same effect in a different way. The only requirement would be to behave *as if* the database was augmented with new records. Implementors are free to do the actual storage in a database separate from the EPSG database.

9.1. Demo application

For demonstrating the effectiveness of GML documents shown in this Engineering Report, a demo Java application is provided in Testbed 18 GitHub repository. This application uses Apache SIS, which provides a map projection library capable to read and write GML documents. Apache SIS supports the advanced features used in the `VoyagerToObservatory.xml` and `PoseToRoboticArm.xml` files such as `DerivedCRS`, `CompoundCRS`, `PassThroughOperation`, and `ParameterValueGroup`. The demo application also illustrates the extension mechanism presented in Clause 8.9 by expanding the set of operation methods known to Apache SIS. The operation methods added by the demo application (necessary for parsing the GML files of Testbed 18) are:

- trajectory to Conventional frame;
- to circular orbit (Spherical domain);
- time-dependent longitude rotation (equatorial CS);
- geographic/spherical conversions; and
- pose irregular series.

For this demo, the above operation methods are implemented with simplistic formulas, sometimes a trivial identity operation. But the formula complexity has no impact on the interoperability discussion.

After the above operation methods are made available to Apache SIS using the standard `java.util.ServiceLoader` mechanism, the following lines of code are sufficient for parsing the GML document and transforming coordinates. In this code, `XML.unmarshal` and `GeneralDirectPosition` are specific to Apache SIS. But the results — `CoordinateOperation` and `DirectPosition` — are implementation neutral GeoAPI interfaces.

```
// Standard Java
import java.io.File;
import javax.xml.bind.JAXBException;

// Implementation neutral GeoAPI
import org.opengis.geometry.DirectPosition;
import org.opengis.referencing.operation.CoordinateOperation;
import org.opengis.referencing.operation.TransformException;

// Implementation specific
import org.apache.sis.xml.XML;
import org.apache.sis.geometry.GeneralDirectPosition;

public class Demo {
    public static void main(String[] args) throws JAXBException,
TransformException {

        CoordinateOperation op = (CoordinateOperation) XML.unmarshal(new
File("VoyagerToObservatory.xml"))
        DirectPosition  source = new GeneralDirectPosition(x, y, z, t);
        DirectPosition  target = op.getMathTransform().transform(source, null);

        System.out.println("Coordinates relative to Voyager: "     + source);
        System.out.println("Coordinates relative to observatory: " + target);
    }
}
```

Figure 32 — Demo application reading GML and transforming coordinates

A

ANNEX A (INFORMATIVE) STANDARD PARAMETERS PROPOSAL

ANNEX A (INFORMATIVE) STANDARD PARAMETERS PROPOSAL

This annex proposes definitions for operation methods used by the draft GeoPose standard and/or by the "Voyager to Observatory platform" scenario defined in this Engineering Report (ER). Those definitions are intended for use with ISO 19111 conceptual model. They can be materialized in GML, WKT, or in an EPSG-like database. The following notes apply to all tables in this annex.

- Identifiers in EPSG namespace are EPSG parameter codes, not CRS codes.
- Identifiers in OGC namespace are tentative. They would need to be reviewed by a standard working group, then validated by a OGC naming authority before formal usage.
- All identifiers should be globally unique. Names and aliases should be unique in the scope of an operation method, but it is acceptable if two distinct operation methods use the same name or alias for different things.

A.1. Operation parameters

The following table provides the content of various `OperationParameter` instances. Each row is a single `OperationParameter` instance and each column contains the value to assign to an `OperationParameter` attribute. Those attributes are:

- `identifier` of type `MD_Identifier`;
- `name` also of type `MD_Identifier`;
- `alias` of type `GenericName`; and
- `remarks` of type `CharacterSequence`.

Note that there is currently no attribute for specifying the parameter value type (numerical, array, characters, filename, etc.) or the kind of unit of measurement (angular, linear, temporal, scale, etc.). Those restrictions can be specified only as free text in the `remarks` attribute.

This table only describes the parameters expected by `CoordinateOperation` instances. The values of those parameters are not specified here. Those values can be specified in separated objects of class `ParameterValue`. A parameter value can be represented by an ISO 19103

`Measure` object, which combines a number with a unit of measurement. Consequently, the parameter descriptions in the following table do not need to restrict parameter values to a specific unit of measurement.

