Geography Markup Language (GML) v1.0

OGC Document Number: 00-029

Date: 12-May-2000

This version: **1.0**
Latest version: **1.0**
Previous version: **this is the first public release**

Editors: Ron Lake, Galdos Systems Inc. <rlake@galdosinc.com>
Adrian Cuthbert, Laser-Scan Ltd. <adrian@lsl.co.uk>

Authors: Adrian Cuthbert, Laser-Scan Ltd. <adrian@lsl.co.uk>
Barry O'Rourke, Compusult Ltd. <barry@compusult.nf.ca>
Edric Keighan, Cubewerx Inc. <ekeighan@cubewerx.com>
Serge Margoulies, IONIC Software s.a. <serge.margoulies@ionicsoft.com>
Jayant Sharma, Oracle Corporation <jsharma@us.oracle.com>
Paul Daisey, U.S. Census Bureau <pdaisey@geo.census.gov>
Ron Lake, Galdos Systems Inc. <rlake@galdosinc.com>
Sandra Johnson, MapInfo Ltd. <sandra_johnson@mapinfo.com>

Abstract

The Geography Markup Language (GML) is an XML encoding for the transport and storage of geographic information, including both the geometry and properties of geographic features. This specification defines the mechanisms and syntax that GML uses to encode geographic information in XML. It is anticipated that GML will make a significant impact on the ability of organizations to share geographic information with one another, and to enable linked geographic datasets. The initial release of this specification is concerned with the XML encoding of what the OpenGIS® Consortium (OCG) calls 'Simple Features'.

Status of this document

This document is an OpenGIS® Consortium Recommendation Paper. It is similar to a *proposed* recommendation in other organizations. While it reflects a public statement of the official view of the OGC, it does not have the status of a OGC Technology Specification. It is anticipated that the position stated in this document will develop in response to changes in the underlying technology. Although changes to this document are governed by a comprehensive review procedure, it is expected that some of these changes

may be significant.

The OGC explicitly invites comments on this document. Please send them to gml.rfc@opengis.org

Available formats

This GML specification is available in the following formats.

- on-line (HTML)
- as a zip file
- as a PDF file

In case of a discrepancy between the various forms of the specification, the on-line version is considered the definitive version.
These links will be made live once the document is adopted and being placed on the OGC site.

Available languages

The English version of this specification is the only normative version.

Table of Contents

1. An Introduction to Geographic Features

1.1. Overview

This section provides an introduction to the key concepts required to understand how Geography Markup Language (GML) models the world. It is based on the OpenGIS® Abstract Specification (go to http://www.opengis.org/ and then follow the link to *OpenGIS Specifications* and look for *OpenGIS Abstract Specification*) which defines a geographic feature as:

> *"A feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth."*

Thus a digital representation of the real world can be thought of as a set of **features**. The state of a feature is defined by a set of **properties**, where each property can be thought of as a {name, type, value} triple. The number of properties a feature may have, together with their names and types, are determined by its **feature type**. Geographic features are those with properties whose values may be a **geometry**. A **feature collection** is a collection of features that can itself be regarded as a feature. Consequently a feature collection has a feature type and thus may have properties of its own, in addition to the features it contains.

This definition of GML is concerned with what the OpenGIS Consortium (OGC) calls 'simple features'. These are features whose geometry properties are restricted to holding 'simple geometry' (for example, coordinates are defined in two dimensions and the path of a curve between coordinates is assumed to be interpolated linearly). The term 'simple features' was originally coined to describe the functionality defined in a set of OpenGIS® Implementation Specifications (go to http://www.opengis.org/ and then follow the link to *OpenGIS Specifications* and look for *OpenGIS Implementation Specifications*).

GML follows the geometry model defined in those specifications. For example, the traditional 0, 1 and 2-dimensional geometries defined in a two-dimensional **spatial reference system** (SRS) are represented by **points**, **line strings** and **polygons**. In addition the geometry model for simple features also allows geometries that are collections of other geometries (either homogeneous, **multi point**, **multi line string** and **multi polygon**, or heterogeneous, **geometry collection**). In all cases the 'top-most' geometry is responsible for indicating in which SRS the measurements have been made.

Consider the example of somebody wishing to build a digital representation of the city of Cambridge in England. This could be represented as a feature collection where the individual features represent such things as rivers, roads and colleges. This classification of real world phenomena determines the feature types that need to be defined. The choice of classification is related to the task to which the digital representation will ultimately be put.

The 'River' feature type might have a property called 'name' whose value should be of the type 'string'. It is common to refer to the typed property. Thus, in the previous example, the 'River' feature type is said to have a *string property* called 'name'. Similarly the 'Road' feature type might have a string property called 'classification' and an *integer property* called 'number'. Properties with simple types (integers, strings, reals, booleans) are collectively referred to as *simple properties*.

The features required to model Cambridge might have *geometry properties* as well as simple properties. Just like other properties, geometry properties must be named. So the 'River' feature type might have a geometry property called 'centerLineOf' and the 'Road' feature type might have a geometry property called 'linearGeometry'. It is possible to be more precise about the type of geometry that can be used as a property value. Thus in the 'River' and 'Road' examples the geometry property could be specialised to be a *line string*

property. Just as it is common to have multiple simple properties defined on a single feature type (for example, the 'College' feature type might have integer properties 'numberOfUndergraduates' and 'numberOfPostgraduates'), so too a feature type may have multiple geometry properties.

Finally the entire model of Cambridge can be expressed as a single feature collection. This feature collection might have a feature type of 'CityModel' which is interpreted ro mean it has a string property called 'modelDate', giving the date when it was constructed, and a geometry property called 'boundedBy' giving the extent over which the model is valid.

1.2 Examples

This document makes use of a simple example to demonstrate how GML can be used to encode information about the real world. This example is based on the Cambridge model described above, and shall be referred to as the 'Cambridge example'. A more precise definition is given below:

> *The Cambridge example has a single feature collection of type 'CityModel' and contains two features using a containment relationship called 'modelMember'. The feature collection has a string property called 'modelDate' with the value 'Feb 2000' and a geometry property called 'boundedBy' with a Box value. The Box geometry is expressed in the SRS identified by the name 'EPSG:4326'. It represents the 'bounding box' of the feature collection.*
>
> *The first of the features is of type 'River' with the name 'Cam' and description 'The river that runs through Cambridge'. It has a geometric property called 'centerLineOf' with a LineString value. This LineString geometry is expressed in the same SRS as used by the bounding box.*
>
> *The second of the features is of type 'Road' with description 'M11'. It has a string property called 'classification' with value 'motorway' and an integer property called 'number' with value '11'. It has a geometric property called 'linearGeometry' with a LineString value. This LineString geometry is also expressed in the same SRS as used by the bounding box.*

In the example the first feature uses only 'standard' property names defined by GML, whereas the second feature uses application specific property names. Thus this example will demonstrate how GML is capable of being used by any application specific model. The example is not designed to provide examples of how the various types of geometry are encoded.

We introduce a second example to illustrate how GML can be used to encode a hierarchy of feature collections. This will be referred to as the 'Schools example'.

The Schools example has a root feature collection of type 'State' that contains two features collections of type 'SchoolDistrict' using the containment relationship 'featureMember'. Each of the 'SchoolDistrict' feature collections contains two features from the type 'School' or 'College' using the containment relationship 'districtMember'.

The 'District' feature type has a string property called 'districtName' and a polygon property called 'extentOf'.

The 'School' feature type has a string property called 'principalName' and a point property called 'location'.

The 'College' feature type has a string property called 'prinicpalName' and a point property called 'pointProperty'.

1.3. Object Models

The Feature Model used by the OpenGIS Consortium is shown in Figure 1.

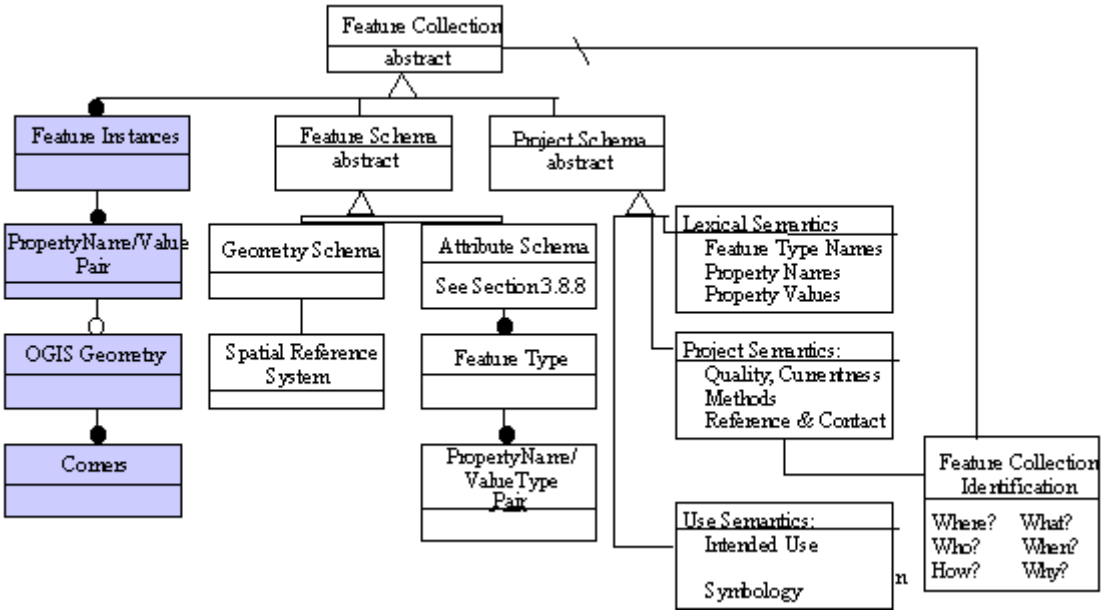


Figure 1. The OGC Feature Model

It is common practice in the Geospatial Information (GI) community to refer to the properties of a feature as attributes. However, for the purposes of avoiding confusion with attributes in XML, this document refers to them as properties.

The 'Simple Features' model represents a simplification of the more general model

described in the OpenGIS Abstract Specification. This simplification was the result of developing a number of implementation specifications. There are two major simplifications:

- Features are assumed to have either simple properties (booleans, integers, reals, strings) or geometric properties.
- Geometries are assumed to be defined in a two-dimensional SRS and use linear interopolation between coordinates.

There are a number of consequences that follow from these simplifications; for example simple features only provide support for 'vector' data, nor are simple features sufficiently expressive to model topology explicitly. It is intended to redress some of these limitations in future versions of GML.

The 'simplified' geometry model is central to a number of specifications and documents. Consequently it is available as a separate document at http://www.opengis.org/geometry.html. **The 'simplified' geometry document should be read in conjunction with this document.**

The 'Simple Features Geometry' document is being worked on in parallel and the link will be fixed for final release (probably after adoption).

2. GML Overview

2.1. GML Profiles

This section discusses the approach to the encoding of OGC Simple Features in XML. While this version of GML is concerned only with the XML encoding of OGC Simple Features, future versions of the GML Specification will deal with more elaborate OGC geometry models.

It is anticipated that GML will appeal to a broad class of users who will in turn wish to employ a variety of XML technologies. GML is thus presented in the form of three profiles as follows:

- **Profile 1:** for those who wish to use a pure DTD based solution and are not prepared to develop application specific DTD's, or wish data to be returned against a fixed set of DTD's. This profile requires the use of GML Feature, and GML Geometry DTD's.
- **Profile 2:** for those who wish to use a pure DTD based solution but are prepared to develop their own application specific DTD's, or are prepared to accept data encoded with a referenced DTD. This profile requires the user to create an application specific Feature DTD that uses the GML Geometry DTD.
- **Profile 3:** for those who are prepared to make use of RDF and RDF Schema. These users will typically require stronger control of the geospatial typing framework (e.g. they must be able to relate a type name to an actual schema

definition). This profile requires the user to create an application specific RDF Schema definition that uses the GML RDF Schema definition. Alternatively Profile 3 users may employ DTD's which are derived in some fashion from an RDF Schema or which can trace their elements to types defined in an associated RDF Schema.

The three profiles are summarized in Figure 2.

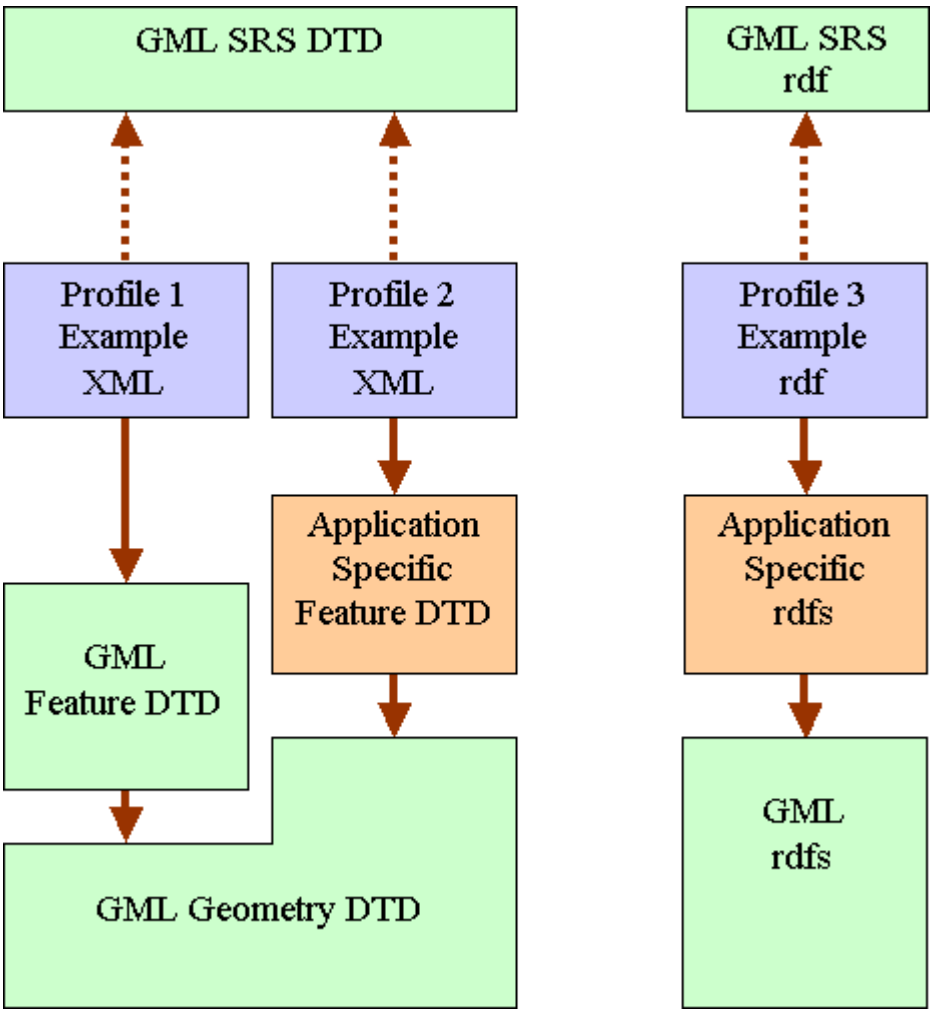


Figure 2. The three profiles of GML

The arrows indicate a reference; the dotted arrows indicate an optional reference. The light green boxes are pre-defined GML definitions. The light brown boxes represent application specific definitions built according to the appropriate GML profile rules. Finally the light purple boxes represent geospatial information encoded using the appropriate GML profile.

GML is currently XML V1.0 compliant and for this reason uses Document Type Definitions (DTD's) rather than XML Schemas. When the W3C's XML Schema Structures [XML SCHEMA] and Data Types [XML DATATYPE] specifications have reached Recommendation status, it is expected that this specification will be modified to include use of XML Schemas.

GML has also been developed so as to be consistent with the W3C Resource Description Format (RDF) Model and Syntax. GML geometry encoding can be used to describe the geometric properties of any RDF resource such as its extent, coverage or location. This enables GML to be used in a wide variety of applications that are not inherently spatial in nature.

