Proposal to Consolidate Geographic Coordinates of Location using WKT

# Motivation

The IEC 61970-301 and IEC 61968-11 CIM standards currently represent geographic locations in a fragmented way. A location’s geometry is defined by multiple **PositionPoint** objects (each holding X, Y, Z coordinates) referencing Location, which is referencing separate **CoordinateReferenceSystem** object. The current approach is comparable to representing a floating-point number by separately storing its sign, exponent, and mantissa – while functional, it is unnecessarily complicated and because of it you cannot parse and store it in one operation, but you need to read the document until you have all parts. A more unified representation can improve deserialization and integration with GIS tools.

## Problem Statement

* Currently information on geometry of an element is fragmented across three classes:
  + *CoordinateReferenceSystem* – defines coordinate system (via a URN like an EPSG code)
  + *Location* – carrier of the geometry, points to *CoordinateReferenceSystem*
  + *PositionPoint* – multiple objects defining a geometry pointing to Location.

A screenshot of a computer

AI-generated content may be incorrect.

* **Difficult Geometry Reconstruction:** Since geometry of an element is usually stored as single entity, consumers of the data must load the whole document and assemble the *Location*’s geometry by collecting all *PositionPoint* instances pointing to *Location* (using their *sequenceNumber* and *groupNumber* to sort them) to be able to store them as geometries of *Location*
* **Difficult Geometry Reconstruction of Difference:**  Usually *CoordinateReferenceSystem* does not change so it is not included in the difference document. To store geometry *CoordinateReferenceSystem* needs to be retrieved from old versions which complicates the deserialization.
* **Atomic property of geometry:** Any change to a *Location*’s shape usually means manipulating the complete set of *PositionPoint*s. To insert or remove a point, one must replace and re-order the whole collection.
* **Integration Challenges with GIS:** The CIM’s current structure does not map cleanly to common GIS data formats. Most applications that manipulate geometries like GIS and spatial databases refer to geometries as single string or a binary blob, not as lists of point components. Converting a CIM Location into a format like GeoJSON or WKT requires custom code to assemble the coordinates array. Likewise, serializing geometries to CIM requires splitting a drawn shape into PositionPoint objects.
* **Serialization Inefficiencies:** In serialization (XML, JSON, RDF), the current model is verbose and is ill-fitted for serialization. A simple polyline might result in dozens of XML elements (one per point) plus a coordinate system element. This increases message size and processing cannot be seamlessly done using xml streaming libraries (what if CoordinateReferenceSystem is at the end of the document?).

# Proposed Change

We propose introducing a new attribute **Location.wkt** of new CIMDataType AsWKT to hold the geometric description in one attribute.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Location.wkt

This attribute will have the geometry encoded as a **Well-Known Text (WKT)**, defined in in accordance with the OGC Simple Feature Specification (06-103r4) chapter 7, usually referred to as WKT1. By combining coordinates into one WKT string, the geometry can be treated as a single attribute of *Location*, rather than an external collection of points.

## Coordinate Reference System

For exchange it is proposed to use only **WGS84** CRS. There are many advantages to using **WGS84** CRS because error in conversion from usual ellipsoid-based coordinate reference systems to WGS84 and back is less than 1 cm and with advanced calculus could be less than 2 mm, so we can conclude that information is not lost by using WGS84.

For local/internal use when local Coordinate Reference System is needed it is possible to create WKT CIMDataType as being a compound of two strings value and crs.

So two proposals are how to deal with crs, one is to have it always fixed like in GeoJSON specification to WGS84 or to allow custom/local crs.

## Converter

A python-based converter is supplied in the Appendix for conversion from old GeographicalLayout to new GeographicalLayout format with wkt as string (not compound) and back.

## GeograpicalLocation Profile

A screenshot of a computer

AI-generated content may be incorrect.