Table A.1 — Values of `OperationParameter` instances (column headers are property names)

IDENTIFIER	NAME	ALIAS	REMARKS
OGC:pose-lat	Latitude of pose	lat	Number in [-90° … +90°] range.
OGC:pose-lon	Longitude of pose	lon	Number in [-180° … +180°] range.
OGC:pose-h	Ellipsoidal height of pose	h	Linear unit (e.g.,er).
OGC:pose-TU	Topocentric U of pose	U	Linear unit.
OGC:pose-TV	Topocentric V of pose	V	Linear unit.
OGC:pose-TW	Topocentric W of pose	W	Linear unit.
OGC:yaw	Yaw		Angular unit.
OGC:pitch	Pitch		Angular unit.
OGC:roll	Roll		Angular unit.
OGC:quaternion-x	Quaternion X	x	
OGC:quaternion-y	Quaternion Y	y	
OGC:quaternion-z	Quaternion Z	z	
OGC:quaternion-w	Quaternion W	w	
OGC:start-instant	Start instant		ISO-8601 or duration since Unix epoch.
OGC:end-instant	End instant		ISO-8601 or duration since Unix epoch.
OGC:valid-time	Valid time		ISO-8601 or duration since Unix epoch.
OGC:interpose	Inter-pose duration		Temporal unit (e.g., milliseconds).
OGC:pose-location	Pose location		Sequence of 3 coordinate values.
OGC:trajectory	Feature trajectory file		Moving Feature CSV encoding format.

IDENTIFIER	NAME	ALIAS	REMARKS
OGC:orbit-radius	Orbit radius		Linear unit (e.g., kilometer).
OGC:orbit-speed	Orbit speed		Speed unit (e.g., kilometer per second).
OGC:vernal-equinox	Vernal equinox		Date when heliocentric longitude is zero.
OGC:lon-rotation	Longitude axis rotation		Angular unit (typically degree).
OGC:lon-rate-change	Rate of change of longitude axis rotation		Angle per time.
EPSG:1047	Parameter reference epoch		Temporal unit (typically year).

The parameters listed in the above table will be used in the next sections for `OperationParameterGroup` and `OperationMethod` definitions. The same parameter instances can be shared by many operation methods or groups.

A.2. Operation parameter groups

The following tables provide the content of various `OperationParameterGroup` instances. A parameter group allows repeating a set of parameters many times, for example at different times for creating a piecewise function. Parameter groups are rarely used, but nevertheless are used in this ER for illustrating an alternative way to encode spacecraft trajectory.

Table A.2 — Definition of frame (regular) parameter group

`OperationParameterGroup` PROPERTY	PROPERTY VALUE
name	“Frame specification (regular)”
identifier	“OGC:pose-regular-frame”
parameter	• OGC:pose-location • OGC:yaw • OGC:pitch • OGC:roll
minimumOccurs	1
maximumOccurs	Infinity

Table A.3 — Definition of frame (irregular) parameter group

`OperationParameterGroup` PROPERTY	PROPERTY VALUE
name	“Frame specification (irregular)”
identifier	“OGC:pose-irregular-frame”
parameter	• OGC:valid-time • OGC:pose-location • OGC:yaw • OGC:pitch • OGC:roll
minimumOccurs	1
maximumOccurs	Infinity

A.3. Operation methods

The following tables provide the content of various `OperationMethod` instances. Each operation method uses some of the `OperationParameter` instances defined in the previous section. Some operation methods also use `OperationParameterGroup` instances, in which case the group contents are repeated in the tables for convenience. The “Advanced GeoPose” method is intentionally excluded (see Clause 6.2 for a rational).

A.3.1. Methods derived from GeoPose

The following operation methods are derived from the draft GeoPose specification. They are not used for the “Voyager to Observatory” scenario of this ER.

Table A.4 — Definition of *Basic-YPR* operation method

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
name	“Pose rotation by YPR”
identifier	“OGC:pose-rotation-YPR”
parameter	• OGC:pose-lat • OGC:pose-lon • OGC:pose-h • OGC:yaw • OGC:pitch

OperationMethod PROPERTY NAME	PROPERTY VALUE
	• OGC:roll
formulaReference	TODO: formula or reference to a formula

Table A.5 — Definition of *Basic-Quaternion* operation method

OperationMethod PROPERTY NAME	PROPERTY VALUE
name	"Pose location and orientation by quaternion"
identifier	"OGC:pose-topocentric-location-orientation-by-quaternion"
remarks	"Sum of squares of quaternion x, y, z and w shall be 1 (ignoring rounding errors)."
parameter	• OGC:pose-lat • OGC:pose-lon • OGC:pose-h • OGC:quaternion-x • OGC:quaternion-y • OGC:quaternion-z • OGC:quaternion-w
formulaReference	TODO: formula or reference to a formula