GML has also been developed to be consistent with the XML Namespaces Recommendation ([XMLNAME]) . In GML Profile 2 and 3, XML Namespaces can be used to distinguish the definitions of geographic features and properties defined in application-specific domains from one another and from those defined by OGC GML.

2.2 Properties and Classes in GML

GML is an XML encoding for geographic features. In order to correctly interpret a GML data file it is necessary to understand the conceptual model that underlies GML. A geographic feature in the OGC Abstract Specification is essentially a named list of properties. Thus we can consider a property as a function that maps a feature onto a property value. A property is characterised by the input feature type and the type of the value that is returned.

For example, if the feature type **House** has a **String** property called **address** then we might write:

address(House) --> String

If, in addition, the **House** feature type has a **Polygon** property called **extentOf** then we could write:

extentOf(House) --> Polygon

More generally we might regard all the possible types of feature, together with all types of property value (Strings, Integers, Polygons etc), as a set of *classes*. Then we can characterise a property as a function with a *domain* (input) class and a *range* (output) class.

We are not restricted to talking about features and their properties, we can also talk about the properties of a geometry, since geometry defines a class. Consider a geometry as a named list of properties, then the **Polygon** class might have an **outerBoundaryIs** property so that one could write:

outerBoundaryIs(Polygon) --> LinearRing

We are then able to compose two functions to obtain:

outerBoundaryIs(extentOf(House)) --> LinearRing

This approach can also be applied to items bigger than features. For example, a **FeatureCollection** can be considered to have multiple named properties (albeit all with

the same name) that have as their values the **Features** in the collection. Thus we can write:

featureMember(FeatureCollection) --> Feature

This forms the theoretical basis for GML. These ideas are stated more formally in the W3C's Resource Description Format Schema (RDF Schema), which GML Profile 3 uses directly. However GML Profiles 1 and 2 can be used without any further consideration of RDF.

When we write GML tags we will distinguish between properties and classes. Tags that represent instances of GML classes will start with an uppercase letter (e.g. Polygon) while tags that represent properties will start with a lowercase letter which subsequent embedded words starting with uppercase letters (e.g. extentOf).

2.3. Geography and Graphics

Simple Features are intended to describe the geography of entities in the real world. As such, the encoding is not concerned with the visualization of geographic features as in the drawing of maps. To draw a map with GML it is necessary to transform the GML into a graphic format, either by direct rendering, or preferably by transforming the XML encoded Simple Features into XML encoded graphics elements such as SVG (Scalable Vector Graphics) [SVG], VML (Vector Markup Language) [VML], or Virtual Reality Markup Language [VRML]. Such a transformation can be done anywhere in the processing chain between the data store and the visualization device.

GML can be considered in relation to POIX [POIX] GML is intended to model the structure and relationships for real world geography. Although not connected to GML, POIX is a much simplified model for position and direction information. POIX data such as might be required in a Portable Digital Assistant (PDA) can be generated from GML data.

GML encoding is intended to support both data storage and data transport. Implementors may decide to store geographic information in GML, or they may decide to convert from some other storage format on demand and use GML only for data transport.

GML is distinct from, and not dependent on any other graphical specification. GML contains no information about how the features it encodes might appear. Yet the visual rendering of a GML structure is dependent on the use of one of several possible vector graphics formats. Transforming GML into SVG (Scalable Vector Graphics), VML (Vector Graphics Markup Language), or VRML (Virtual Reality Markup Language) is strongly recommended for data visualization.

Many different graphical symbolic representations might be generated from a single GML file. These different representations could include both different graphical formats and different symbolizations. A single GML file might thus give rise to multiple types of maps.

In some applications there will be no graphical data display at all. Geographic data might

be simply be routed to a numerical model (e.g. a flood prediction model) for processing. The output of this numerical model may also be expressed in GML.

Coordinates of points in a GML-encoded structure are specified relative to a named Spatial Reference System whose description can also encoded in GML. A data server can supply data encoded in GML but not supply the description of the Spatial Reference System, provided that a named reference to such a description is included. Spatial Reference System descriptions are thus always connected to the geographic data by means of a named reference.

3. Geometry

3.1. Overview

This section describes how GML encodes Geometry into XML. It also introduces the GML Geometry DTD that supports this encoding. This is used explicitly by GML Profiles 1 and 2 (the complete GML Geometry DTD is given in Appendix A). However the XML encoding is also consistent with the RDF Schema definition of Geometry used by GML Profile 3. **Consequently the material in this section should be read by all prospective GML users.**

Conforming to the OGC Simple Features model, GML provides geometry elements corresponding to the following Geometry Classes.

- Point
- LineString
- LinearRing
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection

In addition it provides a coordinates element for encoding coordinates, and a Box element for defining extents. The following sections describe in detail the encoding of each of these types of geometries.

3.2. coordinates Element

A coordinate list is a simple list of coordinate tuples. The separators used to parse the coordinate list are encoded as attributes of the <coordinates> tag. In the example below, the coordinates in a tuple are separated by commas, and the successive tuples in the <coordinates> are separated by whitespace. A coordinate list is not a geometry in the Simple Features sense, merely the coordinate content. All tuples in the string must have the same dimension. A coordinate list is given by the following grammar.

```
<decimal>::='.'

<D>:=[0-9]

<cs>::=","
<ts>::=whitespace (see XML 1.0 [XML]
<coordinate>::='-'<D>+(<decimal><D>+)?
<ctuple>::=<ctuple>|<coordinate><cs><ctuple>
<coordinatelist>::=<coordinatelist>|<ctuple><ts><coordinatelist>

Note that the value of decimal, cs, and ts are
determined by the GML encoding of <coordinates>.
The grammar is illustrated for default values
of decimal, cs and ts.
```

To find the coordinates of any Geometry class instance we introduce the coordinate property. We think of this as a function on the Geometry class instance that returns the coordinates as a coordinate list. The coordinate property has the associated DTD fragment:

```
<!ELEMENT coordinates (#PCDATA) >
<!ATTLIST coordinates
   decimal CDATA #IMPLIED
   cs      CDATA #IMPLIED
   ts      CDATA #IMPLIED>
```

Note that the coordinate value is given by <coordinate>='-'<D>+(<decimal><D>+)?, hence we can encode coordinates as 1.45 or 1,45 etc. depending on the values assigned to the <coordinates> attributes. Note that the default for decimal is '.', for cs is ',' and for ts is whitespace.

Example

```
<coordinates decimal="." cs="," ts="whitespace">
   1.03,2.167 4.167,2.34 4.87,3.0 1.06,2.3
</coordinates>
```

3.3. Point Element

The Point Element is used to encode instances of the Point geometry class. Each Point Element encloses a single coordinates element, the latter containing one and only one coordinate tuple. A Point geometry must specify a SRS in which its coordinates are measured. This is referenced by name. Thus the Point element has an srsName attribute. However this is defined to be optional. This is to allow the Point element to be contained in other elements which might have already specified a SRS. Similar considerations apply to the other geometry elements. The Point element also has an optional ID attribute. The DTD fragment for the Point element is as follows:

```
<!ELEMENT Point (coordinates) >
<!ATTLIST Point
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED>
```

Example

```
<Point srsName="EPSG:4326">
   <coordinates>
      56.1,0.45
   </coordinates>
</Point>
```

3.4. Box Element

The Box Element is used to encode extents. Each Box Element encloses a single coordinates element, the latter containing precisely two coordinate tuples. The first of these is constructued from the minimum values measured along for all the axes, and the second is constructed from the maximum values measured along all the axes. The Box element also has a mandatory srsName, since it cannot be contained by other Geometry classes. It has an optional ID attribute. The DTD fragment for the Box element is as follows:

```
<!ELEMENT Box (coordinates) >
<!ATTLIST Box
   ID      CDATA #IMPLIED
   srsName CDATA #REQUIRED>
```

Example

```
<Box srsName="EPSG:4326">
   <coordinates>
      0.0,0.0 100.0,100.0
   </coordinates>
</Box>
```

3.5. LineString Element

A Line String is a piece-wise linear path. The path is defined by a list of coordinates that are then assumed to be connected by straight line segments. A closed path is indicated by having coincident first and last coordinates. At least two coordinates are required. The DTD fragment is as follows:

```
<!ELEMENT LineString (coordinates) >
<!ATTLIST LineString
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED >
```

Example

```
<LineString srsName="EPSG:4326">
   <coordinates>
      0.0,0.0
      100.0,100.0
   </coordinates>
</LineString>
```

3.6. LinearRing Element

A Linear Ring is a closed, simple piece-wise linear path. The path is defined by a list of coordinates that are then assumed to be connected by straight line segments. The last coordinate must be coincident with the first coordinate. At least four coordinates are

required (the three to define a ring and the fourth duplicated one). Since a LinearRing is used in the construction of Polygons, which define their own SRS, it has no need to define a SRS. The DTD fragment is as follows:

```
<!ELEMENT LinearRing (coordinates) >
<!ATTLIST LinearRing
   ID      CDATA #IMPLIED >
```

Example

```
<LinearRing>
   <coordinates>
      0.0,0.0
      100.0,0.0
      50.0,100.0
      0.0,0.0
   </coordinates>
</LinearRing>
```

3.7. Polygon Element

A Polygon is a connected surface. Any pair of points in the polygon can be connected to one another by a path. The boundary of the Polygon is a set of Linear Rings. We distinguish the outer (exterior) boundary and the inner (interior) boundaries. The Linear Rings of the interior boundary cannot cross one another and cannot be contained within one another. There must be at most one exterior boundary and zero or more interior boundary elements. The ordering of Linear Rings, whether they form clockwise or anti-clockwise paths, is not important. A Polygon is encoded via the DTD fragment:

```
<!ELEMENT Polygon (outerBoundaryIs, innerBoundaryIs*) >
<!ATTLIST Polygon
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED >

<!ELEMENT outerBoundaryIs (LinearRing) >

<!ELEMENT innerBoundaryIs (LinearRing) >
```

Example

```
<Polygon srsName="EPSG:4326">
   <outerBoundaryIs>
      <LinearRing>
         <coordinates>
            0.0,0.0 100.0,0.0 100.0,100.0 0.0,100.0 0.0,0.0
         </coordinates>
      </LinearRing>
   </outerBoundaryIs>
   <innerBoundaryIs>
      <LinearRing>
         <coordinates>
            10.0,10.0 10.0,40.0 40.0,40.0 40.0,10.0 10.0,10.0
         </coordinates>
      </LinearRing>
   </innerBoundaryIs>
   <innerBoundaryIs>
      <LinearRing>
         <coordinates>
```

```
            60.0,60.0 60.0,90.0 90.0,90.0 90.0,60.0 60.0,60.0
         </coordinates>
      </LinearRing>
   </innerBoundaryIs>
</Polygon>
```

3.8. GeometryCollection Element

The GeometryCollection element can be used as a container for arbitrary geometry elements. A GeometryCollection might contain any of the geometry elements such as Points, LineStrings, Polygons, MultiPoints, MultiLineStrings, MultiPolygons and even other GeometryCollections. The GeometryCollection Element has the property geometryMember which returns the next Geometry element in the collection. The geometryMember element can contain any of the GML geometry elements. It should be noted that the srsName attribute can ONLY occur on the outermost GeometryCollection and must not appear as an attribute of any of the enclosed geometry elements. The DTD fragment for the GeometryCollection element is as follows:

```
<!ENTITY % GeometryClasses "(
   Point | LineString | Polygon |
   MultiPoint | MultiLineString | MultiPolygon |
   GeometryCollection )">

<!ELEMENT GeometryCollection (geometryMember)+>
<!ATTLIST GeometryCollection
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED>

<!ELEMENT geometryMember (%GeometryClasses;)>
```

Example

```
<GeometryCollection srsName="EPSG:4326">
   <geometryMember>
      <Point>
         <coordinates>
            50.0,50.0
         </coordinates>
      </Point>
   </geometryMember>
   <geometryMember>
      <LineString>
         <coordinates>
            0.0,0.0 0.0,50.0 100.0,50.0 100.0,100.0
         </coordinates>
      </LineString>
   </geometryMember>
   <geometryMember>
      <Polygon>
         <outerBoundaryIs>
            <LinearRing>
               <coordinates>
                  0.0,0.0 100.0,0.0 50.0,100.0 0.0,0.0
               </coordinates>
            </LinearRing>
         </outerBoundaryIs>
      </Polygon>
   </geometryMember>
</GeometryCollection>
```

3.9. MultiPointElement

A MultiPoint is a collection of Points. It should be noted that the srsName attribute can ONLY occur on the enclosing MultiPoint and must not appear as an attribute of any of the enclosed Points. The DTD fragment for encoding a MultiPoint is as follows:

```
<!ELEMENT MultiPoint (pointMember*) >
<!ATTLIST MultiPoint
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED >

<!ELEMENT pointMember (Point) >
```

Example

```
<MultiPoint srsName="EPSG:4326">
   <pointMember>
      <Point>
         <coordinates>56.1,0.45</coordinates>
      </Point>
   </pointMember>
   <pointMember>
      <Point>
         <coordinates>46.71,9.25</coordinates>
      </Point>
   </pointMember>
   <pointMember>
      <Point>
         <coordinates>56.88,10.44</coordinates>
      </Point>
   </pointMember>
</MultiPoint >
```

3.10. MultiLineString

A MultiLineString is a collection of Line Strings. It should be noted that the srsName attribute can ONLY occur on the enclosing MultiLineString and must not appear as an attribute of any of the enclosed LineStrings. The DTD fragment for MultiLineString is as follows:

```
<!ELEMENT MultiLineString (lineStringMember*) >
<!ATTLIST MultiLineString
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED >

<!ELEMENT lineStringMember (LineString) >
```

Example

```
<MultiLineString srsName="EPSG:4326">
   <lineStringMember>
      <LineString>
         <coordinates>56.1,0.45 67.23,0.67</coordinates>
      </LineString>
   </lineStringMember>
   <lineStringMember>
      <LineString>
```

```
            <coordinates>46.71,9.25 56.88,10.44</coordinates>
         </LineString>
      </lineStringMember>
      <lineStringMember>
         <LineString>
            <coordinates>324.1,219.7 0.45,0.56</coordinates>
         </LineString>
      </lineStringMember>
   </MultiLineString>
```

3.11. MultiPolygon Element

A MultiPolygon is an OGC geometry. It should be noted that the srsName attribute can ONLY occur on the enclosing MultiPolygon and must not appear as an attribute of any of the enclosed Polygons. The GML MultiPolygon is encoded using the following DTD fragment:

```
<!ELEMENT MultiPolygon (polygonMember*) >
<!ATTLIST MultiPolygon
   ID      CDATA #IMPLIED
   srsName CDATA #IMPLIED >

<!ELEMENT polygonMember (Polygon) >
```

Example

```
<MultiPolygon srsName="EPSG:4326">
   <polygonMember>
      <Polygon>
         <outerBoundaryIs>
            <LinearRing>
               <coordinates>
                  0.0,0.0 10.0,0.0 10.0,10.0 0.0,10.0 0.0,0.0
               </coordinates>
            </LinearRing>
         </outerBoundaryIs>
      </Polygon>
   </polygonMember>
   <polygonMember>
      <Polygon>
         <outerBoundaryIs>
            <LinearRing>
               <coordinates>
                  40.0,40.0 50.0,40.0 50.0,50.0 40.0,50.0
40.0,40.0
               </coordinates>
            </LinearRing>
         </outerBoundaryIs>
      </Polygon>
   </polygonMember>
</MultiPolygon>
```


4. Profile 1 - Fixed Feature DTD

4.1 Overview

This section describes the simplest GML Profile. This is defined by three main DTD's, namely:

- GML Feature DTD (gmlfeature.dtd)
- GML Geometry DTD (gmlgeometry.dtd)
- GML Spatial Reference System DTD (ebcsdictionary.dtd)

Using these DTD's one can encode a wide variety of geospatial information. Note that the Geometry DTD (gmlgeometry.dtd) and the Spatial Reference System DTD are shared in common with Profile 2.

This profile is directed at users who do not wish to define their own feature DTD's and who are not going to use RDF (Resource Description Format). For these users, profile 1 provides a standard feature DTD.

4.2. Encoding Geometry

Geometry values are encoded using the GML Geometry DTD introduced in Section 3.