# Benefits

* Streaming deserialization (no need to read whole document to store geometry)
* Easier serialization (WKT support is available as a program library)
* Relies on established standard (WKT) which have wide support
* Atomic operations for difference, single attribute is always changed whole
* Compatibility with GIS and Spatial tools
* Less verbose messages (GeographicLayout file is around 60% smaller)
* This modelling change is backward compatible, PositionPoint and CoordinateReferenceSystem could be deprecated

# Example IEC 61970-552 serialization

To illustrate the change in IEC 61968-552 serialization of GeographicLayout profile, consider a simple example of a Location that represents a line with four points.

## **Current CIM17 structure (Location with CoordinateReferenceSystem and PositionPoints)**

<cim:CoordinateSystem rdf:ID=**"\_ce8fd808-5bfb-4c99-a3ce-d9f7a9ca16df"**>

<cim:IdentifiedObject.name> **Lambert 2008**</cim:IdentifiedObject.name>

<cim:CoordinateSystem.crsUrn>**urn:ogc:def:crs:EPSG::3812**</cim:CoordinateSystem.crsUrn>

<cim:IdentifiedObject.mRID>**ce8fd808-5bfb-4c99-a3ce-d9f7a9ca16df**</cim:IdentifiedObject.mRID>

</cim:CoordinateSystem>

<cim:Location rdf:ID=**"\_05e9859c-01b9-12a7-3320-39fb7da2af1e"**>

<cim:IdentifiedObject.name>**BE-Line\_3**</cim:IdentifiedObject.name>

<cim:Location.CoordinateSystem rdf:resource=**"#\_ce8fd808-5bfb-4c99-a3ce-d9f7a9ca16df"** />

<cim:Location.PowerSystemResources rdf:resource=**"#\_78736387-5f60-4832-b3fe-d50daf81b0a6"** />

<cim:IdentifiedObject.mRID>**05e9859c-01b9-12a7-3320-39fb7da2af1e**</cim:IdentifiedObject.mRID>

</cim:Location>

<cim:PositionPoint rdf:ID=**"\_173034c8-dee1-4705-b41f-4e67c2472360"**>

<cim:PositionPoint.sequenceNumber>**1**</cim:PositionPoint.sequenceNumber>

<cim:PositionPoint.xPosition>**649320.61**</cim:PositionPoint.xPosition>

<cim:PositionPoint.yPosition>**195879.23**</cim:PositionPoint.yPosition>

<cim:PositionPoint.Location rdf:resource=**"#\_05e9859c-01b9-12a7-3320-39fb7da2af1e"** />

</cim:PositionPoint>

<cim:PositionPoint rdf:ID=**"\_f35c229f-84dd-4f11-821a-536c2b975bc8"**>

<cim:PositionPoint.sequenceNumber>**2**</cim:PositionPoint.sequenceNumber>

<cim:PositionPoint.xPosition>**675536.77**</cim:PositionPoint.xPosition>

<cim:PositionPoint.yPosition>**200882.59**</cim:PositionPoint.yPosition>

<cim:PositionPoint.Location rdf:resource=**"#\_05e9859c-01b9-12a7-3320-39fb7da2af1e"** />

</cim:PositionPoint>

<cim:PositionPoint rdf:ID=**"\_4f245aa8-98ac-4ee0-a26b-04660c3d1b39"**>

<cim:PositionPoint.sequenceNumber>**3**</cim:PositionPoint.sequenceNumber>

<cim:PositionPoint.xPosition>**707442.53**</cim:PositionPoint.xPosition>

<cim:PositionPoint.yPosition>**244187.89**</cim:PositionPoint.yPosition>

<cim:PositionPoint.Location rdf:resource=**"#\_05e9859c-01b9-12a7-3320-39fb7da2af1e"** />

</cim:PositionPoint>

<cim:PositionPoint rdf:ID=**"\_ff7e39bc-0c8f-46e5-80b5-60a0b8aa9349"**>

<cim:PositionPoint.sequenceNumber>**4**</cim:PositionPoint.sequenceNumber>

<cim:PositionPoint.xPosition>**705334.79**</cim:PositionPoint.xPosition>

<cim:PositionPoint.yPosition>**306261.47**</cim:PositionPoint.yPosition>

<cim:PositionPoint.Location rdf:resource=**"#\_05e9859c-01b9-12a7-3320-39fb7da2af1e"** />

</cim:PositionPoint>

In this snippet, the Location references a Lambert 2008 (urn:ogc:def:crs:EPSG::3812) coordinate system, and 4 PositionPoint entries must be collected and sorted to yield the line geometry.