Table A.6 — Definition of *Regular-Series* operation method

OperationMethod PROPERTY NAME	PROPERTY VALUE
name	"Pose regular series"
identifier	"OGC:pose-regular-series"

parameter	• OGC:start-instant
	• OGC:end-instant
	• OGC:interpose-duration
	• OGC:pose-regular-frame *(a parameter group, which is repeatable)*
	  • OGC:pose-location
	  • OGC:yaw
	  • OGC:pitch
	  • OGC:roll

formulaReference	TODO: formula or reference to a formula

Table A.7 — Definition of *Irregular-Series* operation method

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
name	"Pose irregular series"
identifier	"OGC:pose-irregular-series"

parameter	• OGC:start-instant
	• OGC:end-instant
	• OGC:pose-irregular-frame *(a parameter group, which is repeatable)*
	  • OGC:valid-time
	  • OGC:pose-location
	  • OGC:yaw
	  • OGC:pitch
	  • OGC:roll

formulaReference	TODO: formula or reference to a formula

A.3.2. Methods required by the ER scenario

The following operation methods are required by the "Voyager to Observatory" scenario defined in this ER. Those operations appear in the `VoyagerToObservatory.xml` file and in the WKT of this document. They are listed here only for illustrating the work that needs to be done for celestial coordinate transformations. Real use case would need more sophisticated operations than the ones described here.

Table A.8 — Definition of *Trajectory to Conventional frame* operation method

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
name	"Trajectory to Conventional frame"
identifier	"OGC:trajectory-to-conventional"
parameter	• OGC:trajectory
formulaReference	TODO: formula or reference to a formula

Table A.9 — Definition of *To circular orbit (Spherical domain)* operation method

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
name	"To circular orbit (Spherical domain)"

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
identifier	“OGC:ToCircularOrbit”
parameter	• OGC:orbit-radius • OGC:orbit-speed • OGC:vernal-equinox
formulaReference	TODO: formula or reference to a formula

Table A.10 — Definition of *Time-dependent longitude rotation (equatorial CS)* operation method

`OperationMethod` PROPERTY NAME	PROPERTY VALUE
name	“Time-dependent longitude rotation (equatorial CS)”
identifier	“OGC:celestial-to-geodetic”
parameter	• OGC:lon-rotation • OGC:lon-rate-change • EPSG:1047 — Parameter reference epoch
formulaReference	TODO: formula or reference to a formula

B

ANNEX B (INFORMATIVE) GML VERBOSITY

ANNEX B (INFORMATIVE) GML VERBOSITY

GML appears more verbose than WKT because of:

- `<element> … </element>` syntax (compared to `ELEMENT[…]`);
- the use of a `<property><Type>` pattern which increases the amount of nested elements;
- Additional information or explicitness (e.g. scope and identifier as mandatory elements); and
- redundancy in object identifications (`gml:id` and `<gml:identifier>`).

The next sections provide more details on issues that are not constrained by the XML language.

B.1. Mandatory identifiers

GML 3.2 requires two kinds of identifiers, both of them mandatory:

- `gml:id` attributes, which are local to the XML document and does not need to contain standard values; and
- `<gml:identifier>` elements, which refer to some dataset outside the document (e.g. primary keys in a database).

The `gml:id` attribute is used as the `xlink` target for reusing a previously defined element without repeating its content. `gml:id` is very useful when needed, but unnecessary when the XML fragment is not the target of at least one `xlink` reference. Unfortunately, `gml:id` is declared mandatory in all GML elements. Consequently, GML files are inflated by fragments like those below even when the elements are not intended to be referenced.

```
<gml:Transformation gml:id="InertialToEarthFixed">
  <gml:identifier codeSpace="TB18-D025">InertialToEarthFixed</gml:identifier>
  ... definition here ...
</gml:Transformation>
```

Figure B.1 — Identifiers in GML elements

Instead of:

```
<gml:Transformation>
  ... definition here ...
</gml:Transformation>
```

Figure B.2 — GML element without identifier

Usually, only a small number of elements in a GML file need to be referenced from other elements (`gml:id`). Likewise, only a small number of elements are intended to be reachable from a persistent database (`<gml:identifier>`). For example the `VoyagerToObservatory.xml` file in Testbed 18 contains 69 `gml:id` attributes, while only 21 of them are really needed, and none of the 69 `<gml:identifier>` elements are needed. About 10% of `VoyagerToObservatory.xml` lines — ignoring comments — are unnecessary `<gml:identifier>`. Even more irritating is when the values of most `<gml:identifier>` elements are only repetitions of the `gml:id` attribute value appearing immediately before them, as in the above example.

Mandating an identifier for all elements not only makes GML files more verbose and redundant, but it also makes GML more tedious to write because it forces authors to invent an identifier for every single element. This requirement is error-prone and often results in misleading identifiers because developers typically create new CRS definitions by doing a copy-and-paste of an existing definition, editing a few elements, then forgetting to change the identifiers of the edited elements. See Annex C for examples of such errors in the OGC definitions server.