4.3. Encoding Geometry Properties

The GML Geometry DTD not only provides the definition to allow the encoding of Geometry values, it also provides the defintions to encode geometry properties. The encoding of a geographic feature (see next section) relies on these to 'tie' geometry values to a feature. The GML Geometry DTD introduces two geometry properties; **boundedBy** and **geometryProperty**.

The boundedBy element is used to indicate the extent of a geographic feature and maps the Feature class onto the Box class. This is 'standard' name in GML and is used by other profiles. The DTD fragment that defines boundedBy is given below and comes from the GML Geometry DTD:

```
<!ELEMENT boundedBy (Box) >
```

The geometricProperty element is used to give a geometric property to a feature. It includes a mandatory typeName attribute to 'name' the geometricProperty. There are no restrictions on the name of the property, nor does GML Profile 1 endorse any specific names for geometryProperties other than boundedBy. This use of an attribute to name a property is peculiar to GML Profile 1 and substitutes for more generic methods used in other profiles (for example providing the name as an element in an application specific DTD in GML Profile 2). The geometricProperty can contain any geometry class, and a feature can contain any number of geometryProperties. The DTD fragment that defines geometricProperty is given below and comes from the GML Feature DTD:

```
<!ELEMENT geometricProperty (%GeometryClasses;) >
<!ATTLIST geometricProperty
    typeName CDATA #REQUIRED >
```

4.4. Encoding Geographic Features

This section describes the encoding of geographic features using GML Profile 1. The material in this section is unique to Profile 1 and can be omitted by readers who employ Profiles 2 or 3.

A geographic feature in the OGC Abstract Specification is a named list of properties. In GML Profile 1 such a geographic feature is represented by a <Feature> tag that encloses zero or more simple or geometry properties. A simple property is any property that can be encoded using parsed character dta. Currently GML Profile 1 restricts simple properties to booleans, integers, reals and strings. More complex data types need to be encoded using a XML encoding of their own and required an appropriately typed property element. Currently GML Profile 1 only provides support for one type of complex data type, namely Geometry, with the geometricProperty element.

GML encourages the use of 'standard' user-friendly names by pre-defining them (see boundedBy above). GML defines **name** and **description** elements as pre-defined elements to hold string properties. These are used across all profiles and are defined in the GML Geometry DTD by the following fragment:

```
<!ELEMENT name (#PCDATA) >

<!ELEMENT description (#PCDATA) >
```

Including these 'feature metadata' elements in the GML Geometry DTD is a matter of convenience, since GML Profiles 1 and 2 are required to include it.

These concepts are best explained using the Cambridge example we introduced in Section 1. First consider how the two individual features are encoded.

River example

```
<Feature typeName="River">
   <name>
      Cam
   </name>
   <description>
      The river that runs through Cambridge.
   </description>
   <geometricProperty typeName="centerLineOf">
      <LineString srsName="EPSG:4326">
         <coordinates>
             0.0,50.0 100.0,50.0
         </coordinates>
      </LineString>
   </geometricProperty>
</Feature>
```

Road example

```
<Feature typeName="Road">
   <description>
      M11
   </description>
   <property typeName="classification">
      motorway
   </property>
   <property typeName="number" type="integer">
      11
   </property>
   <geometricProperty typeName="linearGeometry">
      <LineString srsName="EPSG:4326">
         <coordinates>
            0.0,100.0 100.0,0.0
         </coordinates>
      </LineString>
   </geometricProperty>
</Feature>
```

In these examples we have geographic features with the type names 'River' and 'Road'. In the road example we have a geometry property called 'linearGeometry' and a couple of simple properties that can be encoded as parsed character data; a string property called 'classification' and an integer property called 'number'. Note that GML Profile 1 does not provide a means to describe the feature type, instead it relies on the name of the feature type. Similarly GML Profile 1 cannot describe the type of simple properties, other than to specify its name and state its value type. Currently GML Profile 1 only supports the value types:

- boolean
- integer
- real
- string

In these examples the values of the geometricProperty is a LineString. However GML Profile 1 cannot provide an explicit connection between the typeName of the geometricProperty and the type of the enclosed geometry element.

In GML Profile 1, GML data is stored or exchanged using feature collection documents. A FeatureCollection is a collection of GML Profile 1 Features, as described in the above example fragments. Elements in the FeatureCollection are selected using the featureMember property which is interpreted as returning the next Feature in the collection. A FeatureCollection thus consists of a set of featureMember tags each enclosing Feature elements similar to the above example.

The name of the containment relationship between FeatureCollection and Feature is specified by the typeName attribute on the featureMember tag. It should be noted that, in many ways, the featureMember and FeatureCollection tags should be considered as different parts of the definition of a feature collection. **Thus the typeName attribute for all the featureMember tags in a FeatureCollection should be the same.** If a number of different typeNames are used, then each would correspond to a different interpretation of the feature collection. This would move the definition of the feature collection down from the FeatureCollection class to the featureMember property. This not only defies the

intended distinction of class and property, it makes the interpretation of the boundedBy property of the FeatureCollection ambiguous.

The full GML Feature DTD is:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
============================================================-->
<!--        G e o g r a p h
y                                       -->
<!--        M a r k u
p                                            -->
<!--        L a n g u a g
e                                         -->
<!--
                                                                -
->
<!--        ( G M
L )                                               -->
<!--
                                                                -
->
<!--        F E A T U R E     D T
D                                 -->
<!--
                                                                -
->
<!--        Copyright (c) 2000 OGC All Rights
Reserved.             -->
<!--
============================================================ --
>

<!-- The GML Feature DTD includes the GML Geometry DTD as an
external entity reference. --->

<!ENTITY % GMLGEOMETRYDTD SYSTEM "gmlgeometry.dtd">
%GMLGEOMETRYDTD;

<!-- A feature contains a set of properties (simple and/or
geometric). In addition a feature can optionally contain a
description. A feature must specify its feature type by name
(typeName). It may optionally provide an identifier for use
within its containing feature collection (identifier) -->

<!ELEMENT Feature (
            description?, name?, boundedBy?,
            property*, geometricProperty* )>

<!ATTLIST Feature
   typeName   CDATA #REQUIRED
   identifier CDATA #IMPLIED >

<!-- A feature collection has the same definition as a feature,
but in addition a feature collection may contain featureMembers.
The boundedBy element is mandatory for feature collections. -->
```

```
<!ELEMENT FeatureCollection (
            description?, name?, boundedBy,
            property*, geometricProperty*,
            featureMember* )>

<!ATTLIST FeatureCollection
   typeName   CDATA #REQUIRED
   identifier CDATA #IMPLIED >

<!-- A featureMember can be a Feature or a FeatureCollection. The
name of the containment relationship between the containing
FeatureCollection and contained Features is specified by the
typeName attribute. -->

<!ELEMENT featureMember ( Feature | FeatureCollection )>

<!ATTLIST featureMember
   typeName   CDATA #REQUIRED >

<!-- Simple properties hold the property value as parsed
character data. The type of the value is specified by the type
attribute, which defaults to the 'string' type. The name of the
property is specified by the typeName attribute. -->

<!ELEMENT property (#PCDATA)>
<!ATTLIST property
   typeName   CDATA #REQUIRED
   type ( boolean | integer | real | string ) "string" >

<!-- Geometric properties hold the property value as a contained
geometry element. The name of the property is specified by the
typeName attribute. -->

<!ELEMENT geometricProperty (%GeometryClasses;)>
<!ATTLIST geometricProperty
   typeName   CDATA #REQUIRED >
```

Download this GML Feature DTD (gmlfeature.dtd)

Note that the GML Feature DTD references the GML Geometry DTD. Note further that, as written, the GML Geometry DTD (gmlgeometry.dtd) must reside in the same directory as the GML Feature DTD (gmlfeature.dtd).

Note that a FeatureCollection element contains optional name and description elements, a mandatory boundedBy element, zero or more property elements, zero or more geometry elements and zero or more featureMembers. The property and geometry property elements refer to the FeatureCollection as a whole. The Box geometry element enclosed by the boundedBy element defines a maximum bounding rectangle in the specified spatial reference system (srsName attribute of the Box element) for all of the features in the feature collection.

Note that a Feature element contains optional name and description elements, an optional boundedBy element defining a minimum bounding rectangle for the Feature, zero or more

properties and zero or more geometry properties. The properties (non-geometry properties) can have any type name but must have a value type which is one of boolean, integer, real or string. The interpretation of these value types is up to the application reading the GML Profile 1 data file. It is anticipated that these will be mapped to XML Schema type definitions in a subsequent revision of this specification.

The XML document below provides a complete encoding of the Cambridge example using GML Profile 1. Note that the sections marked in light blue represent the encoding of the feature collection itself. The encoding of the individual features (light green) is the same as described earlier in this section.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE FeatureCollection SYSTEM "gmlfeature.dtd" >

<FeatureCollection typeName="CityModel">
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>
            0.0,0.0 100.0,100.0
         </coordinates>
      </Box>
   </boundedBy>
   <property typeName="modelDate">
      Feb 2000.
   </property>
   <featureMember typeName="modelMember">
      <Feature typeName="River">
         <name>
            Cam
         </name>
         <description>
            The river that runs through Cambridge.
         </description>
         <geometricProperty typeName="centerLineOf">
            <LineString srsName="EPSG:4326">
               <coordinates>
                  0.0,50.0 100.0,50.0
               </coordinates>
            </LineString>
         </geometricProperty>
      </Feature>
   </featureMember>
   <featureMember typeName="modelMember">
      <Feature typeName="Road">
         <description>
            M11
         </description>
         <property typeName="classification">
            motorway
         </property>
         <property typeName="number" type="integer">
            11
         </property>
         <geometricProperty typeName="linearGeometry">
            <LineString srsName="EPSG:4326">
               <coordinates>
                  0.0,100.0 100.0,0.0
```

```
                </coordinates>
             </LineString>
          </geometricProperty>
       </Feature>
    </featureMember>
</FeatureCollection>
```

Download this example XML (example_profile1.xml)

The names in **blue bold** are those taken from the example and are 'extending' the standard set of names defined by GML.

The XML document below provides a complete encoding of the Schools example using GML Profile 1.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE FeatureCollection SYSTEM "gmlfeature.dtd" >

<FeatureCollection typeName="State">
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>0.0,0.0 50.0,50.0</coordinates>
      </Box>
   </boundedBy>
   <featureMember typeName="featureMember">
      <FeatureCollection typeName="SchoolDistrict">
         <property typeName="districtName">111</property>
         <boundedBy>
            <Box srsName="EPSG:4326">
               <coordinates>0.0,0.0 50.0,40.0</coordinates>
            </Box>
         </boundedBy>
         <geometricProperty typeName="extentOf">
            <Polygon srsName="EPSG:4326">
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>0.0,0.0 50.0,0.0 50.0,40.0,
0.0,0.0</coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </geometricProperty>
         <featureMember typeName="districtMember">
            <Feature typeName="School">
               <property typeName="principalName">111-
1</property>
               <geometricProperty typeName="location">
                  <Point srsName="EPSG:4326">
                     <coordinates>20.0,5.0</coordinates>
                  </Point>
               </geometricProperty>
            </Feature>
         </featureMember>
         <featureMember typeName="districtMember">
            <Feature typeName="School">
               <property typeName="principalName">111-
2</property>
               <geometricProperty typeName="location">
```

```
                  <Point srsName="EPSG:4326">
                     <coordinates>40.0,5.0</coordinates>
                  </Point>
               </geometricProperty>
            </Feature>
         </featureMember>
      </FeatureCollection>
   </featureMember>
   <featureMember typeName="featureMember">
      <FeatureCollection typeName="SchoolDistrict">
         <property typeName="districtName">222</property>
         <boundedBy>
            <Box srsName="EPSG:4326">
               <coordinates>0.0,0.0 40.0,50.0</coordinates>
            </Box>
         </boundedBy>
         <geometricProperty typeName="extentOf">
            <Polygon srsName="EPSG:4326">
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>0.0,0.0 40.0,50.0 0.0,50.0
0.0,0.0</coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </geometricProperty>
         <featureMember typeName="districtMember">
            <Feature typeName="School">
               <property typeName="principalName">222-
1</property>
               <geometricProperty typeName="location">
                  <Point srsName="EPSG:4326">
                     <coordinates>5.0,20.0</coordinates>
                  </Point>
               </geometricProperty>
            </Feature>
         </featureMember>
         <featureMember typeName="districtMember">
            <Feature typeName="College">
               <property typeName="principalName">222-
2</property>
               <geometryPropety typeName="pointProperty">
                  <Point srsName="EPSG:4326">
                     <coordinates>5.0,40.0</coordinates>
                  </Point>
               </geometricProperty>
            </Feature>
         </featureMember>
      </FeatureCollection>
   </featureMember>
</FeatureCollection>
```

4.5. Encoding Spatial Reference Systems (informative)

This section describes the encoding of Spatial Reference Systems, sometimes referred to by the more general phrase 'Coordinate Systems', for the Profile 1 User.

The GML Profile 1 user should note that the optional srsName attribute on each of the Geometry elements takes simply a string value. In GML Profile 1 the value of this attribute is treated as a name only, and it is not required that this attribute point to a

spatial reference system dictionary entry. The GML Profile 1 user can thus decide to ignore the encoding of Spatial Reference Systems altogether.

For the reader interested in building spatial reference system dictionaries please see Section 7.0.

5. Profile 2 - User Defined Feature DTD

5.1. Overview

GML Profile 2 uses the same Geometry DTD (gmlgeometry.dtd) and Spatial Reference System DTD's as GML Profile 1. These are augmented with an application specific Feature DTD:

- Application Specific Feature DTD (e.g. example_profile2_schema.dtd)
- GML Geometry DTD (gmlgeometry.dtd)
- GML Spatial Reference System DTD (ebcsdictionary.dtd)

Unlike GML Profile 1, GML Profile 2 does not have a fixed GML Feature DTD. Instead the user can construct their own application specific feature DTD following normative rules of the GML specification.

The GML Geometry DTD provides the user with a predefined set of geometry properties that they can use to describe geographic features by including them in their application specific Feature DTD. These geometry properties include common properties of geographic entities such as location and extent.

In addition GML Profile 2 also provides the user with some basic metadata for describing geographic features including name and description.

5.2. Encoding Geometry

Geometry values are encoded using the GML Geometry DTD introduced in Section 3.

5.3. Encoding Geometry Properties

This section describes the geometry properties that are provided as part of the GML Geometry DTD. These properties are used by the GML Profile 2 users when they construct their own application specific Feature DTD. The GML Geometry DTD provides a number of 'descriptive names' for geometry properties. These are encoded in the English language currently. Subsequent translations of this specification into other languages will provide these geometry properties in other languages as well using the xmllang attribute.

There are three levels of naming geometry properties in GML:

1. **Formal names:** these name geometry properties in a formal manner based on the type of geometry allowed as a property value.
2. **Descriptive names:** these provide a set of GML endorsed synonyms for the formal names. Although these offer no additional functionality, they represent a more user-fiendly set of names. Later releases of GML will provide more information on their use.
3. **User-defined names:** there is always a need to allow users their own choice of names.

GML Profile 2 introduces a number of formal names. These can be described using the notation from Section 2.2:

```
geometryProperty( Feature ) --> Geometry

boundedBy( Feature ) --> Box

pointProperty( Feature ) --> Point
lineStringProperty( Feature ) --> LineString
polygonProperty( Feature ) --> Polygon

muliPointProperty( Feature ) --> MultiPoint
multiLineStringProperty( Feature ) --> MultiLineString
multiPolygonProperty( Feature ) --> MultiPolygon

geometryCollectionProperty( Feature ) --> GeometryCollection
```

Note that GML Profile 2 can make use of geometryProperty which is defined in the GML Geometry DTD. This is different from the geometricProperty defined as part of GML Feature DTD in GML Profile 1, although the role they play is similar. Different names were required to avoid a name clash.

GML Profile 2 also introduces descriptive names for these properties dependent on the type of geometry they map onto:

- **pointProperty:** location, position, centerOf
- **lineStringProperty:** centerLineOf, edgeOf
- **polygonProperty:** extentOf, coverage
- **multiPointProperty:** multiLocation, multiPosition, multiCenterOf
- **multiLineStringProperty:** multiCenterLineOf, multiEdgeOf
- **multiPolygonProperty:** multiExtentOf, multiCoverage

The precise semantics of these geometry properties (e.g. "What does position of an object mean?" or "Are location and position synonymous?") is not currently part of the GML specification, however, it is anticipated that these will be defined in a subsequent release.

It should be noted that there are no inherent restrictions in the type of geometry property a feature type may have. For example, the 'Radio Tower' feature type could have a geometry property called 'location' that returns a Point geometry to identify its location, and have another geometry property called 'extentOf' that returns a Polygon geometry

describing its physical structure. There is no requirement or all these geometry return types to be the same.

5.3.1. Point Properties

A point property is a geometry property that takes values in the class of Points. It might be used for example to express the location of a feature. In GML Profile 2 the domain of point property is Feature.

GML defines the following explicit point properties which are sub-properties of pointProperty:

- **centerOf**
- **location**
- **position**

Example using descriptive name 'centerOf'

```
<centerOf>
   <Point srsName="EPSG:4326">
      <coordinates>
         0.0,0.0
      </coordinates>
   </Point>
</centerOf>
```

5.3.2. Line String Properties

A line string property is a geometry property that takes values in the class of LineStrings. It might be used for example to express the centerline or edges of a feature. In GML the domain of line string property is Feature.

GML defines the following explicit line string properties which are sub-properties of lineStringProperty:

- **centerLineOf**
- **edgeOf**

Example using formal name 'lineStringProperty'

```
<lineStringProperty>
   <LineString srsName="EPSG:4326">
      <coordinates>
         0.0,0.0 100.0,100.0
      </coordinates>
   </LineString>
</lineStringProperty>
```

5.3.3. Polygon Properties

A polygon property is a geometry property that takes values in the class of Polygons. It might be used for example to express the extent or coverage of a feature. In GML the domain of polygon property is Feature

GML defines the following explicit polygon properties which are sub-properties of polygonProperty:

- **extentOf**
- **coverage**

Example using descriptive name 'extentOf'

```
<extentOf>
   <Polygon srsName="ESPG:4326">
      <outerBoundaryIs>
         <LinearRing>
            <coordinates>
               0.0,0.0 100.0,0.0 50.0,100.0 0.0,10.0
            </coordinates>
         </LinearRing>
      </outerBoundaryIs>
   </Polygon>
</extentOf>
```

5.3.4. Multi Geometry Properties

There are corresponding geometry properties defined for the 'multi-geometries'. A complete definition of the GML Geometry DTD, which includes the geometry property definitions, can be found in Appendix A.

5.4. Encoding Geographic Features

GML Profile 2 allows the user to construct an application specific Feature DTD. Before looking at the rules that govern this DTD, it is illustrative to note how it significantly improves the readability of the resulting feature encoding. Consider the XML fragments from encoding our standard River and Road examples (compare with Section 4.4):

River example

```
<River>
   <name>
      Cam
   </name>
   <description>
      The river that runs through Cambridge.
   </description>
   <centerLineOf>
      <LineString srsName="EPSG:4326">
         <coordinates>
            0.0,50.0 100.0,50.0
         </coordinates>
      </LineString>
```

```
      </centerLineOf>
   </River>
```

Road example

```
   <Road>
      <description>
         M11
      </description>
      <classification>
         motorway
      </classification>
      <number>
         11
      </number>
      <linearGeometry>
         <LineString srsName="EPSG:4326">
            <coordinates>
               0.0,100.0 100.0,0.0
            </coordinates>
         </LineString>
      </linearGeometry>
   </Road>
```

The parts marked in **blue bold** indicate changes from the GML Profile 1 encoding. Note that this approach is more consistent with the XML Namespace Specification [XMLNS] as we can more clearly write the road example fragment with namespaces **gml** and **camb** (for Cambridge) as:

```
   <camb:Road>
      <gml:description>
         M11
      </gml:description>
      <camb:classification>
         motorway
      </camb:classification>
      <camb:number>
         11
      </camb:number>
      <camb:linearGeometry>
         <gml:LineString srsName="EPSG:4326">
            <gml:coordinates>
               0.0,100.0 100.0,0.0
            </gml:coordinates>
         </gml:LineString>
      </camb:linearGeometry>
   </camb:Road>
```

Addition of the namespace references makes it clear that description, LineString etc. are defined in the **gml** namespace, while Road and number are defined in the **camb** namespace.

The fragment from the application specific Feature DTD that defines the Road and River feature types is given below:

```
<!ELEMENT River (
            description?, name?, boundedBy?,
            centerLineOf ) >

<!ELEMENT Road (
            description?, name?, boundedBy?,
            classification, number, linearGeometry ) >

<!ELEMENT classification (#PCDATA) >
<!ELEMENT number (#PCDATA) >

<!ELEMENT linearGeometry (LineString) >
```

(where the names in **bold** are those that come from the example and are not defined by GML.)

The rules governing the defintion of application specific feature types are:

- For each application specific feature type define a new element with the appropriate name (in this example **River** and **Road**). These elements should allow for the optional containment of name, description and boundedBy elements (all of these are defined in the GML Geometry DTD).
- For each application specific property define a new element with the appropriate name (in this example **classification**, **number** and **linearGeometry**). These elements should each be defined to contain the appropriate data type. In this example the simple data types (string and integer) are held as parsed character data. The geometry data type is held as a geometry element of the correct type (in this example LineString) which are defined in the GML Geometry DTD.
- The application specific feature type elements should allow the containment of the relevant property elements. These can be either GML defined properties (for example centerLineOf) or application specific properties (for example **classification**, **linearGeometry**). In this example the River element can contain a centerLineOf element.

In addition it is necessary to define a featue collection that can contain the roads and rivers. This is done with the DTD fragment:

```
<!ELEMENT CityModel (
            description?, name?, boundedBy,
            modelDate,
            modelMember*) >

<!ELEMENT modelDate (#PCDATA) >

<!ELEMENT modelMember ( Road | River ) >
```

(where the names in **bold** are those that come from the example and are not defined by GML.)

Since a feature collection is a type of feature, all the previous rules apply. However there are additional rules governing the definition of the feature type representing the feature collection:

- The feature collection element (in this example **CityModel**) must contain a boundedBy element. The Box contained by the boundedBy property defines the spatial extent of all of the features in the feature collection.
- The feature collection element references the contained features through an appropriate 'member' property, which is defined as an element (in this example **modelMember**). It is possible to enforce some cardinality constraints on the number of features in the feature collection, since the feature collection element must contain the member property element. In this example a CityModel can contain zero or more modelMembers.
- The member property element is defined to contain one of the application specific feature types (in this example either a **Road** or a **River**).
- There should only be one member property defined per feature collection.

The final rule reflects the fact that the feature collection and member property elements define the FeatureCollection *together*. If more than one member property element were allowed per feature collection element, the definition of the collection effectively moves from the feature collection class to the member property. This breaks the unified concept of a FeatureCollection which requires both. For example a FeatureCollection has a boundedBy property. It should be stressed that FeatureCollections are not designed to solve the general problem of relationships between features. Clearly this level of encoding in XML can, at best, describe a simple hierarchy of FeatureCollections and does not allow a Feature to participate in more than one FeatureCollection. Perhaps more surprisingly, it does not allow the description of 'structures' whereby a FeatureCollection like a 'State' might be expected to refer to a single 'Capital' Feature and a set of 'County' Features. In this example the set of 'County' Features is itself a FeatureCollection.

> *It is important to note that this level of flexibility poses some technical problems. For example it is very difficult for an application to mechanically determine the set of allowable feature types for features in a feature collection. In those circumstances where there is no requirement for a hierarchy of feature collections, the problem can be reduced by requiring a fixed name member property (for example featureMember) and inspecting its definition.*

Finally the application specific feature DTD must reference the GML Geometry DTD, typically through an external entity reference. The full application specific feature DTD for the Cambridge example is given below:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ENTITY % GMLGEOMETRYDTD SYSTEM "gmlgeometry.dtd">
%GMLGEOMETRYDTD;

<!ELEMENT CityModel (
            description?, name?, boundedBy,
            modelDate,
            modelMember*) >

<!ELEMENT modelDate (#PCDATA) >

<!ELEMENT modelMember ( Road | River ) >

<!ELEMENT River (
            description?, name?, boundedBy?,
            centerLineOf ) >

<!ELEMENT Road (
            description?, name?, boundedBy?,
            classification, number, linearGeometry ) >

<!ELEMENT classification (#PCDATA) >
<!ELEMENT number (#PCDATA) >

<!ELEMENT linearGeometry (LineString) >
```

Download this example schema (example_profile2_schema.dtd)

Note that in this example it is assumed that the application specific Feature DTD and the GML Geometry DTD are in the same directory. It also explains why it was convenient to place the feature metadata elements (**name** and **description**) in the GML Geometry DTD.

The following XML document encodes the Cambridge example using the application specific Feature DTD defined above. The sections in light blue represent the encoding of the feature collection, while those in light green represent the individual feature encodings given earlier in this section. The parts in **blue bold** represent differences with the GML Profile 1 encoding.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE CityModel SYSTEM "example_profile2_schema.dtd">

<CityModel>
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>
            0.0,0.0 100.0,100.0
         </coordinates>
      </Box>
   </boundedBy>
   <modelDate>
      Feb 2000.
```

```
      </modelDate>
      <modelMember>
        <River>
            <name>
               Cam
            </name>
            <description>
               The river that runs through Cambridge.
            </description>
            <centerLineOf>
               <LineString srsName="EPSG:4326">
                  <coordinates>
                     0.0,50.0 100.0,50.0
                  </coordinates>
               </LineString>
            </centerLineOf>
         </River>
      </modelMember>
      <modelMember>
         <Road>
            <description>
               M11
            </description>
            <classification>
               motorway
            </classification>
            <number>
               11
            </number>
            <linearGeometry>
               <LineString srsName="EPSG:4326">
                  <coordinates>
                     0.0,100.0 100.0,0.0
                  </coordinates>
               </LineString>
            </linearGeometry>
         </Road>
      </modelMember>
   </CityModel>
```

Download this example XML (example_profile2_external_schema.xml)

Note that the application specific Feature DTD does not have to be external. The following example uses an internal application specific Feature DTD and references the GML geometry DTD through an external entity reference:

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE FeatureCollection [

<!ENTITY % GMLGEOMETRYDTD SYSTEM "gmlgeometry.dtd">
%GMLGEOMETRYDTD;

<!ELEMENT FeatureCollection (
            description?, boundedBy,featureMember* )>

<!ELEMENT featureMember (Road)>
```

```
<!ELEMENT Road (description?,centerLineOf)>

]>


<FeatureCollection>
   <description>
      A couple of roads around Cambridge.
   </description>
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>
            0.0,0.0 100.0,100.0
         </coordinates>
      </Box>
   </boundedBy>
   <featureMember>
      <Road>
         <description>
            M11
         </description>
         <centerLineOf>
            <LineString srsName="EPSG:4326">
               <coordinates>
                  0.0,100.0 100.0,0.0
               </coordinates>
            </LineString>
         </centerLineOf>
      </Road>
   </featureMember>
   <featureMember>
      <Road>
         <description>
            A14
         </description>
         <centerLineOf>
            <LineString srsName="EPSG:4326">
               <coordinates>
                  0.0,50.0 0.0,100.0
               </coordinates>
            </LineString>
         </centerLineOf>
      </Road>
   </featureMember>
</FeatureCollection>
```

Download this example XML (example_profile2_internal_schema.xml)

The XML document below provides a complete encoding of the Schools example using GML Profile 2 with an internal schema. Note that it is necessary to define two member properties (featureMember and districtMember) to support the feature collections classes (State and District). Furthermore note that it is possible to require a SchoolDistrict to have at least one School or College within it. Names in **bold** in the schema defintion are specific to the Schools example.

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE State [

<!ENTITY % GMLGEOMETRYDTD SYSTEM "gmlgeometry.dtd">
%GMLGEOMETRYDTD;

<!ELEMENT State ( name?, description?, boundedBy,
            featureMember* )>
<!ELEMENT featureMember( SchoolDistrict )>
<!ELEMENT SchoolDistrict ( name?, description?, boundedBy,
            districtName, extentOf, districtMember+ )>
<!ELEMENT districtName (#PCDATA)>
<!ELEMENT districtMember ( College | School )>
<!ELEMENT School ( name?, description?, boundedBy?,
            principalName, location )>
<!ELEMENT College ( name?, description?, boundedBy?,
            principalName, pointProperty )>
<!ELEMENT principalName (#PCDATA)>

]>

<State>
   <boundedBy>
      <Box srsName="EPSG:4326">
         <coordinates>0.0,0.0 50.0,50.0</coordinates>
      </Box>
   </boundedBy>
   <featureMember>
      <SchoolDistrict>
         <districtName>111</districtName>
         <boundedBy>
            <Box srsName="EPSG:4326">
               <coordinates>0.0,0.0 50.0,40.0</coordinates>
            </Box>
         </boundedBy>
         <extentOf>
            <Polygon srsName="EPSG:4326">
               <outerBoundaryIs>
                  <LinearRing>
                     <coordinates>0.0,0.0 50.0,0.0 50.0,40.0,
0.0,0.0</coordinates>
                  </LinearRing>
               </outerBoundaryIs>
            </Polygon>
         </extentOf>
         <districtMember>
            <School>
               <principalName>111-1</principalName>
               <location>
                  <Point srsName="EPSG:4326">
                     <coordinates>20.0,5.0</coordinates>
                  </Point>
               </location>
            </School>
         </districtMember>
         <districtMember>
            <School>
               <principalName>111-2</principalName>
```

```
                  <location>
                     <Point srsName="EPSG:4326">
                        <coordinates>40.0,5.0</coordinates>
                     </Point>
                  </location>
               </School>
            </districtMember>
         </SchoolDistrict>
      </featureMember>
      <featureMember>
         <SchoolDistrict>
            <districtName>222</districtName>
            <boundedBy>
               <Box srsName="EPSG:4326">
                  <coordinates>0.0,0.0 40.0,50.0</coordinates>
               </Box>
            </boundedBy>
            <extentOf>
               <Polygon srsName="EPSG:4326">
                  <outerBoundaryIs>
                     <LinearRing>
                        <coordinates>0.0,0.0 40.0,50.0 0.0,50.0
0.0,0.0</coordinates>
                     </LinearRing>
                  </outerBoundaryIs>
               </Polygon>
            </extentOf>
            <districtMember>
               <School>
                  <principalName>222-1</principalName>
                  <location>
                     <Point srsName="EPSG:4326">
                        <coordinates>5.0,20.0</coordinates>
                     </Point>
                  </location>
               </School>
            </districtMember>
            <districtMember>
               <College>
                  <principalName>222-2</principalName>
                  <pointProperty>
                     <Point srsName="EPSG:4326">
                        <coordinates>5.0,40.0</coordinates>
                     </Point>
                  </pointProperty>
               </College>
            </districtMember>
         </SchoolDistrict>
      </featureMember>
   </State>
```

5.5. Encoding Spatial Reference Systems (informative)

This section describes the encoding of Spatial Reference Systems, sometimes referred to by the more general phrase 'Coordinate Systems', for the Profile 2 User.

The GML Profile 2 user should note that the optional srsName attribute on each of the Geometry elements takes simply a string value. In GML Profile 2 the value of this attribute is treated as a name only, and it is not required that this attribute point to a spatial reference system dictionary entry. The GML Profile 2 user can thus decide to

ignore the encoding of Spatial Reference Systems altogether.

For the reader interested in building spatial reference system dictionaries please see Section 7.0.

6. Profile 3 - RDF Foundations of GML

6.1. Overview

One of the most important challenges facing the users of geospatial information is to understand the meaning of the data. Much of this meaning was captured in legacy systems by encoding it in non-standard ways within the structure of the data records. A particular data layer, for example, might within a particular GIS environment be used "most of the time" to represent roads and highways. In another system the same roads might be represented by particular numeric feature codes. Translating between such systems is often problematic because the inherent meaning of the data is not captured as part of the data itself but rather in terms of a set of conventions or rules of practice. The result is that data translation, when it happens, must then be accompanied by a painstaking manual process to restore the meaning in the new environment. Knowledge of these and similar problems has been a major motivating factor in the development of GML.

One of the objectives of GML has been to provide a means of encoding geospatial information (e.g. feature types) in such a way that the types employed can be referenced to an external typing framework. Given a GML class instance such as a <Road> (as in GML Profile 2) it should be possible to look up the definition of the class Road in a suitable namespace. Furthermore it should be possible to build feature type definitions from other feature and geometry type definitions.

In the spatial world there is of course no possibility of universal agreement on a set of feature types. The notion of road, for example, typically differs from one geographic region to another. Even within the same geographic region the notion of road required by an ambulance driver may be radically different than that of an insurance investigator, even when they are referring to the same road in the real world. We thus require a means not only to relate different spatial concepts to one another, but also to be able to distribute the description of these concepts in an organized manner.

GML Profile 1 provides an easy to learn XML based encoding for geospatial information. It does not, however, provide a means to relate feature type names to the actual type definitions. This same shortcoming applies also to GML Profile 2. While the use of namespaces in Profile 2 can clearly discriminate what might be ambiguous typeName values in Profile 1, (we can for example write <gc:Road> and <usgs:Road> to discriminate two different road definitions) there is no requirement even with Profile 2 that there is a type definition at the referenced namespace "location".

To resolve these problems GML has been built on the W3C Resource Description Format (RDF). Doing so provides the developer with both a third Profile (GML Profile 3) for encoding geospatial information using RDF, and a formal set of definitions (using RDF Schema) for GML itself.

To make this clearer we refer the reader to Figure 2. With the exception of the GML Feature DTD in GML Profile 1, all of the DTD's used in Profile 2 can be mechanically generated from the GML RDF Schema definitions.

6.2. Encoding Geometry

This section discusses the RDF Schema definitions for the GML Geometry Classes. Note that these definitions are entirely consistent with the GML Geometry DTD of GML Profiles 1 and 2. Consequently this Section does not include examples of geometry class encodings. For these the reader is referred back to Section 3. This Section provides an alternative basis for the encodings using RDF Schema rather than a DTD. It might be noted that sections of the GML Geometry DTD can be mechanically generated from the RDF Schema definitions for the GML Geometry Classes.

6.2.1. Geometry Class

We define an abstract class from which all geometry classes can sub-class. All geometries have a Spatial Reference System, identified by name. The RDF Schema definition for the **Geometry** class is as follows:

```
<rdfs:Class rdf:ID = "Geometry" >
   <rdfs:comment>

Geometry is the root class of the hierarchy. Geometry is an
abstract (non-instantiable) class. All instantiable geometry
classes referenced in this specification are defined so that valid
instances of a geometry class are topologically closed (i.e. all
defined  geometries include their boundary).

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID = "srsName" >
   <rdfs:domain rdf:resource = "#Geometry" />
   <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-rdf-
schema-19990303#Literal" />
</rdf:Property>
```

6.2.2. Point Class

The **Point** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "Point" >
   <rdfs:subClassOf rdf:resource = "#Geometry" />
</rdfs:Class>
```

The **Point** class is capable of referencing coordinate data using the **coordinates** property defined below.

6.2.2. Box Class

The **Box** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "Box" >
   <rdfs:subClassOf rdf:resource = "#Geometry" />
</rdfs:Class>
```

The **Box** class is capable of referencing coordinate data using the **coordinates** property defined below.

6.2.4. Curve Class

The **Curve** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "Curve" >
   <rdfs:subClassOf rdf:resource = "#Geometry" />
   <rdfs:comment>

A Curve is a one-dimensional geometric object usually stored as a
sequence of points, with the subtype of Curve specifying the form
of the interpolation between points. This specification defines
only one subclass of Curve, LineString, which uses linear
interpolation between points.  This is the only 1-D Geometry class
which appears in the GML DTD.

   </rdfs:comment>
</rdfs:Class>
```

The **Curve** class is capable of referencing coordinate data using the **coordinates** property defined below.

6.2.5. Line String Class

The **LineString** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "LineString" >
   <rdfs:subClassOf rdf:resource = "#Curve" />
   <rdfs:comment>

Lines, LineStrings and LinearRings are all Curves. A Line String
is a Curve with linear interpolation between points. Each
consecutive pair of points defines a line segment. A Line is a
LineString with exactly 2 points. In GML the points of a
LineString are defined by a coordinate list and are not defined by
GML Points.

   </rdfs:comment>
</rdfs:Class>
```

The **LineString** class is capable of referencing coordinate data using the **coordinates**

property (defined below) since it sub-classes the **Curve** class.

6.2.6. Linear Ring Class

The **LinearRing** class is defined in RDF Schema as:

```
<rdfs:Class  rdf:ID = "LinearRing" >
   <rdfs:subClassOf  rdf:resource = "#Curve" />
   <rdfs:comment>

A LinearRing is a LineString that is both closed and simple.  In
GML, the points of a LinearRing are defined by a coordinate list
and are not defined by GML Points.

   </rdfs:comment>
</rdfs:Class>
```

The **LinearRing** class is capable of referencing coordinate data using the **coordinates** property (defined below) since it sub-classes the **Curve** class.

6.2.7. Polygon Class

The **Polygon** class is defined as a subclass of GML Geometry on which are defined two properties, namely **outerBoundaryIs** and **innerBoundaryIs**. These two properties return respectively elements of the inner and outer boundary of the polygon. These are, in trun, represented by **LinearRings**. The outer boundary property can appear only once as a property of a polygon class instance. The inner boundary property can appear zero or more times on a given polygon class instance. The RDF Schema definition for the **Polygon** class is thus:

```
<rdfs:Class  rdf:ID = "Surface" >
   <rdfs:subClassOf  rdf:resource = "#Geometry" />
</rdfs:Class>

<rdfs:Class  rdf:ID = "Polygon" >
   <rdfs:subClassOf  rdf:resource = "#Surface" />
</rdfs:Class>

<rdf:Property ID = "outerBoundaryIs" >
   <rdfs:range  resource = "#LinearRing" />
   <rdfs:domain resource = "#Polygon" />
</rdf:Property>

<rdf:Property ID = "innerBoundaryIs" >
   <rdfs:range  resource = "#LinearRing" />
   <rdfs:domain resource = "#Polygon" />
</rdf:Property>
```

6.2.8. Geometry Collection Class

The **GeometryCollection** class has a **geometryMember** property that returns the next Geometry in the GeometryCollection. The GeometryCollection class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID="GeometryCollection">
   <rdfs:subClassOf rdf:resource="#Geometry"/>
   <rdfs:subClassOf rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Container" />
   <rdfs:comment>

A GeometryCollection is a geometry that is a collection of 1 or
more geometries. All the elements in a GeometryCollection must be
in the same Spatial Reference System. This is also the Spatial
Reference System for the GeometryCollection. GeometryCollection
places no other constraints on its elements. Subclasses of
GeometryCollection may restrict membership based on dimension and
may also place other constraints on the degree of spatial overlap
between elements.

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID = "geometryMember">
   <rdfs:range  rdf:resource = "#Geometry" />
   <rdfs:domain rdf:resource = "#GeometryCollection" />
   <rdfs:comment>

Selects next member, a Geometry, in the GeometryCollection.
(Plays same role as the li tag in rdf).

   </rdfs:comment>
</rdf:Property
```

6.2.9. MultiPoint Class

The **MultiPoint** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "MultiPoint">
   <rdfs:subClassOf rdf:resource = "#GeometryCollection" />
   <rdfs:comment>

A MultiPoint is a 0 dimensional geometric collection. The elements
of a MultiPoint are restricted to Points. The points are not
connected or ordered.  A MultiPoint is simple if no two Points in
the MultiPoint are equal (have identical coordinate values). The
boundary of a MultiPoint is the empty set.

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID="pointMember">
   <rdfs:range  rdf:resource="#Point"/>
   <rdfs:domain rdf:resource="#MultiPoint"/>
   <rdfs:comment>

Returns the next Point in a MultiPoint.

   </rdfs:comment>
</rdf:Property>
```

6.2.10. MultiLineString Class

The **MultiLineString** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "MultiCurve">
   <rdfs:subClassOf rdf:resource = "#GeometryCollection" />
   <rdfs:comment>

A MultiCurve is a sub-class of GeometryCollection.

   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID = "MultiLineString">
   <rdfs:subClassOf rdf:resource = "#MultiCurve" />
   <rdfs:comment>

A MultiLineString is a MultiCurve whose elements are LineStrings.

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID = "lineStringMember">
   <rdfs:range  rdf:resource = "#LineString" />
   <rdfs:domain rdf:resource = "#MultiLineString" />
   <rdfs:comment>

Returns the next LineString in a MultiLineString.

   </rdfs:comment>
</rdf:Property>
```

6.2.11. MultiPolygon Class

The **MultiPolygon** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "MultiSurface">
   <rdfs:subClassOf rdf:resource = "#GeometryCollection" />
   <rdfs:comment>

A MultiSurface is a sub-class of GeometryCollection.

   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID = "MultiPolygon">
   <rdfs:subClassOf rdf:resource = "#MultiSurface" />
   <rdfs:comment>

A MultiPolygon is a MultiSurface whose elements are Polygons.

   </rdfs:comment>
</rdfs:Class>
```

```
<rdf:Property ID = "polygonMember">
   <rdfs:range  rdf:resource = "#Polygon" />
   <rdfs:domain rdf:resource = "#MultiPolygon" />
   <rdfs:comment>

Returns the next Polygon in a multiPolygon.

   </rdfs:comment>
</rdf:Property>
```

6.2.12. coordinates Property

In order to assign coordinates to geometry class instances GML provides the coordinates property. In this release the **coordinates** property has a range of **Literal**. In a subsequent revision this is expected to be an XML Schema representation of coordinate array. The **coordinates** property is defined in RDF Schema as:

```
<rdf:Property ID = "coordinates" >
   <rdfs:domain rdf:resource = "#Curve" />
   <rdfs:domain rdf:resource = "#Box" />
   <rdfs:domain rdf:resource = "#Point" />
   <rdfs:range  rdf:resource =  "http://www.w3.org/TR/1999/PR-rdf-
schema-19990303#Literal" />
</rdf:Property>
```

Note that this definition also permits the **Point**, **Box**, **LineString** and **LinearRing** classes to have a **coordinates** property.

6.3. Encoding Geometry Properties

This section discusses the RDF Schema definitions for the GML Geometry Properties. Note that these definitions are entirely consistent with the GML Geometry DTD of GML Profile 2. Consequently this Section does not include examples of geometry class encodings. For these the reader is referred back to Section 5.3. This Section provides an alternative basis for the encodings using RDF Schema rather than a DTD. It might be noted that sections of the GML Geometry DTD can be mechanically generated from the RDF Schema definitions for the GML Geometry Properties.

6.3.1 Geometry Properties

We distinguish geometry properties from geometry classes. A geometry property is a function on a Feature that takes it values in a corresponding geometry class. The domain of all of the geometry properties is Feature (see the next Section for a more complete definition of Feature using RDF Schema).

The relationships between the **Feature** and **Geometry** classes and the **geometryProperty** property are defined using RDF Schema as:

```
<rdfs:Class rdf:ID = "Feature">
   <rdfs:comment>

Abstract feature class.  Features can take zero or more geometry
properties

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID = "geometryProperty">
   <rdfs:range  resource = "#Geometry" />
   <rdfs:domain resource = "#Feature"  />
   <rdfs:comment>

Abstract geometry property of a feature.

   </rdfs:comment>
</rdf:Property>
```

This says that any Feature can have a geometryProperty whose value is a Geometry. However if one wishes to be more specific about either the type of geometry that can be held as a property or the naming of the property then one can create sub-properties of geometryProperty.

6.3.2 Point Properties

A Point property is a special case of a Geometry property where the range of the property is restricted to a sub-class of Geometry, namely a Point. When defining this using RDF Schema it is not necessary to respecify the domain since that is inherited from **geometryProperty**. Thus the **pointProperty** is defined in GML using RDF Schema as:

```
<rdf:Property ID = "pointProperty">
   <rdfs:range rdf:resource= "#Point" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a point of the selected
feature. The coordinate values of the point if present are to be
interpreted in the coordinate system associated with the
pointproperty.

   </rdfs:comment>
</rdf:Property>
```

This just says that **pointProperty** is a **geometryProperty** (via subPropertyOf) whose range is Point and whose domain is Feature.

GML defines three additional sub-properties of **pointProperty**, namely:

- **position**
- **location**

- **centerOf**

These just represent additional descriptive names that mean the same as pointProperty. These names might be considered more suitable for everyday usage. This is achieved in RDF Schema by creating a sub-property with the relevant name, no other details are required since they are inherited. The complete set of definitions is given in Appendix C, but the basic RDF Schema definition of the sub-properties of **pointProperty** are given below:

```
<rdf:Property ID = "position">
   <rdfs:subPropertyOf rdf:resource = "#pointProperty" />
</rdf:Property>

<rdf:Property ID = "location">
   <rdfs:subPropertyOf rdf:resource = "#pointProperty" />
</rdf:Property>

<rdf:Property ID = "centerOf">
   <rdfs:subPropertyOf rdf:resource = "#pointProperty" />
</rdf:Property>
```

6.3.3. LineString Properties

A linestring property is a **geometryProperty** that takes values in the class of LineStrings. It might be used for example to express the centerline or edges of a feature. The definition of the **lineStringProperty** in GML is as follows:

```
<rdf:Property ID = "lineStringProperty">
   <rdfs:range rdf:resource = "#LineString" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a linestring of the
selected feature. The coordinate values of the linestring if
present are to be interpreted in the coordinate system associated
with the linestringproperty.

   </rdfs:comment>
</rdf:Property>
```

GML provides two additional **lineStringProperties**:

- **centerLineOf**
- **edgeOf**

6.3.4. Polygon Properties

A polygon property is a **geometryProperty** that takes values in the class of Polygons. It might be used for example to express the extent or coverage of a feature. The definition of the **polygonProperty** in GML is as follows:

```
<rdf:Property ID = "polygonProperty">
   <rdfs:range rdf:resource = "#Polygon" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a polygon of the selected
feature. The coordinate values of the polygon if present are to be
interpreted in the coordinate system associated with the
polygonproperty.

   </rdfs:comment>
</rdf:Property>
```

GML provides two additional **polygonProperties**:

- **extentOf**
- **coverage**

6.3.5. MultiPoint Properties

A MultiPoint property is a **geometryProperty** which takes values in the class of MultiPoints. It might be used for example to express the extent or coverage of a discrete point feature. The definition of **multiPointProperty** in GML is as follows:

```
<rdf:Property ID = "multiPointProperty">
   <rdfs:range rdf:resource = "#MultiPoint" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a multipoint of the
selected feature.

   </rdfs:comment>
</rdf:Property>
```

Several multiPoint properties are provided in GML. Note that these are like the **pointProperties** with the prefix multi. This is required in GML since RDF does not support polymorphism. This may be revised in a future release. The currently supported **multiPointProperties** are:

- **multiLocation**
- **multiCenterOf**
- **multiPosition**

6.3.6. MultiLineString Properties

A MultiLineString property is a geometryProperty which takes values in the class of MultiLines. It might be used for example to express the edges of a complex feature. The definition of **multiLineStringProperty** in GML is as follows:

```
<rdf:Property ID = "multiLineStringProperty">
   <rdfs:range rdf:resource = "#MultiLineString" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a multilinestring of the
selected feature.

   </rdfs:comment>
</rdf:Property>
```

GML provides two additional **multiLineStringProperties**:

- **multiCenterLineOf**
- **multiEdgeOf**.

6.3.7. MultiPolygon Properties

A **multiPolygonProperty** is a geometryProperty which takes values in the class of MultiPolygons. It might be used for example to express the extent of a complex feature. The definition of **multiPolygonProperty** in GML is as follows:

```
<rdf:Property ID = "multiPolygonProperty">
   <rdfs:range rdf:resource = "#MultiPolygon" />
   <rdfs:subPropertyOf rdf:resource = "#geometryProperty" />
   <rdfs:comment>

Abstract property function that returns a MultiPolygon of the
selected feature.