## **Proposed WKT CIM structure (Location with wkt attribute)**

Here, the entire line geometry is expressed as a WKT string. The coordinates correspond to the same four points as before but in WGS84. The coordinate system element is no longer needed explicitly – WGS84 is assumed.

<cim:Location rdf:ID=**"\_05e9859c-01b9-12a7-3320-39fb7da2af1e"**>

<cim:IdentifiedObject.name>**BE-Line\_3**</cim:IdentifiedObject.name>

<cim:Location.PowerSystemResources rdf:resource=**"#\_78736387-5f60-4832-b3fe-d50daf81b0a6"**/>

<cim:IdentifiedObject.mRID>**05e9859c-01b9-12a7-3320-39fb7da2af1e**</cim:IdentifiedObject.mRID>

<cim:Location.wkt>**LINESTRING (4.30173 50.803, 4.69046 50.8994, 5.08825 51.3026, 5.07294 51.8765)**</cim:Location.wkt>

</cim:Location>

# Example IEC 61968-100 Payload

To illustrate the change in IEC 61968-100 payload of WorkRequests profile, consider a simple example of a WorkLocation that represents a line with four points.

## **Current CIM17 structure (Location with CoordinateReferenceSystem and PositionPoints)**

<m:WorkRequests xmlns:m="http://iec.ch/TC57/2015/WorkRequests#"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://iec.ch/TC57/2015/WorkRequests# WorkRequests.xsd">

    <m:WorkRequest>

        <m:mRID>4c708a06-4bda-441b-a9aa-5a1d7dbf1460</m:mRID>

        <m:Work>

            <m:mRID>bcd88f60-3ae3-44dd-8f42-bf0ea5dd0eb8</m:mRID>

            <m:kind>other</m:kind>

            <m:statusKind>approved</m:statusKind>

            <m:WorkLocation>

                <m:mRID>05e9859c-01b9-12a7-3320-39fb7da2af1e</m:mRID>

                <m:CoordinateSystem>

                    <m:crsUrn>urn:ogc:def:crs:EPSG::3812</m:crsUrn>

                </m:CoordinateSystem>

                <m:PositionPoints>

                    <m:sequenceNumber>1</m:sequenceNumber>

                    <m:xPosition>649320.61</m:xPosition>

                    <m:yPosition>195879.23</m:yPosition>

                </m:PositionPoints>

                <m:PositionPoints>

                    <m:sequenceNumber>2</m:sequenceNumber>

                    <m:xPosition>675536.77</m:xPosition>

                    <m:yPosition>200882.59</m:yPosition>

                </m:PositionPoints>

                <m:PositionPoints>

                    <m:sequenceNumber>3</m:sequenceNumber>

                    <m:xPosition>707442.53</m:xPosition>

                    <m:yPosition>244187.89</m:yPosition>

                </m:PositionPoints>

                <m:PositionPoints>

                    <m:sequenceNumber>4</m:sequenceNumber>

                    <m:xPosition>705334.79</m:xPosition>

                    <m:yPosition>306261.47</m:yPosition>

                </m:PositionPoints>

            </m:WorkLocation>

        </m:Work>

    </m:WorkRequest>

</m:WorkRequests>

## **Proposed WKT CIM structure (Location with wkt attribute):**

Here, the entire line geometry is expressed as a WKT string. The coordinates correspond to the same four points as before but in WGS84. The coordinate system element is no longer needed explicitly – WGS84 is assumed.