Mandating identifiers also makes all elements reachable from external documents. By analogy with object oriented programming, this is equivalent to forcing developers to declare all elements as public and to forbid private elements. This goes against encapsulation, where the capability to make an element unreachable (except indirectly through the parent element) is desirable because (among other reasons) it gives more freedom to make changes to that element in future versions.

This problem is not inherited from OGC/ISO abstract models. Identifiers are optional in both ISO 19111 abstract model and in ISO 19162 WKT specification.

B.2. “Property → type” pattern

The `<property><Type>` pattern is found in GML and ISO 19115-3 (formerly ISO 19139) files, where each property’s value is wrapped in another XML element defining the value type. Often the names are identical except for the case of the first letter. Example (omitting `gml:` prefix for brevity):

```
<parameterValue>
  <ParameterValue>
    ... definition here ...
  </ParameterValue>
</parameterValue>
```

Figure B.3 — Example of `<property><Type>` pattern in GML

A possible rational for this pattern is to support specifying the subtype of the child element. In the following example, `CartesianCS` is a subtype of `CoordinateSystem` (see also Annex B.3 for the more redundant GML way to encode this information):

```
<coordinateSystem>
  <CartesianCS>
    ... definition here ...
  </CartesianCS>
</coordinateSystem>
```

Figure B.4 — GML way to specify a coordinate system

But the above pattern could have been replaced by the following snippet instead (following is not standard GML, but shows how GML could have been defined):

```
<coordinateSystem xsi:type="gml:CartesianCS">
  ... definition here ...
</coordinateSystem>
```

Figure B.5 — How coordinate system could have been specified

The participants are not aware of the technical reasons for the use of the `<property><Type>` nested element pattern instead of `xsi:type` attribute. The `<property><Type>` pattern is not only more verbose, but it is also more difficult to implement in frameworks like Java XML Bindings (JAXB) because it forces developers to create wrapper classes (reflecting XML wrapper elements) for every property. Apache SIS has about 150 such "useless" classes (hidden from the public API) for handling this redundancy in ISO 19115 and ISO 19111 XML.

The `<property><Type>` pattern also caused another problem. About 10 years ago, a popular approach was to automatically generate Java classes from XML schema using an "XML to Java" compiler. Some "OGC API" projects flourished on the web (outside OGC) where authors did exactly that with GML schema. The compiler blindly reproduced the `<property><Type>` nested element pattern in Java classes, which resulted in twice the amount of Java classes compared to what really exists in the conceptual model. Those compilation results were published in non-OGC projects without cleanup. It is not clear if the developers saw the problem. At that time, XML schema were thought by some as a source of truth even more authoritative than UML. This inflation in the programming language API contributed to the perceived complexity of OGC standards.

B.3. Properties with type-dependent names

A single property in the ISO abstract model is sometimes materialized by many properties in GML schema, with one property for each possible value type. For example, in addition to the `<coordinateSystem>` property, the GML schema also defines `<affineCS>`, `<cartesianCS>`, `<cylindricalCS>`, `<ellipsoidalCS>`, `<linearCS>`, `<polarCS>`, `<sphericalCS>`, and `<userDefinedCS>` properties with identical content except that the value type is restricted to a specific coordinate system subtype. For example the `<coordinateSystem>` snippet in Annex B.2 can also be written as below:

```
<cartesianCS>
  <CartesianCS>
```

```
    ... definition here ...
  </CartesianCS>
</cartesianCS>
```

Figure B.6 — GML way to specify a coordinate system (more redundant)

This approach adds redundancy in the schema for no new information. This redundancy vastly increases the amount of code generated by automatic tools like JAXB (tools generating Java codes from XML schema). For example in Java, a simple task which should be a single line of Java code — `getCoordinateSystem()` — requires about 50 lines of code when using an API generated automatically from the GML schema. This complexity is caused by the necessity to explore many paths for all `cartesianCS`, `affineCS`, *etc.* possible elements and their nested children.

This redundancy does not exist in ISO 19111 abstract model; it is a GML addition. It is also at odds with object-oriented programming paradigms, where the use of a generic type also accepts all possible subtypes. In XML, the subtype can be specified with the `xsi:type` attribute mentioned in Annex B.2. Even if `xsi:type` is not used, the more verbose `<property><Type>` pattern already serves the same purpose. The rational for adding yet another (apparently fully redundant) typing mechanism is unclear. The complexity cost is considerable when using binding frameworks such as JAXB, for no new information.

B.4. Recommendation for more compact GML

Except for the `<element> … </element>` syntax, the above-cited constraints making GML so verbose are not really constraints with the XML language. The GML verbosity problem is largely due to disconcerting design choices in the GML schema rather than to the XML format. The bad reputation that the XML format received is not always deserved. GML could be simplified in the following ways.