   </rdfs:comment>
</rdf:Property>
```

GML provides two additional **multiPolygonProperties**:

- **multiExtentOf**
- **multiCoverage**

6.4. Encoding Geographic Features

This section describes the RDF Schema classes for GML Features and FeaturCollections. We note that the Profile 3 developer can use these classes to derive additional feature types or geometry classes in their application namespace.

The **Feature** class is defined in RDF Schema as:

```
<rdfs:Class rdf:ID = "Feature">
</rdfs:Class>

<rdf:Property ID = "name" >
   <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-rdf-
schema-19990303#Literal" />
   <rdfs:domain rdf:resource = "#Feature" />
</rdf:Property>

<rdf:Property ID = "description" >
   <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-rdf-
schema-19990303#Literal" />
   <rdfs:domain rdf:resource = "#Feature" />
</rdf:Property>

<rdf:Property ID = "boundedBy" >
   <rdfs:range  rdf:resource = "#Box"/ >
   <rdfs:domain rdf:resource = "#Feature" />
</rdf:Property>
```

This says a that a Feature may have **name** and **description** simple properties whose range are **Literal**. In addition a Feature may have a **boundedBy** geometry property whose range is a **Box**.

In GML Features are typically grouped into FeatureCollections. While there is no set construct in RDF Schema we introduce the **FeatureCollection** class in GML using the following RDF Schema.

```
<rdfs:Class rdf:ID="FeatureCollection">
   <rdfs:subClassOf rdf:resource="#Feature"/>
   <rdfs:comment>

A collection (set) of Features.

   </rdfs:comment>
</rdfs:Class>

<rdf:Property ID = "featureMember" >
   <rdfs:range  rdf:resource = "#Feature" />
   <rdfs:domain rdf:resource = "#FeatureCollection" />
   <rdfs:comment>

Function which returns next Feature in a FeatureCollection.

   </rdfs:comment>
</rdf:Property>
```

This says that a FeatureCollection is a sub-class of Feature and thus inherits name, description and boundedBy properties. In addition it has a **featureMember** property which is to used to select Features from the FeatureCollection.

Note that we do NOT define a Property Class to encode simple properties since this is already part of RDF. Using RDF we can define any number of properties for any RDF

Class. We have merely added a **geometryProperty** with the domain Feature (see previous Section on 'Encoding Geometry properties'). Application specific RDF Schema defintions are then expected to subclass from Feature (using RDF Schema subClassOf) to create application specific feature types such as Road, Building or River. Such derived subclasses can then automatically use the geometryProperty since it is inherited from Feature.

GML Profile 3 provides the ability to add new feature and geometry types in a clearer and more formal manner than is possible with GML Profile 1 or Profile 2. This is illustrated by considering the Cambridge example. The application specific RDF Schema for the Cambridge example is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xml:lang="en"
     xmlns:gml = "http://www.opengis.org/gml#"
     xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs = "http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#">

   <rdfs:Class rdf:ID = "CityModel" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#FeatureCollection" />
   </rdfs:Class>

   <rdfs:Class rdf:ID = "River" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#Feature" />
   </rdfs:Class>

   <rdfs:Class rdf:ID = "Road" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#Feature" />
   </rdfs:Class>

   <rdf:Property ID = "modelDate" >
      <rdfs:domain rdf:resource = "#CityModel" />
      <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Literal" />
   </rdf:Property>

   <rdf:Property ID = "classification" >
      <rdfs:domain rdf:resource = "#Road" />
      <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Literal" />
   </rdf:Property>

   <rdf:Property ID = "number" >
      <rdfs:domain rdf:resource = "#Road" />
      <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Literal" />
   </rdf:Property>

   <rdf:Property ID = "linearGeometry" >
      <rdfs:domain rdf:resource = "#Road" />
```

```
      <rdfs:subPropertyOf rdf:resource =
"http://www.opengis.org/gml#lineStringProperty" />
   </rdf:Property>

   <rdf:Property ID = "modelMember" >
      <rdfs:domain rdf:resource = "#CityModel" />
      <rdfs:range  rdf:resource = "#Road" />
      <rdfs:range  rdf:resource = "#River" />
      <rdfs:subPropertyOf rdf:resource =
"http://www.opengis.org/gml#featureMember" />
   </rdf:Property>

</rdf:RDF>
```

Download this example schema (example_profile3_schema.rdfs)

where the names in **blue bold** are specific to the example and not already defined by GML. Using this RDF Schema definition it is possible to encode the Cambridge example as a set of RDF records, as show below:

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xml:lang="en"

xmlns:camb="http://www.xyzcorp.com/camb/example_profile3_schema.rdf#"
   xmlns:gml="http://www.opengis.org/gml/gml.rdf#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">

   <camb:CityModel>
      <gml:boundedBy>
         <gml:Box srsName="EPSG:4326">
           <gml:coordinates>
               0.0,0.0 100.0,100.0
            </gml:coordinates>
         </gml:Box>
      </gml:boundedBy>
      <camb:modelDate>
         Feb 2000.
      </camb:modelDate>
      <camb:modelMember>
         <camb:River>
            <gml:name>
               Cam
            </gml:name>
            <gml:description>
               The river that runs through Cambridge.
            </gml:description>
            <gml:centerLineOf>
               <gml:LineString srsName="EPSG:4326">
                  <gml:coordinates>
                     0.0,50.0 100.0,50.0
                  </gml:coordinates>
               </gml:LineString>
            </gml:centerLineOf>
         </camb:River>
      </camb:modelMember>
      <camb:modelMember>
```

```
            <camb:Road>
               <gml:description>
                  M11
               </gml:description>
               <camb:classification>
                  motorway
               </camb:classification>
               <camb:number>
                  11
               </camb:number>
               <camb:linearGeometry>
                  <gml:LineString srsName="EPSG:4326">
                     <gml:coordinates>
                        0.0,100.0 100.0,0.0
                     </gml:coordinates>
                  </gml:LineString>
               </camb:linearGeometry>
            </camb:Road>
         </camb:modelMember>
      </camb:CityModel>
   </rdf:RDF>
```

Download this example RDF (example_profile3.rdf)

> *To make use of this example it will be necessary to alter the URLs for the **gml** and **camb** namespaces. The RDF Schema files that are referred to are the GML definition (see Appendix C) and the example schema defined previously.*

It might be noted that, if one ignores the <rdf:RDF> tag and the namespace prefixes, the encoding of the FeatureCollection is identical to that for GML Profile 2 (see Section 5.4). The above example uses four namespaces:

1. **rdf:** Resource Description Format from W3C
2. **rdfs:** RDF Schema from W3C
3. **gml:** Geography Markup Language RDF Schema definition from OGC
4. **camb:** application specific Cambridge RDF Schema definition from the fictitious company xyzcorp.

In might be noted that GML Profile 3 can be used in writing conventional RDF meta-data descriptions as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xml:lang = "en"
   xmlns:st=""
   xmlns:gml="http://www.opengis.org/gml/gml.rdf"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">

   <rdf:Description about =
"http://www.nasa.gov/shuttleradarmap.html" >
```

```
      <st:MappedBy>
         Shuttle SST-99
      </st:MappedBy>
      <st:vehicle>
         Endeavour
      </st:vehicle>
      <st:launchedOn>
         Februrary 11 2000
      </st:launchedOn>

      <gml:coverage>
         <Polygon srsName = "LtLong" >
            <outerBoundaryIs>
               <LinearRing>
                  <coordinates>
                     -180,-54 -180,60 180,60 180,-54
                  </coordinates>
               </LinearRing>
            </outerBoundaryIs>
         </Polygon>
      </gml:coverage>

   </rdf:Description>
</rdf:RDF>
```

The application specific RDF Schema for the Schools example is shown below, the names in **blue bold** are specific to the Schools example. The application specific classes (State, SchoolDistrict and School) inherit basic simple and geomtery properties from the base GML classes. Note that the member property between State and SchoolDistrict (namely featureMember) is also inherited from the standard GML Feature and FeatureCollection classes.

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xml:lang="en"
     xmlns:gml = "http://www.opengis.org/gml#"
     xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs = "http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#">

   <rdfs:Class rdf:ID = "State" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#FeatureCollection" />
   </rdfs:Class>

   <rdfs:Class rdf:ID = "SchoolDistrict" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#FeatureCollection" />
   </rdfs:Class>

   <rdfs:Class rdf:ID = "School" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#Feature" />
   </rdfs:Class>
```

```
   <rdfs:Class rdf:ID = "College" >
      <rdfs:subClassOf rdf:resource =
"http://www.opengis.org/gml#Feature" />
   </rdfs:Class>

   <rdf:Property ID = "districtName" >
      <rdfs:domain rdf:resource = "#SchoolDistrict" />
      <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Literal" />
   </rdf:Property>

   <rdf:Property ID = "principalName" >
      <rdfs:domain rdf:resource = "#School" />
      <rdfs:domain rdf:resource = "#College" />
      <rdfs:range  rdf:resource = "http://www.w3.org/TR/1999/PR-
rdf-schema-19990303#Literal" />
   </rdf:Property>

   <rdf:Property ID = "districtMember" >
      <rdfs:domain rdf:resource = "#SchoolDistrict" />
      <rdfs:range  rdf:resource = "#School" />
      <rdfs:range  rdf:resource = "#College" />
      <rdfs:subPropertyOf rdf:resource =
"http://www.opengis.org/gml#featureMember" />
   </rdf:Property>

</rdf:RDF>
```

6.6. Using Profile 3 in conjunction with Profile 2

The fact that the encodings generated by GML Profiles 2 and 3 are more or less identical is no accident. We anticipate that many users will want to use RDF Schema to define their initial feature types (and possibly new geometry classes and properties as well) and then mechanically generate a DTD to be used as in GML Profile 2. This enables them to have formal definitions for their feature types and at the same time employ widely available XML 1.0 validating parsers. This approach can be summarized as:

- Write application specific schema in RDF Schema building on the GML RDF Schema definition (the **gml** namespace of the previous Section). A user might define schemas for roads, rivers, buildings, railways, mountain peaks, valleys etc. These RDF Schema would then live in the users application namespace (for example the **camb** namespace in the previous Section).
- Generate an application specific Feature DTD from the above RDF Schemas following the rules of GML Profile 2 (See Section 5.4). This can be done mechanically, for example using an XSLT script [XSLT].
- Write your GML data as for GML Profile 2.

6.7. Spatial Reference Systems (informative)

This section describes the encoding of Spatial Reference Systems, sometimes referred to

by the more general phrase 'Coordinate Systems', for the Profile 3 User.

The GML Profile 3 user should note that the optional srsName attribute on each of the Geometry elements takes simply a string value. In GML Profile 3 the value of this attribute is treated as a name only, and it is not required that this attribute point to a spatial reference system dictionary entry. The GML Profile 3 user can thus decide to ignore the encoding of Spatial Reference Systems altogether.

For the reader interested in building spatial reference system dictionaries please see Section 7.0.

6.8. Feature Identity (informative)

All GML Geometry Classes have an optional ID attribute. Its value must be an RDF IDRef as described in the RDF Model and Syntax Specification [RDFMS].

If a Geometry in a file with URI = "`http://www.xyzcorp.com/mydata.xml`" has the ID = "`p143`", then any reference to this geometry external to the file would be = "`http://www.xyzcorp.com/mydata.xml#p143`". If a FeatureCollection is requested from a FeatureCollection server database and copied to a client side file, the Geometry ID's are not altered. The geometry with ID = "`p143`" in "`http://www.xyzcorp.com/mydata.xml`" remains "`p143`" when copied to the client, unless the client wishes to refer to the geometry which resides on the server. This also applies to Features defined through application DTD's. All RDF Schema class instances can have an optional ID attribute that is resolved as a URI in this manner.

6.9. Feature and Geometry References: (informative)

In RDF the resource attribute can be used to refer to a resource. This same mechanism is used in GML as shown by the following example:

```
<Feature resource =
"http://www.xyzcorp.com/mydata#house23" />
```

This is equivalent to including the referenced feature in-line in the document. The same mechanism can be applied to geometry class instances. The following example encodes the fact that "`yourhouse`" and "`myhouse`" have the same location.

```
<Building ID = "yourhouse" .. >
   <location>
      <Point ID = "134">
         <coordinates>
            2455.12, 3443.78
         </coordinates>
      </Point>
  </location>
</Building>

<Building ID = "myhouse" .. >
```

```
    <location>
       <Point resource = "#134" />
    </location>
</Building>
```


7. Spatial Reference Systems (informative)

7.1. Overview

The material in this section is still under review and is expected to change substantially over the next several revisions. Both RDF Schema definitions and DTD's are presented in this section. These are not wholly consistent with one another at the present time.

Spatial Reference Systems (SRS) are encoded using a separate DTD. This DTD is based on the OGC SQL V1.1 Simple Features Specification (OGC 99-036) that is in turn partly based on the EPSG (European Petroleum Standard Group) web site and tables for spatial reference systems.

The encoding of Spatial Reference Systems is intended to support:

- Client validation of a server specified Spatial Reference System. The client can request the SRS description (an XML document) and compare it to its own specifications or show it to a user for verification.
- Client display of a server specified Spatial Reference System.
- Use by a Coordinate Transformation Service to validate an input data source's Spatial Reference System. A Coordinate Transformation Service can compare the SRS description with its own specifications to see if the SRS is consistent with the selected transformation.
- To control automated coordinate transformation by supplying input and output reference system names and argument values.

In this model, Spatial Reference Systems (Earth Based Coordinate Systems) are divided into three types namely:

- Projected (2D)
- Geographic (2D)
- Geocentric

Editor's Note:

Terminology for reference systems in the geospatial community is inconsistent. The OGC has been using Spatial Reference System for what is really a subset of possible coordinate systems. All OGC Spatial Reference Systems in the Implementation Specifications (e.g. OGC SQL V1.1) are really Earth-Based Coordinate Systems. To change terminology will require change

orders to multiple specifications !!

This document will use the following terms that conflict with the current OGC usage.

- ***Spatial Reference System*** *- any means of providing a relative or absolute location, direction or extent. This includes ordinal as well as cardinal measures.*
- ***Coordinate System*** *- a mapping from the points of a spatial region to a Euclidean vector space. Multiple Coordinate systems are required to cover planetary bodies.*
- ***EarthBasedCoordinate System*** *- A Coordinate System that provides coordinates for a point on the Earth relative to the Earth itself. This is accomplished using some model for the figure of the Earth.*

Since a change order to alter these naming conventions has not been drafted, we will continue to use the term Spatial Reference System. Except where explicitly noted we mean an EarthBasedCoordinateSystem.

All of these Spatial Reference Systems refer only to locations on the earth relative to the earth itself.

Projected (2D) systems are based on a Projection and a (2D) Geographic spatial reference system.

Geographic Systems (2D) provide a means of assigning angular coordinates to locations on the surface of the earth and depends in turn on a geodetic datum and ellipsoid for the earth model.

Geocentric Systems provide a means of assigning rectangular coordinates (relative to the earth's center) to points on the earth's surface (or above) based on model of the earth based on a datum and spheroid.

Mixed angular and rectangular coordinate systems are not currently supported.

XML DTD's are not well suited to maintenance of a complex structure like a spatial reference system dictionary. To assist in this process we have broken the logical DTD into several DTD's each of which are used to define a number of sub-dictionaries, including:

- Earth Based Coordinate System Dictionary (ebcsdictionary.dtd)
- Geodetic Datum Dictionary (geodeticdatumdictionary.dtd)
- Ellipsoid Dictionary (ellipsoiddictionary.dtd)
- Projection Parameter Dictionary (projectionparameterdictionary)
- Projection Dictionary (projectiondictionary.dtd)
- UnitsDictionary (unitsdictionary.dtd)

7.2. Geocentric Systems

A Geocentric system is encoded in terms of a datum, spheroid (ellipsoid) , a linear unit of measure, and a choice of Prime Meridian.

The datum is specified as a name only.

7.2.1. Geographic Systems

Geographic Systems use angular coordinates to specify the location of point on the surface of the earth. In order that such coordinates be convertible to other systems, the Geographic System also provides a Prime Meridian, a datum surface and a spheroid (ellipsoid).

The following example is drawn from dictionary of Earth Based Coordinate Systems.

Example