<m:WorkRequests xmlns:m="http://iec.ch/TC57/2015/WorkRequests#"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://iec.ch/TC57/20xx/WorkRequests# WorkRequests.xsd">

    <m:WorkRequest>

        <m:mRID>4c708a06-4bda-441b-a9aa-5a1d7dbf1460</m:mRID>

        <m:Work>

            <m:mRID>bcd88f60-3ae3-44dd-8f42-bf0ea5dd0eb8</m:mRID>

            <m:kind>other</m:kind>

            <m:statusKind>approved</m:statusKind>

            <m:WorkLocation>

                <m:mRID>05e9859c-01b9-12a7-3320-39fb7da2af1e</m:mRID>

                <m:wkt>LINESTRING (4.30173 50.803, 4.69046 50.8994, 5.08825 51.3026, 5.07294 51.8765)</m:wkt>

            </m:WorkLocation>

        </m:Work>

    </m:WorkRequest>

</m:WorkRequests>

# Potential Problems Considered

## WKT could be large

WKT strings could get large for certain element like edges of an area of GeographicalRegion which at resolution of 100m could grow to 50KB of text. Some databases would need to have their column datatypes adjusted to handle these lengths of text. Most of elements like transmission lines or cables should not exceed 2000 points which would be less than 4KB of text.

## WGS84 Conversion Errors

Local systems like ETRS89/UTM and other local systems like HTRS96/TM based on ETRS89/UTM which in turn is tied to the same reference ellipsoid as WGS84 can be converted without much loss.

Example: converting using GDAL library from HTRS96/TM(EPSG:3765) to WGS84 and back yields an error of 1 cm (500000.00, 5000000.00 -> 15.000001°, 45.123456° -> 500000.01, 5000000.01)

# Appendix A Convert from PositionPoints to wkt/geojson and back for IEC 61970-552

Use this python program with arguments file, format (wkt, cim17), target\_epsg (4326 for WGS84):

import sys

from lxml import etree

from shapely.geometry import LineString, MultiLineString, Polygon, MultiPolygon, mapping, shape

from shapely import wkt

from pyproj import Transformer

import geojson

import uuid

def parse\_cim(file, format, target\_epsg='4326'):

    tree = etree.parse(file)

    root = tree.getroot()

    ns = root.nsmap

    if format == 'cim17':

        convert\_to\_cim17(root, ns, target\_epsg)

    else:

        convert\_from\_cim(root, ns, format, '4326')

    return tree

def convert\_from\_cim(root, ns, format, target\_epsg):

    crs\_dict = {}

    for cs in root.findall('.//cim:CoordinateSystem', ns):

        cs\_id = cs.attrib[f'{{{ns["rdf"]}}}ID']

        crs\_urn = cs.findtext('cim:CoordinateSystem.crsUrn', namespaces=ns)

        epsg\_code = crs\_urn.split('::')[-1]

        crs\_dict[cs\_id] = epsg\_code

    for loc in root.findall('.//cim:Location', ns):

        loc\_id = loc.attrib[f'{{{ns["rdf"]}}}ID']

        crs\_ref = loc.find('cim:Location.CoordinateSystem', ns).attrib[f'{{{ns["rdf"]}}}resource'][1:]

        epsg\_from = crs\_dict[crs\_ref]

        transformer = Transformer.from\_crs(f"EPSG:{epsg\_from}", f"EPSG:{target\_epsg}", always\_xy=True)

        points = {}

        for pp in root.xpath(f'.//cim:PositionPoint[cim:PositionPoint.Location[@rdf:resource="#{loc\_id}"]]', namespaces=ns):

            group = int(pp.findtext('cim:PositionPoint.groupNumber', default='1', namespaces=ns))

            seq = int(pp.findtext('cim:PositionPoint.sequenceNumber', namespaces=ns))

            x = float(pp.findtext('cim:PositionPoint.xPosition', namespaces=ns))

            y = float(pp.findtext('cim:PositionPoint.yPosition', namespaces=ns))

            z\_text = pp.findtext('cim:PositionPoint.zPosition', namespaces=ns)

            if z\_text is not None:

                z = float(z\_text)