- **Make all identifiers optional** (compatible change):
  - `gml:id` and `<gml:identifier>` are often redundant;
  - They are pertinent in a minority of cases; and
  - Their frequent misleading values (from copy-and-paste) may cause more harm than good.
- **Remove all properties that are repetitions of a more generic property** (compatible change with deprecation cycle):
  - CRS should have a single `<coordinateSystem>` property, no `<cartesianCS>`, `<ellipsoidalCS>`, `<sphericalCS>`, *etc.*
- **Replace the `<property><Type>` nested element pattern by the more standard `xsi:type` attribute** (incompatible change).

The first two points may be relatively easy to apply. But the last point would require major changes in the GML standard. More compact XML documents for encoding CRS and coordinate operations are technically possible, but a question would be how far any revision can go with incompatible changes (impacts on backwards compatibility).

For visualizing what the changes would be, consider the following valid GML file. This example is derived from EPSG::4326 with remarks and other metadata omitted.

```
<gml:GeodeticCRS gml:id="epsg-crs-4326">
  <gml:identifier codeSpace="IOGP">urn:ogc:def:crs:EPSG::4326</gml:identifier>
  <gml:name codeSpace="EPSG">WGS 84</gml:name>
  <gml:domainOfValidity>
    <gmd:EX_Extent id="extent-world">
      <gmd:description>
        <gco:CharacterString>World.</gco:CharacterString>
      </gmd:description>
      <gmd:geographicElement>
        <gmd:EX_GeographicBoundingBox>
          <gmd:westBoundLongitude>
            <gco:Decimal>-180.0</gco:Decimal>
          </gmd:westBoundLongitude>
          <gmd:eastBoundLongitude>
            <gco:Decimal>180.0</gco:Decimal>
          </gmd:eastBoundLongitude>
          <gmd:southBoundLatitude>
            <gco:Decimal>-90.0</gco:Decimal>
          </gmd:southBoundLatitude>
          <gmd:northBoundLatitude>
            <gco:Decimal>90.0</gco:Decimal>
          </gmd:northBoundLatitude>
        </gmd:EX_GeographicBoundingBox>
      </gmd:geographicElement>
    </gmd:EX_Extent>
  </gml:domainOfValidity>
  <gml:scope>Horizontal component of 3D system.</gml:scope>
  <gml:ellipsoidalCS>
    <gml:EllipsoidalCS gml:id="epsg-cs-6422">
      <gml:identifier codeSpace="IOGP">urn:ogc:def:cs:EPSG::6422</gml:
identifier>
      <gml:name codeSpace="EPSG">Ellipsoidal 2D CS.</gml:name>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9122" gml:id=
"epsg-axis-106">
          <gml:identifier codeSpace="IOGP">urn:ogc:def:axis:EPSG::106</gml:
identifier>
          <gml:name codeSpace="EPSG">Geodetic latitude</gml:name>
          <gml:axisAbbrev>Lat</gml:axisAbbrev>
          <gml:axisDirection codeSpace="EPSG">north</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
      <gml:axis>
        <gml:CoordinateSystemAxis uom="urn:ogc:def:uom:EPSG::9122" gml:id=
"epsg-axis-107">
          <gml:identifier codeSpace="IOGP">urn:ogc:def:axis:EPSG::107</gml:
identifier>
          <gml:name codeSpace="EPSG">Geodetic longitude</gml:name>
          <gml:axisAbbrev>Lon</gml:axisAbbrev>
          <gml:axisDirection codeSpace="EPSG">east</gml:axisDirection>
        </gml:CoordinateSystemAxis>
      </gml:axis>
    </gml:EllipsoidalCS>
```

```
  </gml:ellipsoidalCS>
  <gml:geodeticDatum>
    <gml:GeodeticDatum gml:id="epsg-datum-6326">
      <gml:identifier codeSpace="IOGP">urn:ogc:def:datum:EPSG::6326</gml:
identifier>
      <gml:name codeSpace="EPSG">World Geodetic System 1984</gml:name>
      <gml:domainOfValidity xlink:href="#extent-world"/>
      <gml:scope>Satellite navigation.</gml:scope>
      <gml:anchorDefinition>Defined through a consistent set of station
coordinates.</gml:anchorDefinition>
      <gml:primeMeridian>
        <gml:PrimeMeridian gml:id="epsg-meridian-8901">
          <gml:identifier codeSpace="IOGP">urn:ogc:def:meridian:EPSG::8901</
gml:identifier>
          <gml:name codeSpace="EPSG">Greenwich</gml:name>
          <gml:greenwichLongitude uom="urn:ogc:def:uom:EPSG::9102">0.0</gml:
greenwichLongitude>
        </gml:PrimeMeridian>
      </gml:primeMeridian>
      <gml:ellipsoid>
        <gml:Ellipsoid gml:id="epsg-ellipsoid-7030">
          <gml:identifier codeSpace="IOGP">urn:ogc:def:ellipsoid:EPSG::7030</
gml:identifier>
          <gml:name codeSpace="EPSG">WGS 84</gml:name>
          <gml:semiMajorAxis uom="urn:ogc:def:uom:EPSG::9001">6378137.0</gml:
semiMajorAxis>
          <gml:secondDefiningParameter>
            <gml:SecondDefiningParameter>
              <gml:inverseFlattening uom="urn:ogc:def:uom:EPSG::9201">
298.257223563</gml:inverseFlattening>
            </gml:SecondDefiningParameter>
          </gml:secondDefiningParameter>
        </gml:Ellipsoid>
      </gml:ellipsoid>
    </gml:GeodeticDatum>
  </gml:geodeticDatum>
</gml:GeodeticCRS>
```