```
<EBCS ID="4326" Dimension="2">
   <Geographic2D>
      <Name>
         WGS 84
      </Name>
      <Authority>
         EPSG
      </Authority>
      <GeodeticDatum ID="http://www.opengis.org/datums/epsg#6326" />
      <PrimeMeridian
ID="http://www.opengis.org/primemeridian/epsg#8901" />
      <CoordinateAxis ID="Lat" Unit="
http://www.opengis.org/units/epsg#9108" />
      <CoordinateAxis ID="Long" Unit="
http://www.opengis.org/units/epsg#9801" />
   </Geographic2D>
</EBCS>
```

Note from the example that the definitions of GeodeticDatum, PrimeMeridian, and CoordinateUnits are not coded in-line.This is in order to allow for separate dictionaries of these items and to minimize maintenance problems.It will be up to the application to locate the referenced item (e.g. PrimeMeridian) and fetch it for processing if required.

7.2.2. Projected Systems

Projected systems provide a means of mapping from the surface of the earth onto a flat surface (Euclidean Plane or surface homemorphic to the Euclidean plane (e.g. Cylinder). So that the project system coordinates can be related to other systems, the Projected Spatial Reference System (Earth Based Coordinate System) provides an underlying Geographic Reference System with a datum, ellipsoid, and Prime Meridian.

Each projected coordinate system can have zero or more parameters associated with it. Standard parameter names can be found in an associated dictionary of Parameter names.

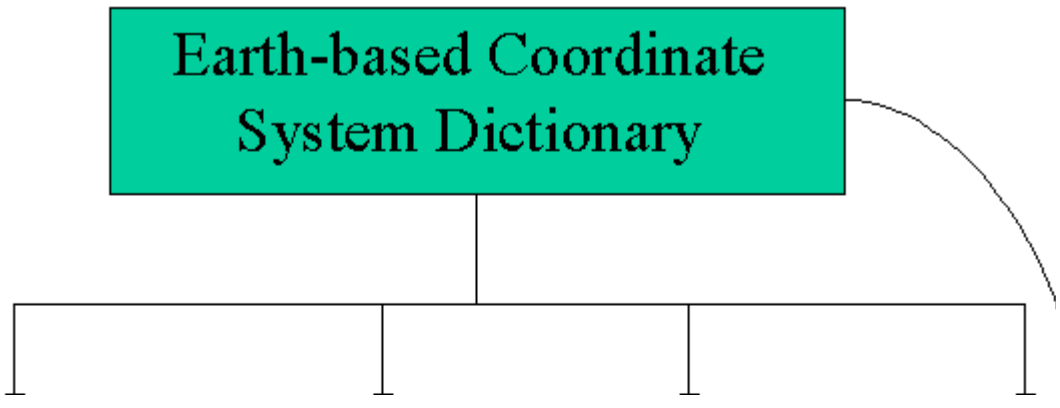
Example

```
<EBCS ID="27700" Dimension="2">
   <Projected2D>
      <name>
         OSGB 1936 / British National Grid
      </name>
      <abbreviation>
         British National Grid
      </abbreviation>
      <authority>
         EPSG
      </authority>
      <Projection
ID="http://www.opengis.org/projections/epsg#TransverseMercator">
         <latitude_of_origin>49</latitude_of_origin>
         <central_meridian>-2</central_meridian>
         <scale_factor>0.999601272</scale_factor>
         <false_easting>400000</false_easting >
         <false_northing>-100000</false_northing >
      </Projection>
      <geographic2dused>
         http://www.opengis.org/ebcsdictionary/epsg#4277
      </geographic2dused>
      <CoordinateAxis ID="E"
Unit="http://www.opengis.org/units/epsg#9001" />
      <CoordinateAxis ID="N"
Unit="http://www.opengis.org/units/epsg#9001" />
   </Projected2D>
</EBCS>
```

7.2.3. Supporting Dictionaries (DTD)

The main DTD (ebcsdictionary.dtd) is supported in GML by a set of DTD's which define the encoding of supporting dictionaries for items such as geodetic datums, ellipsoids, and units. These supporting dictionaries are NOT encoded into the earth-based coordinate system dictionary for reasons of maintainability and data integrity.

Note that the elements in these dictionaries are referenced from one another as shown in Figure 10. At present it is up to the application to decide how to use these references.An application might, for example, import the referenced elements and assemble a complete encoding of a particular coordinate system, or it might simply check the value of a particular data field.



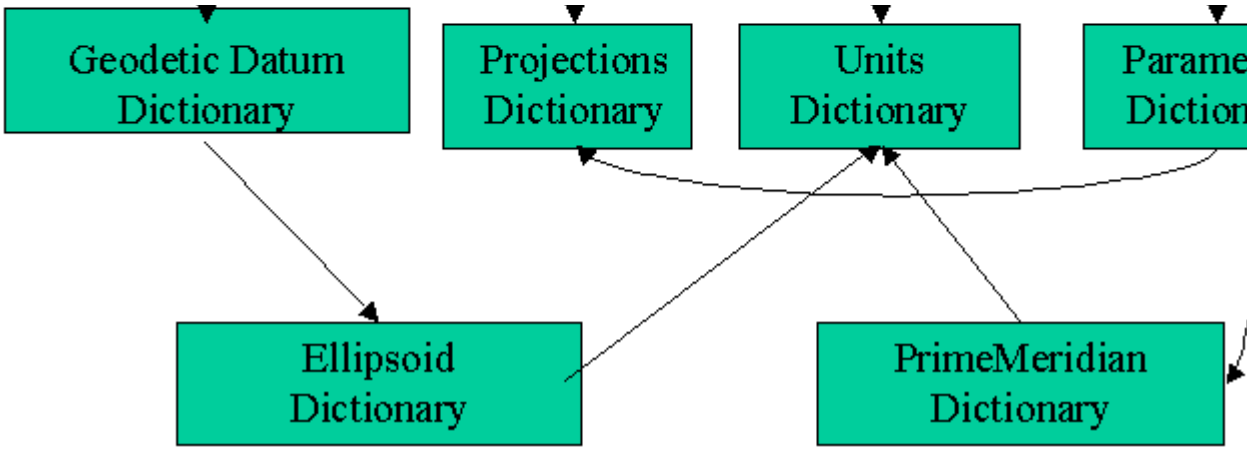


Figure 3. Supporting Dictionaries for Earth Based Coordinate Systems

Each of these dictionaries is defined by a separate DTD. These are attached below:

- Geodetic Datums (Horizontal Datum)
- Ellipsoids
- Standard Parameters
- Prime Meridians
- Units (note that this combines both Linear and Angular Units)

When Xpointer/Xlink technology becomes available (Xpointer reached recommendation status in December 1999), a range reference will enable an XML file to retrieve any dictionary element (or elements) in a single reference statement.

Appendix A: Geometry DTD

```
<!-- ============================================================ -->
<!--        G e o g r a p h y                                     -->
<!--        M a r k u p                                           -->
<!--        L a n g u a g e                                       -->
<!--                                                              -->
<!--        ( G M L )                                             -->
<!--                                                              -->
<!--        G E O M E T R Y   D T D                               -->
<!--                                                              -->
<!--        Copyright (c) 2000 OGC All Rights Reserved.           -->
<!-- ============================================================ -->

<!-- the coordinate element holds a list of coordinates as parsed
character data. Note that it does not reference a SRS and does not
constitute a proper geometry class. -->
<!ELEMENT coordinates (#PCDATA) >
<!ATTLIST coordinates
   decimal  CDATA    #IMPLIED
   cs       CDATA    #IMPLIED
```

```
   ts       CDATA    #IMPLIED >

<!-- the Box element defines an extent using a pair of coordinates and a
SRS name. -->
<!ELEMENT Box (coordinates) >
<!ATTLIST Box
   ID       CDATA    #IMPLIED
   srsName  CDATA    #REQUIRED >

<!-- ========================================================== -->
<!--        G E O M E T R Y   C L A S S   D e f i n i t i o n s       --
>
<!-- ========================================================== -->

<!-- a Point is defined by a single coordinate. -->
<!ELEMENT Point (coordinates) >
<!ATTLIST Point
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a LineString is defined by two or more coordinates, with linear
interoplation between them. -->
<!ELEMENT LineString (coordinates) >
<!ATTLIST LineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a Polygon is defined by an outer boundary and zero or more inner
boundaries. These boundaries are themselves defined by LinerRings. -->
<!ELEMENT Polygon (outerBoundaryIs, innerBoundaryIs*) >
<!ATTLIST Polygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT outerBoundaryIs (LinearRing) >
<!ELEMENT innerBoundaryIs (LinearRing) >

<!-- a LinearRing is defined by four or more coordinates, with linear
interpolation between them. The first and last coordinates must be
coincident. -->
<!ELEMENT LinearRing (coordinates) >
<!ATTLIST LinearRing
   ID       CDATA    #IMPLIED >

<!-- a MultiPoint is defined by zero or more Points, referenced through a
pointMember element. -->
<!ELEMENT MultiPoint (pointMember+) >
<!ATTLIST MultiPoint
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT pointMember (Point) >

<!-- a MultiLineString is defined by zero or more LineStrings, referenced
through a lineStringMember element. -->
<!ELEMENT MultiLineString (lineStringMember+) >
<!ATTLIST MultiLineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT lineStringMember (LineString) >
```

```
<!-- a MultiPolygon is defined by zero or more Polygons, referenced
through a polygonMember element. -->
<!ELEMENT MultiPolygon (polygonMember+) >
<!ATTLIST MultiPolygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT polygonMember (Polygon) >

<!-- a GeometryCollection is defined by zero or more geometries,
referenced through a geometryMember element. A geometryMember element may
be any one of the geometry classes. -->
<!ENTITY % GeometryClasses "(
   Point | LineString | Polygon |
   MultiPoint | MultiLineString | MultiPolygon |
   GeometryCollection )" >

<!ELEMENT GeometryCollection (geometryMember+) >
<!ATTLIST GeometryCollection
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT geometryMember %GeometryClasses; >

<!-- ============================================================ -->
<!--   G E O M E T R Y   P R O P E R T Y   D e f i n i t i o n s   -->
<!-- ============================================================ -->

<!-- GML provides an 'endorsed' name to define the extent of a feature.
The extent is defined by a Box element, the name of the property is
boundedBy. -->
<!ELEMENT boundedBy (Box) >

<!-- the generic geometryProperty can accept a geometry of any class. -->
<!ELEMENT geometryProperty (%GeometryClasses;) >

<!-- the pointProperty has three descriptive names: centerOf, location
and position. -->
<!ELEMENT pointProperty (Point) >
<!ELEMENT centerOf (Point) >
<!ELEMENT location (Point) >
<!ELEMENT position (Point) >

<!-- the lineStringProperty has two descriptive names: centerLineOf and
edgeOf. -->
<!ELEMENT lineStringProperty (LineString) >
<!ELEMENT centerLineOf (LineString)>
<!ELEMENT edgeOf (LineString)>

<!-- the polygonProperty has two descriptive names: coverage and
extentOf. -->
<!ELEMENT polygonProperty (Polygon) >
<!ELEMENT coverage (Polygon)>
<!ELEMENT extentOf (Polygon)>

<!-- the multiPointProperty has three descriptive names: multiCenterOf,
multiLocation and multiPosition. -->
<!ELEMENT multiPointProperty (MultiPoint) >
<!ELEMENT multiCenterOf (MultiPoint) >
```

```
<!ELEMENT multiLocation (MultiPoint) >
<!ELEMENT multiPosition (MultiPoint) >

<!-- the multiLineStringProperty has two descriptive names:
multiCenterLineOf and multiEdgeOf. -->
<!ELEMENT multiLineStringProperty (MultiLineString) >
<!ELEMENT multiCenterLineOf (MultiLineString) >
<!ELEMENT multiEdgeOf (MultiLineString) >

<!-- the multiPolygonProperty has two descriptive names: multiCoverage
and multiExtentOf. -->
<!ELEMENT multiPolygonProperty (MultiPolygon) >
<!ELEMENT multiCoverage (MultiPolygon) >
<!ELEMENT multiExtentOf (MultiPolygon) >

<!ELEMENT geometryCollectionProperty (GeometryCollection) >

<!-- ============================================================ -->
<!--     F E A T U R E   M E T A D A T A   D e f i n i t i o n s    -->
<!-- ============================================================ -->

<!-- Feature metadata, included in GML Geometry DTD for convenience; name
and description are two 'standard' string properties defined by GML. -->

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

Download this GML Geometry DTD (gmlgeometry.dtd)

Appendix B: Spatial Reference Systems DTD's (informative)

```
<!ELEMENT EBCS_DICTIONARY (EBCS*) >

<!ELEMENT EBCS (Projected2D | Geographic2D | Geocentric) >
<!ATTLIST EBCS
   ID        CDATA #REQUIRED
   Dimension CDATA #REQUIRED >

<!ELEMENT Projected2D(
             Name?,
             Abbreviation?,
             Alias?,
             Authority?,
             ProjectionClass,
             Geographic2DUsed,
             CoordinateAxis*,
             Origin?) >

<!ELEMENT Geographic2D (
            Name?,
            Abbreviation?,
            Alias?,
            Authority?,
```

```
              GeodeticDatum,
              PrimeMeridian,
              CoordinateAxis*,
              Origin?) >

<!ELEMENT Geocentric (
              Name?,
              Abbreviation?,
             Alias?,
              Authority?,
              GeodeticDatum,
              PrimeMeridian,
              CoordinateAxis*,
              Origin?) >

<!ELEMENT Name (#PCDATA) >

<!ELEMENT Abbreviation (#PCDATA) >

<!ELEMENT Alias (#PCDATA) >

<!ELEMENT Authority (#PCDATA) >

<!ELEMENT ProjectionClass (Parameter*) >
<!ATTLIST ProjectionClass
   ID     CDATA #REQUIRED >

<!ELEMENT Geographic2DUsed EMPTY >
<!ATTLIST Geographic2DUsed
   ID     CDATA #REQUIRED >

<!ELEMENT Parameter (#PCDATA) >
<!ATTLIST Parameter
   ID     CDATA #REQUIRED
   Units  CDATA #IMPLIED >

<!ELEMENT CoordinateAxis EMPTY >
<!ATTLIST CoordinateAxis
   ID     CDATA #REQUIRED
   Unit   CDATA #REQUIRED >

<!ELEMENT Origin (coordinates?) >
<!ATTLIST Origin
   ID     CDATA #REQUIRED >

<!ELEMENT GeodeticDatum EMPTY >
<!ATTLIST GeodeticDatum
   ID     CDATA #REQUIRED >

<!ELEMENT PrimeMeridian EMPTY >
<!ATTLIST PrimeMeridian
   ID     CDATA #REQUIRED >

<!ELEMENT coordinates (#PCDATA) >
```

This DTD is used by itself (does not require the other DTD's) to construct a library of spatial reference systems. These are then referenced by the geometry class instances defined within the GML Geometry DTD (See Appendix A). The top level SRS DTD is

as follows:

Note that the current release of GML supports the definition of entries for Earth Based Coordinate System Dictionaries only. Subsequent revisions of the GML Specification will provide as well for other types of reference systems.