                coord = transformer.transform(x, y, z)

            else:

                coord = transformer.transform(x, y)

            points.setdefault(group, []).append((seq, coord))

        geometries = []

        for group in sorted(points.keys()):

            coords = [pt for seq, pt in sorted(points[group])]

            if coords[0] == coords[-1]:

                geometries.append(Polygon(coords))

            else:

                geometries.append(LineString(coords))

        if all(isinstance(g, Polygon) for g in geometries):

            geom = geometries[0] if len(geometries) == 1 else MultiPolygon(geometries)

        elif all(isinstance(g, LineString) for g in geometries):

            geom = geometries[0] if len(geometries) == 1 else MultiLineString(geometries)

        else:

            geom = MultiLineString([g.exterior if isinstance(g, Polygon) else g for g in geometries])

        if format == 'wkt':

            wkt\_elem = etree.SubElement(loc, f'{{{ns["cim"]}}}Location.wkt')

            wkt\_elem.text = geom.wkt

        elif format == 'geojson':

            geojson\_elem = etree.SubElement(loc, f'{{{ns["cim"]}}}Location.geojson')

            geojson\_elem.text = geojson.dumps(mapping(geom))

    for elem in root.xpath('.//cim:PositionPoint | .//cim:CoordinateSystem', namespaces=ns):

        elem.getparent().remove(elem)

def convert\_to\_cim17(root, ns, target\_epsg):

    crs\_id = "crs\_" + target\_epsg

    crs\_elem = etree.SubElement(root, f'{{{ns["cim"]}}}CoordinateSystem', attrib={f'{{{ns["rdf"]}}}ID': crs\_id})

    etree.SubElement(crs\_elem, f'{{{ns["cim"]}}}CoordinateSystem.crsUrn').text = f"urn:ogc:def:crs:EPSG::{target\_epsg}"

    for loc in root.findall('.//cim:Location', ns):

        loc\_id = loc.attrib[f'{{{ns["rdf"]}}}ID']

        for geom\_elem\_name in ['Location.wkt', 'Location.geojson']:

            geom\_elem = loc.find(f'cim:{geom\_elem\_name}', ns)

            if geom\_elem is not None:

                geom = wkt.loads(geom\_elem.text) if geom\_elem\_name.endswith('Location.wkt') else shape(geojson.loads(geom\_elem.text))

                loc.remove(geom\_elem)

                group\_number = 1

                geometries = geom.geoms if hasattr(geom, 'geoms') else [geom]

                for g in geometries:

                    coords = list(g.exterior.coords) if isinstance(g, Polygon) else list(g.coords)

                    for seq, coord in enumerate(coords, start=1):

                        pp\_elem = etree.SubElement(root, f'{{{ns["cim"]}}}PositionPoint', attrib={f'{{{ns["rdf"]}}}ID': f'\_{uuid.uuid4()}'})

                        etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.groupNumber').text = str(group\_number)

                        etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.sequenceNumber').text = str(seq)

                        etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.xPosition').text = str(coord[0])

                        etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.yPosition').text = str(coord[1])

                        if len(coord) == 3:

                            etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.zPosition').text = str(coord[2])

                        etree.SubElement(pp\_elem, f'{{{ns["cim"]}}}PositionPoint.Location', attrib={f'{{{ns["rdf"]}}}resource': f'#{loc\_id}'})

                    group\_number += 1

if \_\_name\_\_ == "\_\_main\_\_":

    cim\_file = sys.argv[1]

    format = sys.argv[2]

    tree = parse\_cim(cim\_file, format=format)

    output\_file = cim\_file.replace('.xml', f'\_{format}.xml')

    etree.indent(tree, space="  ")

    tree.write(output\_file, pretty\_print=True, xml\_declaration=True, encoding='UTF-8')

    print(f"File written to {output\_file}")

These cases are covered:

* Multiple Coordinate Reference Systems (different EPSG codes)
* Grouped and Disjointed Points
* 3D Coordinates (zPosition)
* Polygon vs Lines