Figure B.7 — CRS definition using current GML schema (valid)

The following is the same definition as above, but with the proposals in this section applied. The number of lines goes down from 78 to 44. In addition, the "gml:" prefix is omitted for better readability. This is allowed by the XML standard when defining a default namespace.

```
<GeodeticCRS id="epsg-crs-4326">
  <identifier codeSpace="IOGP">urn:ogc:def:crs:EPSG::4326</identifier>
  <name codeSpace="EPSG">WGS 84</name>
  <domainOfValidity id="extent-world">
    <gmd:description>World.</gmd:description>
    <gmd:geographicElement xsi:type="gmd:EX_GeographicBoundingBox">
      <gmd:westBoundLongitude>-180.0</gmd:westBoundLongitude>
      <gmd:eastBoundLongitude> 180.0</gmd:eastBoundLongitude>
      <gmd:southBoundLatitude> -90.0</gmd:southBoundLatitude>
      <gmd:northBoundLatitude>  90.0</gmd:northBoundLatitude>
    </gmd:geographicElement>
  </domainOfValidity>
  <scope>Horizontal component of 3D system.</scope>
  <coordinateSystem xsi:type="EllipsoidalCS">
    <name codeSpace="EPSG">Ellipsoidal 2D CS.</name>
    <axis>
      <name codeSpace="EPSG">Geodetic latitude</name>
      <axisAbbrev>Lat</axisAbbrev>
```

```
      <axisDirection codeSpace="EPSG">north</axisDirection>
    </axis>
    <axis>
      <name codeSpace="EPSG">Geodetic longitude</name>
      <axisAbbrev>Lon</axisAbbrev>
      <axisDirection codeSpace="EPSG">east</axisDirection>
    </axis>
  </coordinateSystem>
  <datum xsi:type="GeodeticDatum">
    <name codeSpace="EPSG">World Geodetic System 1984</name>
    <domainOfValidity xlink:href="#extent-world"/>
    <scope>Satellite navigation.</scope>
    <anchorDefinition>Defined through a consistent set of station coordinates.
</anchorDefinition>
    <primeMeridian>
      <name codeSpace="EPSG">Greenwich</name>
      <greenwichLongitude uom="urn:ogc:def:uom:EPSG::9102">0.0</
greenwichLongitude>
    </primeMeridian>
    <ellipsoid>
      <name codeSpace="EPSG">WGS 84</name>
      <semiMajorAxis uom="urn:ogc:def:uom:EPSG::9001">6378137.0</semiMajorAxis>
      <secondDefiningParameter>
        <inverseFlattening uom="urn:ogc:def:uom:EPSG::9201">298.257223563</
inverseFlattening>
      </secondDefiningParameter>
    </ellipsoid>
  </datum>
</GeodeticCRS>
```

Figure B.8 — CRS definition using pseudo-GML (invalid)

C

ANNEX C (INFORMATIVE) ERRORS IN OGC DEFINITIONS

C

ANNEX C (INFORMATIVE) ERRORS IN OGC DEFINITIONS

The OGC definitions server contains a few planetary CRS definitions. As of October 2022, the server contained definitions for the Moon, Mercury, Venus, and Mars. However, some elements in those definitions have misleading values, in particular (but not limited to) in `gml:id` attributes and `<gml:identifier>` elements. This section describes problems in the file found at the following URL:

http://www.opengis.net/def/crs/IAU/0/Mars2000

But the discussion applies also to other files in the same directory. Note that all problems related to `gml:id` and `<gml:identifier>` could have been avoided if GML identifiers were not mandatory (see Annex B.1).