Appendix C: RDF Schema Definition of GML

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ============================================================= -->
<!--        G e o g r a p h y                                      -->
<!--        M a r k u p                                            -->
<!--        L a n g u a g e                                        -->
<!--                                                               -->
<!--        ( G M L )                                              -->
<!--                                                               -->
<!--        R D F   S c h e m a   D e f i n i t i o n s            -->
<!--                                                               -->
<!--        Copyright (c) 2000 OGC All Rights Reserved.            -->
<!-- ============================================================= -->

<rdf:RDF xml:lang="en"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#">

<!--
================================================================================
->
<!--===  This section describes the abstract classes and properties for core G
=========-->
<!--
================================================================================
->

   <rdfs:Class rdf:ID="Geometry">
      <rdfs:comment>
Geometry is the root class of the hierarchy. Geometry is an abstract (non-inst
class.  All instantiable geometry classes referenced in this specification are
so that valid instances of a geometry class are topologically closed (i.e. all
geometries include their boundary).
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Feature">
      <rdfs:comment>
A Feature is a Property List, some of whose properties are of type geometry. S
classes of geographic feature are created by subtyping from the GML Feature cl
the application namespace.
      </rdfs:comment>
   </rdfs:Class>
```

```
   <rdfs:Class rdf:ID="GeometryCollection">
      <rdfs:subClassOf rdf:resource="#Geometry"/>
      <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Container"/>
      <rdfs:comment>
A GeometryCollection is a geometry that is a collection of 1 or more geometrie
the elements in a GeometryCollection must be in the same Spatial Reference. Th
also the Spatial Reference for the GeometryCollection. GeometryCollection plac
other constraints on its elements. Subclasses of GeometryCollection may restri
membership based on dimension and may also place other constraints on the degr
spatial overlap between elements.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Box">
      <rdfs:subClassOf rdf:resource="#Geometry"/>
      <rdfs:comment>
A rectangular area defined by two points and the four orthogonal geodesic curv
defined by these two points
      </rdfs:comment>
   </rdfs:Class>

   <rdf:Property ID="geometryMember">
      <rdfs:range rdf:resource="#Geometry"/>
      <rdfs:domain rdf:resource="#GeometryCollection"/>
      <rdfs:comment>
selects next member in the geometry collection. Plays same role as li tag in r
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="geometryProperty">
      <rdfs:range rdf:resource="#Geometry"/>
      <rdfs:domain rdf:resource="#Feature"/>
      <rdfs:comment>
Abstract property which is the parent of all geospatial properties.  While OGC
some standard geometry properties users can create additional properties by us
subProperty relationship and deriving from the OGC properties.
      </rdfs:comment>
   </rdf:Property>

   <rdfs:Class rdf:ID="FeatureCollection">
      <rdfs:subClassOf rdf:resource="#Feature"/>
      <rdfs:comment>
A collection (set) of Features
      </rdfs:comment>
   </rdfs:Class>

   <rdf:Property ID="featureMember">
      <rdfs:range rdf:resource="#Feature"/>
      <rdfs:domain rdf:resource="#FeatureCollection"/>
      <rdfs:comment>
Function which returns next Feature in a FeatureCollection
      </rdfs:comment>
   </rdf:Property>

<!--================================================================================
<!-============== This next section defines common metadata properties ======
<!--================================================================================
```

```
   <rdf:Property ID="name">
      <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Literal"/>
      <rdfs:domain rdf:resource="#Feature"/>
   </rdf:Property>

   <rdf:Property ID="boundedBy">
      <rdfs:range rdf:resource="#Box"/>
      <rdfs:domain rdf:resource="#Feature"/>
   </rdf:Property>

   <rdf:Property ID="description">
      <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Literal"/>
      <rdfs:domain rdf:resource="#Feature"/>
   </rdf:Property>

   <rdf:Property ID="srsName">
      <rdfs:range rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Literal"/>
      <rdfs:domain rdf:resource="#Geometry"/>
   </rdf:Property>

   <rdf:Property ID="coordinates">
      <rdfs:domain rdf:resource="#Curve"/>
      <rdfs:domain rdf:resource="#Box"/>
      <rdfs:domain rdf:resource="#Point"/>
      <rdfs:range  rdf:resource="http://www.w3.org/TR/1999/PR-rdf-schema-
19990303#Literal"/>
   </rdf:Property>

<!--================================================================================
<!--=======  This next section defines the GML Geometry Classes ==============
<!--================================================================================

   <rdfs:Class rdf:ID="Point">
      <rdfs:subClassOf rdf:resource="#Geometry"/>
      <rdfs:comment>
Point geometry class. A Point is a 0-dimensional geometry and represents a sin
location in coordinate space. A Point has an x-coordinate value and a y-coordi
value. Note that GML is more general than the OGC SQL v1.1 specification and d
Points of 0-4 (or larger) dimension. The boundary of a Point is the empty set.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Curve">
      <rdfs:subClassOf rdf:resource="#Geometry"/>
      <rdfs:comment>
A Curve is a one-dimensional geometric object usually stored as a sequence of
with the subtype of Curve specifying the form of the interpolation between poi
specification defines only one subclass of Curve, LineString, which uses linea
interpolation between points.  This is the only 1-D Geometry class which appea
GML DTD. Topologically a Curve is a one-dimensional geometric object that is t
homeomorphic image of a real, closed, interval D [a, b] {x in R2| a le x le b}
mapping f:[a,b] --- R2 as defined in [1], section 3.12.7.2.

A Curve is simple if it does not pass through the same point twice ([1], secti
3.12.7.3).
```

A Curve is closed if its start point is equal to its end point. ([1], section 3.12.7.3).

The boundary of a closed Curve is empty.

A Curve that is simple and closed is a Ring.

The boundary of a non-closed Curve consists of its two end points. ([1], secti 3.12.3.2).

A Curve is defined as topologically closed.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="LineString">
      <rdfs:subClassOf rdf:resource="#Curve"/>
      <rdfs:comment>
Lines, LineStrings and LinearRings are all Curves. A Line String is a Curve wi interpolation between points. Each consecutive pair of points defines a line s Line is a LineString with exactly 2 points. In GML the points of a LineString defined by a coordinate list and are not defined by GML Points.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="LinearRing">
      <rdfs:subClassOf rdf:resource="#Curve"/>
      <rdfs:comment>
A LinearRing is a LineString that is both closed and simple. In GML, the poin LinearRing are defined by a coordinate list and are not defined by GML Points.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Surface">
      <rdfs:subClassOf rdf:resource="#Geometry"/>
      <rdfs:comment>
Abstract geometry class for 2D geometries
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="Polygon">
      <rdfs:subClassOf rdf:resource="#Surface"/>
      <rdfs:comment>
A Polygon is a planar Surface, defined by 1 exterior boundary and 0 or more in boundaries. Each interior boundary defines a hole in the Polygon. The assertio polygons (the rules that define valid polygons) are:

1. Polygons are topologically closed.

2. The boundary of a Polygon consists of a set of LinearRings that make up it exterior and interior boundaries. Note that these are captured in GML via the eboundaryis and iboundaryis properties of the Polygon.

3. No two rings in the boundary cross, the rings in the boundary of a Polygon intersect at a Point but only as a tangent.

4. A Polygon may not have cut lines, spikes or punctures:

5. The Interior of every Polygon is a connected point set.

6. The Exterior of a Polygon with 1 or more holes is not connected. Each hole a connected component of the Exterior.

In the above assertions, Interior, Closure and Exterior have the standard topo definitions. The combination of 1 and 3 make a Polygon a Regular Closed point Polygons are simple geometries in accordance with the terminology of the OGC A Specififcation 99-101.

```
      </rdfs:comment>
   </rdfs:Class>

   <rdf:Property ID="outerBoundaryIs">
      <rdfs:range rdf:resource="#LinearRing"/>
      <rdfs:domain rdf:resource="#Polygon"/>
      <rdfs:comment>
```

This property returns the outer boundary of a polygon

```
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="innerBoundaryIs">
      <rdfs:range rdf:resource="#LinearRing"/>
      <rdfs:domain rdf:resource="#Polygon"/>
      <rdfs:comment>
```

This property returns a connected component of the interior boundary of a poly polygon can have zero or more iboundaryis properties

```
      </rdfs:comment>
   </rdf:Property>

   <rdfs:Class rdf:ID="MultiPoint">
      <rdfs:subClassOf rdf:resource="#GeometryCollection"/>
      <rdfs:comment>
```

A MultiPoint is a 0 dimensional geometric collection. The elements of a MultiP restricted to Points. The points are not connected or ordered. A MultiPoint i if no two Points in the MultiPoint are equal (have identical coordinate values boundary of a MultiPoint is the empty set.

```
      </rdfs:comment>
   </rdfs:Class>

   <rdf:Property ID="pointMember">
      <rdfs:range rdf:resource="#Point"/>
      <rdfs:domain rdf:resource="#MultiPoint"/>
      <rdfs:comment>
```

Returns the next point in a multipoint

```
      </rdfs:comment>
   </rdf:Property>

   <rdfs:Class rdf:ID="MultiCurve">
      <rdfs:subClassOf rdf:resource="#GeometryCollection"/>
      <rdfs:comment>
```

A MultiCurve is a one-dimensional eometryCollection whose elements are Curves. MultiCurve is present in this specification only to provide the context for th definition of a Multi-Line String. MultiCurve is simple if and only if all of elements are simple, the only intersections between any two elements occur at that are on the boundary. The boundary of a MultiCurve is obtained by applyin 'mod 2' union rule: A point is in the boundary of a MultiCurve if it is in the boundaries of an odd number of elements of the MultiCurve ([1], section 3.12.3

```
MultiCurve is closed if all of its elements are closed.

The boundary of a closed MultiCurve is always empty.  MultiCurve is defined as
topologically closed.
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="MultiLineString">
      <rdfs:subClassOf rdf:resource="#GeometryCollection"/>
      <rdfs:comment>
A MultiLineString is a MultiCurve whose elements are LineStrings
      </rdfs:comment>
   </rdfs:Class>

   <rdf:Property ID="lineStringMember">
      <rdfs:range rdf:resource="#LineString"/>
      <rdfs:domain rdf:resource="#MultiLineString"/>
      <rdfs:comment>
Returns the next linestring in a multilinestring
      </rdfs:comment>
   </rdf:Property>

   <rdfs:Class rdf:ID="MultiSurface">
      <rdfs:subClassOf rdf:resource="#GeometryCollection"/>
      <rdfs:comment>
Abstract class for complex 2-D geometries
      </rdfs:comment>
   </rdfs:Class>

   <rdfs:Class rdf:ID="MultiPolygon">
      <rdfs:subClassOf rdf:resource="#MultiSurface"/>
      <rdfs:comment>
A MultiPolygon is a MultiSurface whose elements are Polygons. The assertions f
MultiPolygons are:

1.  The interiors of 2 Polygons that are elements of a MultiPolygon may not in

2.  The Boundaries of any 2 Polygons that are elements of a MultiPolygon may n
'cross' and may touch at only a finite number of points. (Note that crossing i
prevented by assertion 1 above).

3.  A MultiPolygon is defined as topologically closed.

4.  A MultiPolygon may not have cut lines, spikes or punctures; a MultiPolygon
Regular, Closed point set:

5.  The interior of a MultiPolygon with more than 1 Polygon is not connected,
number of connected components of the interior of a MultiPolygon is equal to t
of Polygons in the MultiPolygon.

The boundary of a MultiPolygon is a set of closed curves (LinearRings) corresp
the boundaries of its element Polygons.  Each Curve in the boundary of the Mul
is in the boundary of exactly 1 element Polygon, and every Curve in the bounda
element Polygon is in the boundary of the MultiPolygon.
      </rdfs:comment>
   </rdfs:Class>
```

```
   <rdf:Property ID="polygonMember">
      <rdfs:range rdf:resource="#Polygon"/>
      <rdfs:domain rdf:resource="#MultiPolygon"/>
      <rdfs:comment>
Returns the next polygon in a multipolygon
      </rdfs:comment>
   </rdf:Property>

<!--================================================================================
>
<!--============== This section defines the GML geometry properties. ==========
>
<!--========== All of these properties are sub-Properties of geometryproperty
>
<!--================================================================================
>

   <rdf:Property ID="pointProperty">
      <rdfs:range rdf:resource="#Point"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a point of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="lineStringProperty">
      <rdfs:range rdf:resource="#LineString"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a linestring of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="polygonProperty">
      <rdfs:range rdf:resource="#Polygon"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a polygon of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="location">
      <rdfs:subPropertyOf rdf:resource="#pointProperty"/>
      <rdfs:comment>
Returns a point of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="position">
      <rdfs:subPropertyOf rdf:resource="#pointProperty"/>
      <rdfs:comment>
Returns a point of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="centerOf">
      <rdfs:subPropertyOf rdf:resource="#pointProperty"/>
      <rdfs:comment>
Returns the center point of the selected feature.
```

```
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="centerLineOf">
      <rdfs:subPropertyOf rdf:resource="#lineStringProperty"/>
      <rdfs:comment>
Returns a linestring which is the centerline of  the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="edgeOf">
      <rdfs:subPropertyOf rdf:resource="#lineStringProperty"/>
      <rdfs:comment>
Returns a linestring which is an edge of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="extentOf">
      <rdfs:subPropertyOf rdf:resource="#polygonProperty"/>
      <rdfs:comment>
Returns a polygon which is the centerline of  the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="coverage">
      <rdfs:subPropertyOf rdf:resource="#polygonProperty"/>
      <rdfs:comment>
Returns a polygon which is the centerline of  the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiPointProperty">
      <rdfs:range rdf:resource="#MultiPoint"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a multipoint of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiLineStringProperty">
      <rdfs:range rdf:resource="#MultiLineString"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a multilinestring of the selected feat
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiPolygonProperty">
      <rdfs:range rdf:resource="#MultiPolygon"/>
      <rdfs:subPropertyOf rdf:resource="#geometryProperty"/>
      <rdfs:comment>
Abstract property function that returns a MultiPolygon of the selected feature
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiLocation">
      <rdfs:subPropertyOf rdf:resource="#multiPointProperty"/>
      <rdfs:comment>
Returns a multipoint of the selected feature.
      </rdfs:comment>
```

```
   </rdf:Property>

   <rdf:Property ID="multiPosition">
      <rdfs:subPropertyOf rdf:resource="#multiPointProperty"/>
      <rdfs:comment>
Returns a multipoint of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiCenterOf">
      <rdfs:subPropertyOf rdf:resource="#multiPointProperty"/>
      <rdfs:comment>
Returns the multi-center point of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiCenterLineOf">
      <rdfs:subPropertyOf rdf:resource="#multiLineStringProperty"/>
      <rdfs:comment>
Returns a multilinestring which is the multicenterline of  the selected featur
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiEdgeOf">
      <rdfs:subPropertyOf rdf:resource="#multiLineStringProperty"/>
      <rdfs:comment>
Returns a multilinestring which is a set of edges of the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiExtentOf">
      <rdfs:subPropertyOf rdf:resource="#multiPolygonProperty"/>
      <rdfs:comment>
Returns a MultiPolygon which is the extent of  the selected feature.
      </rdfs:comment>
   </rdf:Property>

   <rdf:Property ID="multiCoverage">
      <rdfs:subPropertyOf rdf:resource="#multiPolygonProperty"/>
      <rdfs:comment>
Returns a MultiPolygon which is the coverage of  the selected feature.
      </rdfs:comment>
   </rdf:Property>
</rdf:RDF>
```

Download this GML RDF Schema Definition (gml.rdfs)

Appendix D: References

[POIX] Point of Interest Exchange Language Specification. Available at http://www.w3.org/TR/poix/

[QNAME] QNAME specification. Available at http://18.29.1.23:80/TR/REC-xml-names/#NT-QName

[RDFMS] Resource Description Framework (RDF) Model and Syntax. Available at http://www.w3.org/TR/REC-rdf-syntax

[RDFSchema] Resource Description Framework (RDF) Schemas; Brickley, Guha, Layman eds., World Wide Web Consortium Working Draft; http://www.w3.org/TR/PR-rdf-schema

[SVG] Scalable Vector Graphics. Available at http://www.w3.org/TR/SVG/

[URI] Uniform Resource Identifiers (URI): Generic Syntax; Berners-Lee, Fielding, Masinter, Internet Draft Standard August, 1998; RFC2396.

[VML] Vector Markup Language. Available at: http://www.w3.org/TR/NOTE-VML

[VRML] Virtual Reality Markup Language. Available at: http://www.vrml.org/VRML2.0/FINAL

[XML SCHEMA] XML Schema Part 1: Structures. Available at http://www.w3.org/TR/xmlschema-1

[XML SCHEMA DATATYPES] XML Schema Part 2: DataTypes. Available at http://www.w3.org/TR/xmlschema-2

[XML] XML 1.0 Recommendation from the W3C. Available at http://www.w3.org/TR/REC-xml

[XMLNS] XML Namespace specification. Available at http://18.29.1.23:80/TR/REC-xml-names

[XSLT] XSL Transformations. Available at http://www.w3.org/TR/xslt