C.1. Problematic XML fragments

```
<gml:GeographicCRS gml:id="epsg-crs-4326">
```

- **Misleading identifier:** Technically speaking the use of `gml:id="epsg-crs-4326"` on Mars is not wrong because `gml:id` attributes are local to the file and do not need to be a global identifier. However, the `epsg-crs-4326` value strongly suggests the use of EPSG:4326 CRS, which is reserved for a terrestrial CRS. A different identifier should have been invented, for example simply “Mars2000”.

```
<gml:identifier codeSpace="http://www.ietf.org/rfc/rfc3986">
  http://www.opengis.net/def/crs/IAU/0/Mars2000
</gml:identifier>
```

- `http://www.ietf.org/rfc/rfc3986` is not a code space controlled by OGC. A better alternatives would be `urn:ogc:def:crs:IAU` or `http://www.opengis.net/def/crs/IAU`.
- `http://www.opengis.net/def/crs/IAU/0/Mars2000` should be only a code rather than a URL, because the code space is specified by above `codeSpace` attribute. A possible value at this location could be “Mars2000”.

```
<gml:domainOfValidity/>
<gml:scope/>
```

- The optional `<gml:domainOfValidity>` element could be omitted instead of providing an empty element.
- The `<gml:scope>` element is mandatory. Is this CRS suitable for large or small map scales?

```
<gml:usesEllipsoidalCS>
  <gml:EllipsoidalCS gml:id="ogrcrs514">
    <gml:metaDataProperty/>
    <gml:identifier codeSpace="urn:ogc:def:axis:EPSG::">9902</gml:identifier>
```

- `<gml:usesEllipsoidalCS>` is deprecated. It should be `<gml:ellipsoidalCS>`.
- The optional `<gml:metaDataProperty>` element could be omitted instead of providing an empty element.
- **Wrong identifier:** The `urn:ogc:def:axis:EPSG::9902` identifier is related to the geodetic longitude axis, while a coordinate system identifier was expected. A better value would be `urn:ogc:def:cs:EPSG::6422` if the use of terrestrial terminology is considered acceptable (see Clause 8.3).

```
    <gml:axis>
      <gml:CoordinateSystemAxis gml:id="ogrcrs515" uom="http://www.opengis.
net/def/axis/EPSG/0/9102">
        <gml:identifier codeSpace="OGP">http://www.opengis.net/def/axis/
EPSG/0/107</gml:identifier>
        <gml:axisAbbrev>Lat</gml:axisAbbrev>
        <gml:axisDirection codeSpace="IAU">north</gml:axisDirection>
      </gml:CoordinateSystemAxis>
    </gml:axis>
```

- The UOM code should be 9122 instead of 9102 if consistency with EPSG database is desired.
- The `IAU` code space in `<gml:axisDirection>` is a good choice if the intent is that “east” and “north” should be understood as defined by IAU (for example in relation with the direction of planet rotation) instead of as defined by ISO. However, in such a case, the “east” and “north” words may need to be replaced by something more specific for avoiding confusion with the “east” and “north” axis directions defined by ISO. In addition, the `urn:ogc:def:cs:EPSG::6422` identifier suggested above would need to be replaced by something in IAU code space, for example, `urn:ogc:def:cs:IAU::ellipsoidal`. The Identifier of Coordinate System axes would also need to be replaced.

```
    <gml:axis>
      <gml:CoordinateSystemAxis gml:id="ogrcrs516" uom="http://www.opengis.
net/def/axis/EPSG/0/9102">
        <gml:identifier codeSpace="urn:ogc:def:axis:EPSG::">9902</gml:
identifier>
        <gml:axisAbbrev>Lon</gml:axisAbbrev>
        <gml:axisDirection codeSpace="IAU">east</gml:axisDirection>
      </gml:CoordinateSystemAxis>
    </gml:axis>
```

- The UOM code should be 9122 instead of 9102 here if consistency with EPSG database is desired.

- **Wrong identifier:** EPSG:9902 is the identifier of an axis *name*, not a coordinate system axis. This identifier should be `urn:ogc:def:axis:EPSG::106`.
- See comment about `IAU` code space in previous XML fragment.

```
<gml:usesGeodeticDatum>
  <gml:GeodeticDatum gml:id="ogrcrs517">
    <gml:metaDataProperty/>
    <gml:identifier codeSpace="urn:ogc:def:axis:EPSG::">9902</gml:identifier>
```

- `<gml:usesGeodeticDatum>` is deprecated. It should be `<gml:geodeticDatum>`.
- The optional `<gml:metaDataProperty>` element could be omitted instead of providing an empty element.
- **Wrong identifier:** The `urn:ogc:def:axis:EPSG::9902` identifier is related to the geodetic longitude axis, while a datum identifier was expected. Furthermore, EPSG datum cannot be reused for planets other than Earth. A better alternative would be `urn:ogc:def:datum:IAU::Mars2000`.

```
    <gml:usesPrimeMeridian>
      <gml:PrimeMeridian gml:id="ogrcrs518">
        <gml:metaDataProperty/>
        <gml:identifier codeSpace="OGP">http://www.opengis.net/def/meridian/
EPSG/0/8901</gml:identifier>
        <gml:name>Greenwich</gml:name>
        <gml:greenwichLongitude uom="http://www.opengis.net/def/uom/
EPSG/0/9102">0</gml:greenwichLongitude>
      </gml:PrimeMeridian>
    </gml:usesPrimeMeridian>
```

- `<gml:usesPrimeMeridian>` is deprecated and should be `<gml:primeMeridian>`.
- The optional `<gml:metaDataProperty>` element could be omitted instead of providing an empty element.
- **Wrong identifier:** EPSG:8901 is for Greenwich meridian, which is not applicable on Mars.
- **Wrong name:** “Greenwich” is not applicable on Mars. A better name would be “Reference meridian” or “Airy”.

```
    <gml:usesEllipsoid>
      <gml:Ellipsoid gml:id="ogrcrs519">
        <gml:metaDataProperty/>
        <gml:identifier codeSpace="OGP">http://www.opengis.net/def/meridian/
EPSG/0/8901</gml:identifier>
```

- `<gml:usesEllipsoid>` is deprecated and should be `<gml:ellipsoid>`.
- The optional `<gml:metaDataProperty>` element could be omitted instead of providing an empty element.
- **Wrong identifier:** EPSG:8901 is a meridian identifier, while an ellipsoid identifier is expected here. A better value would be “urn:ogc:def:ellipsoid:IAU::Mars2000”.

C.2. Alternative planetary definitions

CRS definitions for Mars are available on this server:

- WKT format: http://voparis-vespa-crs.obspm.fr:8080/web/mars.html
- GML format: TODO

Those definitions avoid the problems described in Annex C.1. A proposed action is to submit those definitions to the OGC planetary group, then if accepted, use them as replacements for the definitions currently on the OGC server.

D

ANNEX D (INFORMATIVE) REVISION HISTORY

ANNEX D (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-11-07	0.1	Martin Desruisseaux	all	First complete version
2022-11-12	0.2	Carl Reed	all	Review
2022-11-18	0.3	Martin Desruisseaux	all	Minor clarifications
2023-01-05	0.4	Martin Desruisseaux	§iv, 5, 6.4, 8.9, 9.1	Replace GitLab URL by Git Hub
2023-01-13	0.5	Martin Desruisseaux	§i, 1, 5.4, 8.4	CRS and planetary WG comments

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Martin Daly: OGC 01-009, *Coordinate Transformation Services — OLE/COM*. Open Geospatial Consortium (2001). https://portal.ogc.org/files/?artifact id=999.

[2] Akinori Asahara, Ryosuke Shibasaki, Nobuhiro Ishimaru, David Burggraf: OGC 14-084r2, *OGC® Moving Features Encoding Extension: Simple Comma Separated Values (CSV)*. Open Geospatial Consortium (2015). https://docs.ogc.org/is/14-084r2/14-084r2.html.

[3] Akinori Asahara, Ryosuke Shibasaki, Nobuhiro Ishimaru, David Burggraf: OGC 14-083r2, *OGC® Moving Features Encoding Part I: XML Core*. Open Geospatial Consortium (2015). https://docs.ogc.org/is/14-083r2/14-083r2.html.

[4] ISO: Geographic information — Rules for application schema, 2013. https://www.iso.org/standard/59193.html

[5] OGC: OGC 22-036: 3D+ Standards Framework Engineering Report, 2022. https://portal.ogc.org/files/?artifact_id=103466

[6] Dawson & Woods: ITRF to GDA94 coordinate transformations, 2010. https://www.ga.gov.au/__data/assets/pdf_file/0008/61100/ITRF-to-GDA94-coordinate-transformations.pdf

[7] IOGP: EPSG Geodetic Parameter Dataset. https://epsg.org/

[8] Roger Lott: GGXF: Gridded Geodetic data eXchange Format, 2022. https://github.com/opengeospatial/CRS-Gridded-Geodetic-data-eXchange-Format/

[9] NASA: SPICE: Navigation and ancillary information facility. https://naif.jpl.nasa.gov/naif/

[10] OGC: Common Query Language (CQL2), draft. https://github.com/opengeospatial/ogcapi-features/tree/master/cql2

[11] OGC: Testbed 18 GitHub repository — D025 GML sample files. https://github.com/opengeospatial/testbed_resources/tree/main/Testbed-18/3D_plus/src/main/resources

[12] IOGP: Coordinate Conversions and Transformations including Formulas, 2022. https://epsg.org/guidance-notes.html